Introducción al entorno de programación R y su aplicación en el análisis estadístico de datos

P. Agr. Ludwing Isaí Marroquín Jiménez

23 may 2025

Tabla de contenidos

Introducción

La estadística clásica constituye un pilar esencial en la investigación científica y en la toma de decisiones fundamentadas en datos. Este manual ha sido elaborado para quienes se inician en el análisis estadístico, con el objetivo de introducir de manera gradual y comprensible las herramientas fundamentales del lenguaje R, ampliamente reconocido en la ciencia de datos y la estadística aplicada. A lo largo del texto, se abordan desde los conceptos básicos hasta técnicas más avanzadas, acompañando la teoría con ejemplos prácticos que facilitan la comprensión y la aplicación en contextos reales. El propósito central es ofrecer una base sólida que permita a los lectores utilizar R de forma efectiva en el análisis de datos, sin requerir experiencia previa en programación o estadística (Ihaka & Gentleman, 1996; R Core Team, 2023).

Propósito del manual

El manual está diseñado para guiar a personas principiantes en el uso de R, abarcando desde la instalación y configuración del entorno hasta la aplicación de técnicas estadísticas clásicas y modernas. Se exploran temas como la manipulación y visualización de datos, la gestión de proyectos, la exportación de resultados y la realización de análisis estadísticos descriptivos e inferenciales. Cada tema se desarrolla con ejemplos prácticos y ejercicios que permiten aplicar los conocimientos adquiridos en situaciones reales. Además, se enfatiza el uso de R como una herramienta de código abierto, destacando su flexibilidad, capacidad de extensión y su papel en la promoción de la reproducibilidad científica (Ihaka & Gentleman, 1996; R Core Team, 2023).

A lo largo del manual, se presentan las principales características de R y su entorno de desarrollo integrado, RStudio, resaltando su utilidad tanto en proyectos académicos como profesionales. El texto está dirigido a estudiantes, investigadores y profesionales interesados en adquirir competencias en programación estadística, priorizando la claridad, la organización y la reproducibilidad en los análisis.

Organización del manual

El contenido del manual se estructura de manera progresiva, iniciando con los aspectos más elementales y avanzando hacia herramientas y técnicas estadísticas de mayor complejidad. Cada sección incluye explicaciones detalladas, ejemplos prácticos y ejercicios diseñados para consolidar el aprendizaje. Asimismo, se incorporan recomendaciones y buenas prácticas que facilitan la asimilación de los conceptos y fomentan la reproducibilidad en los análisis realizados.

El manual se organiza en las siguientes secciones:

1. **Introducción a R y RStudio:** En este capítulo se presentan los conceptos básicos de R y RStudio, incluyendo sus características principales, el proceso de instalación y configuración, así como la preparación del entorno de trabajo para el análisis de datos.
2. **Conceptos básicos de R:** Se abordan los fundamentos esenciales del lenguaje R, tales como los primeros pasos en la consola, la estructura de los datos, la importación de información, el uso de operadores, funciones y la gestión de paquetes.
3. **Manipulación de datos:** Este apartado introduce las técnicas fundamentales para la manipulación de datos en R, utilizando tanto las herramientas base del lenguaje como los paquetes dplyr y tidyr, permitiendo transformar, organizar y preparar los datos para su análisis.
4. **Visualización de datos:** Se exploran las distintas opciones para la visualización de datos, comenzando con las herramientas básicas de R y avanzando hacia la creación de gráficos personalizados mediante el paquete ggplot2, facilitando la interpretación y comunicación de los resultados.
5. **Gestión y exportación de resultados:** En este capítulo se explica cómo gestionar proyectos en R, exportar resultados de análisis, gráficos y tablas en formatos adecuados para su uso en informes y presentaciones, así como la integración con herramientas de control de versiones como Git y GitHub.
6. **Referencias:** Se proporcionan las referencias bibliográficas empleadas para la construcción de este manual, que le permiten al lector profundizar en el aprendizaje y la aplicación de R en distintos contextos estadísticos.
7. **Ejemplos de análisis estadístico con R:** Finalmente, se desarrollan ejemplos detallados de análisis estadístico, incluyendo estadística descriptiva, regresión lineal simple y múltiple, mostrando la aplicación práctica de R en el análisis de datos reales.

Cada capítulo está diseñado para ser independiente, permitiendo que los lectores avancen a su propio ritmo y consulten las secciones según sus necesidades.

Pre requisitos

Este manual no requiere conocimientos previos en programación ni en análisis estadístico. Está diseñado específicamente para principiantes, por lo que se parte desde cero, explicando cada concepto de manera clara y detallada. Todo lo que se necesita es:

1. **Interés por aprender**: La curiosidad y disposición para explorar un nuevo lenguaje de programación.
2. **Acceso a una computadora**: Con capacidad para instalar R y RStudio, herramientas que se explican paso a paso en el manual.
3. **Paciencia y práctica**: Como cualquier habilidad nueva, aprender R requiere tiempo y dedicación. Los ejemplos y ejercicios incluidos están diseñados para facilitar este proceso.

Con este enfoque, cualquier persona, independientemente de su experiencia previa, podrá utilizar este manual como una guía para iniciarse en el análisis estadístico con R.

Software y convenciones

La versión en línea de este manual está disponible en https://introduccion-r-cete.vercel.app/, y la fuente en español se encuentra alojada en el repositorio de GitHub https://github.com/Ludwing-MJ/introduccion_R_CETE. El desarrollo del manual se realizó utilizando Quarto, una herramienta que permite transformar archivos con extensión .qmd en formatos publicables como HTML, PDF y EPUB, facilitando la integración de código, resultados y texto en un solo documento reproducible.

Durante la elaboración del manual se emplearon diversos paquetes del ecosistema de R, entre los que destacan knitr y bookdown, los cuales permiten combinar las ventajas de LaTeX y R para la generación de documentos dinámicos y reproducibles (Xie et al., 2018). Esta integración posibilita que los ejemplos de código y los resultados presentados sean fácilmente replicables por el

A lo largo del manual, se presentan fragmentos de código que pueden ser copiados y ejecutados directamente en la consola de R para obtener los mismos resultados que se muestran en el texto. Los bloques de código se destacan en recuadros similares al siguiente:

```
4 + 6
a <- c(1, 5, 6)
5 * a
1:10
```

Los resultados generados por la ejecución de estos códigos se identifican con el numero uno encerrado entre cochetes (`[1]`) al inicio de cada línea, indicando que corresponden a la salida producida por R. Todo lo que comience con `[1]` representa resultados y no debe ser copiado como parte del código. Por ejemplo, al ejecutar el bloque anterior, se obtendrían los siguientes resultados:

```
[1] 10
```

```
[1]  5 25 30
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

Para garantizar la reproducibilidad y transparencia, se recomienda que el lector utilice versiones actualizadas de R y de los paquetes mencionados. La información sobre el entorno de desarrollo y las versiones de los paquetes utilizados en la construcción de este manual puede consultarse ejecutando el siguiente comando en R:

```
devtools::session_info()
```

```
Warning in system2("quarto", "-V", stdout = TRUE, env = paste0("TMPDIR=", : el
comando ejecutado '"quarto"
TMPDIR=C:/Users/FAUSAC/AppData/Local/Temp/RtmpuAe4nk/file1dc03bea44a4 -V' tiene
el estatus 1


- Session info -------------------------------------------------------------
 setting  value
 version  R version 4.4.3 (2025-02-28 ucrt)
 os       Windows 11 x64 (build 26100)
 system   x86_64, mingw32
 ui       RTerm
 language (EN)
 collate  Spanish_Guatemala.utf8
 ctype    Spanish_Guatemala.utf8
 tz       America/Guatemala
 date     2025-05-23
 pandoc   3.4 @ C:/Program Files/RStudio/resources/app/bin/quarto/bin/tools/ (via rmarkdo
 quarto   NA @ C:\\PROGRA~1\\Quarto\\bin\\quarto.exe

- Packages -----------------------------------------------------------------
 package     * version date (UTC) lib source
 cachem        1.1.0   2024-05-16 [1] CRAN (R 4.4.3)
 cli           3.6.5   2025-04-23 [1] CRAN (R 4.4.3)
 devtools      2.4.5   2022-10-11 [1] CRAN (R 4.4.3)
 digest        0.6.37  2024-08-19 [1] CRAN (R 4.4.3)
 ellipsis      0.3.2   2021-04-29 [1] CRAN (R 4.4.3)
 evaluate      1.0.3   2025-01-10 [1] CRAN (R 4.4.3)
 fastmap       1.2.0   2024-05-15 [1] CRAN (R 4.4.3)
 fs            1.6.6   2025-04-12 [1] CRAN (R 4.4.3)
 glue          1.8.0   2024-09-30 [1] CRAN (R 4.4.3)
 htmltools     0.5.8.1 2024-04-04 [1] CRAN (R 4.4.3)
 htmlwidgets   1.6.4   2023-12-06 [1] CRAN (R 4.4.3)
 httpuv        1.6.16  2025-04-16 [1] CRAN (R 4.4.3)
 jsonlite      2.0.0   2025-03-27 [1] CRAN (R 4.4.3)
 knitr         1.50    2025-03-16 [1] CRAN (R 4.4.3)
 later         1.4.2   2025-04-08 [1] CRAN (R 4.4.3)
 lifecycle     1.0.4   2023-11-07 [1] CRAN (R 4.4.3)
 magrittr      2.0.3   2022-03-30 [1] CRAN (R 4.4.3)
 memoise       2.0.1   2021-11-26 [1] CRAN (R 4.4.3)
 mime          0.13    2025-03-17 [1] CRAN (R 4.4.3)
 miniUI        0.1.2   2025-04-17 [1] CRAN (R 4.4.3)
 pkgbuild      1.4.7   2025-03-24 [1] CRAN (R 4.4.3)
 pkgload       1.4.0   2024-06-28 [1] CRAN (R 4.4.3)
 profvis       0.4.0   2024-09-20 [1] CRAN (R 4.4.3)
 promises      1.3.2   2024-11-28 [1] CRAN (R 4.4.3)
 purrr         1.0.4   2025-02-05 [1] CRAN (R 4.4.3)
 R6            2.6.1   2025-02-15 [1] CRAN (R 4.4.3)
 Rcpp          1.0.14  2025-01-12 [1] CRAN (R 4.4.3)
 remotes       2.5.0   2024-03-17 [1] CRAN (R 4.4.3)
```

```
 rlang         1.1.6   2025-04-11 [1] CRAN (R 4.4.3)
 rmarkdown     2.29    2024-11-04 [1] CRAN (R 4.4.3)
 rstudioapi    0.17.1  2024-10-22 [1] CRAN (R 4.4.3)
 sessioninfo   1.2.3   2025-02-05 [1] CRAN (R 4.4.3)
 shiny         1.10.0  2024-12-14 [1] CRAN (R 4.4.3)
 urlchecker    1.0.1   2021-11-30 [1] CRAN (R 4.4.3)
 usethis       3.1.0   2024-11-26 [1] CRAN (R 4.4.3)
 vctrs         0.6.5   2023-12-01 [1] CRAN (R 4.4.3)
 xfun          0.52    2025-04-02 [1] CRAN (R 4.4.3)
 xtable        1.8-4   2019-04-21 [1] CRAN (R 4.4.3)

 [1] C:/Users/FAUSAC/AppData/Local/R/win-library/4.4
 [2] C:/Program Files/R/R-4.4.3/library

------------------------------------------------------------------------------
```

Capítulo I

Introducción a R y RStudio

1 Conceptos básicos de R

1.1 ¿Qué es R?

R es un lenguaje de programación y un entorno computacional especializado en el análisis estadístico, la visualización de datos y la investigación científica. Su desarrollo fue iniciado en 1996 por Ross Ihaka y Robert Gentleman, quienes lo concibieron como una herramienta flexible y robusta para realizar análisis reproducibles y generar visualizaciones de alta calidad (Ihaka & Gentleman, 1996). Desde su creación, R ha evolucionado hasta convertirse en una de las plataformas más utilizadas en los ámbitos científico, académico y profesional, gracias a su capacidad de adaptación, su naturaleza de código abierto y el respaldo de una comunidad global activa.

1.1.1 Características principales de R

El lenguaje R se distingue principalmente por su orientación al análisis estadístico y científico de datos, abarcando desde pruebas estadísticas básicas, como t de Student y análisis de varianza (ANOVA), hasta modelos avanzados de regresión y análisis multivariado. Una de sus fortalezas más reconocidas es la capacidad de generar visualizaciones de datos de alta calidad mediante paquetes especializados, como ggplot2, que permiten explorar, interpretar y comunicar patrones complejos de manera efectiva (Wickham, 2016).

R es un software de código abierto, lo que significa que es gratuito y su desarrollo es impulsado por una comunidad internacional de usuarios y desarrolladores. Esta característica fomenta la colaboración, la transparencia y la mejora continua del entorno. La extensibilidad de R es notable, ya que cuenta con un repositorio central (CRAN) que, hasta 2023, alberga más de 19,000 paquetes, los cuales amplían sus capacidades para abordar tareas especializadas como análisis genómico, minería de texto, modelado espacial, entre otros (R Core Team, 2023).

Otra característica fundamental de R es su contribución a la reproducibilidad científica. El uso de scripts y cuadernos de trabajo permite documentar cada paso del análisis, facilitando la replicación y verificación de resultados por parte de otros investigadores. Además, R es altamente interoperable, permitiendo la integración con otros lenguajes de programación como Python, C++ y SQL, así como la importación y exportación de datos en múltiples formatos, incluyendo CSV, Excel, JSON y bases de datos relacionales (R Core Team, 2023).

1.1.2 ¿Por qué es especial R?

R trasciende su función como herramienta de cálculo estadístico, constituyéndose en un entorno integral para la manipulación, análisis y visualización de datos, así como para la automatización de flujos de trabajo analíticos. Su flexibilidad y capacidad de personalización lo convierten en una opción preferente para investigadores, analistas y profesionales de diversas disciplinas, quienes pueden adaptar el entorno a sus necesidades específicas (Grolemund & Wickham, 2017).

La vitalidad de la comunidad de usuarios y desarrolladores de R es un factor clave en su evolución. Esta comunidad no solo contribuye al desarrollo de nuevos paquetes y recursos, sino que también promueve la difusión de buenas prácticas y la formación continua, asegurando que R se mantenga a la vanguardia en el análisis de datos y análisis estadístico (R Core Team, 2023).

1.2 ¿Qué es RStudio?

RStudio es un Entorno de Desarrollo Integrado (IDE) diseñado para optimizar el trabajo con el lenguaje de programación R. Este entorno proporciona una interfaz intuitiva y organizada, compuesta por paneles que facilitan el acceso a las principales herramientas y funciones necesarias para el análisis estadístico y la visualización de datos. La estructura de RStudio permite a los usuarios gestionar de manera eficiente sus proyectos y recursos, promoviendo buenas prácticas en la organización y documentación del trabajo analítico (Xie et al., 2018).

1.2.1 Características principales de RStudio

Entre las características más destacadas de RStudio se encuentra su sistema de gestión de proyectos, el cual permite organizar archivos, scripts y conjuntos de datos en directorios independientes, favoreciendo la estructura y la reproducibilidad de los análisis. Esta funcionalidad resulta fundamental para mantener el orden y facilitar la colaboración en equipos de trabajo, ya que cada proyecto puede configurarse con su propio entorno y dependencias, lo que minimiza errores y mejora la trazabilidad de los procesos analíticos (Xie et al., 2018).

RStudio admite una amplia gama de formatos de datos, incluyendo CSV, Excel, HTML y bases de datos SQL, lo que facilita la importación y exportación de información desde diversas fuentes. Además, el entorno soporta la creación de gráficos interactivos y aplicaciones web mediante paquetes como shiny y plotly, ampliando las posibilidades de visualización y comunicación de resultados. La integración con paquetes de R es directa y eficiente, permitiendo instalar, actualizar y gestionar extensiones como ggplot2, dplyr y tidyr, lo que incrementa significativamente las capacidades analíticas del entorno (Xie et al., 2018; Wickham, 2016).

Este IDE es compatible con los principales sistemas operativos, como Windows, macOS y Linux, lo que garantiza su accesibilidad para una amplia variedad de usuarios. Asimismo, RStudio ofrece opciones de personalización, permitiendo modificar la apariencia, los atajos de teclado y la disposición de los paneles, así como integrar herramientas externas como Git para el control de versiones y la gestión colaborativa de proyectos (Xie et al., 2018).

1.2.2 Beneficios de usar RStudio

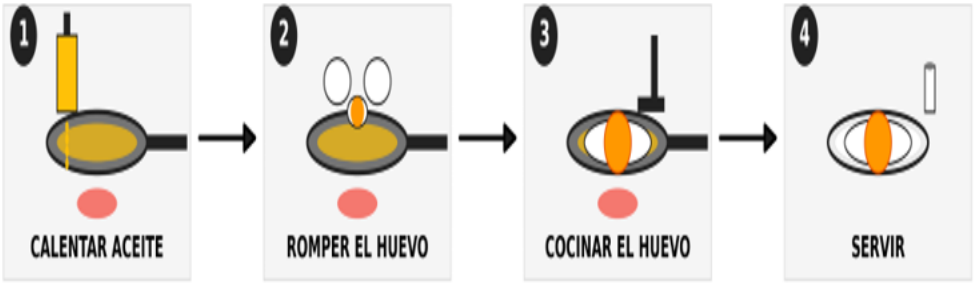
El uso de RStudio proporciona ventajas sustanciales en el proceso de análisis estadístico de datos. Este entorno incrementa la eficiencia al facilitar la organización y agilizar las tareas analíticas, y promueve la reproducibilidad mediante herramientas como R Markdown y el sistema de proyectos. Su interfaz gráfica resulta accesible tanto para usuarios principiantes como avanzados, permitiendo trabajar con datos, gráficos, modelos estadísticos y aplicaciones interactivas en un solo entorno. La flexibilidad de RStudio lo convierte en una herramienta adaptable a diversas necesidades analíticas, consolidándose como la opción preferente para quienes buscan un entorno de trabajo integral y eficiente (Xie et al., 2018).

1.3 Reproducibilidad y replicabilidad en la investigación científica

La reproducibilidad y la replicabilidad son pilares fundamentales en la investigación científica contemporánea, ya que garantizan la validez y la confiabilidad de los hallazgos. Numerosos estudios han evidenciado que una proporción considerable de investigadores experimenta dificultades para replicar experimentos previos, principalmente debido a la insuficiente documentación de los procedimientos y análisis originales (Baker, 2016). El uso de herramientas tradicionales como Excel o Infostat puede limitar la transparencia, ya que parte de los cálculos se realiza de manera interna y los gráficos suelen requerir modificaciones manuales, lo que dificulta la reproducción exacta de los resultados y la verificación independiente de los análisis.

1.3.1 El papel de R en la reproducibilidad

R se caracteriza por su capacidad para documentar de manera precisa y estructurada cada etapa del análisis a través de scripts, lo que permite que los procedimientos sean replicados y reinterpretados en el futuro, incrementando la transparencia y la credibilidad científica (Gentleman & Temple Lang, 2007). Esta documentación detallada facilita la reutilización de métodos en nuevos estudios, optimizando tanto el tiempo como los recursos disponibles. Un script en R puede ser comparado con una receta detallada, en la que cada paso está claramente especificado y puede adaptarse a diferentes conjuntos de datos según las necesidades del análisis, permitiendo así la replicación exacta o la adaptación a nuevos contextos (Xie et al., 2018).



1.3.2 Definición y características de la reproducibilidad

La reproducibilidad se define como la capacidad de obtener los mismos resultados utilizando los mismos datos y métodos empleados en el análisis original. Este principio es esencial para la verificación y validación de los hallazgos científicos, ya que permite que otros investigadores, o el propio autor, puedan replicar los resultados siempre que dispongan de los datos y procedimientos originales (National Academies of Sciences, Engineering, and Medicine, 2019). Para alcanzar la reproducibilidad, es imprescindible contar con acceso a los datos originales y una documentación exhaustiva de los métodos utilizados, asegurando que los resultados sean consistentes al repetir el análisis. La reproducibilidad fomenta la transparencia, facilita la verificación de los resultados y promueve la colaboración científica, ya que otros investigadores pueden comprender y construir sobre el trabajo existente (Wilkinson et al., 2016).

1.3.3 Definición y características de la replicabilidad

Por otro lado, la replicabilidad se refiere a la obtención de resultados consistentes al realizar un estudio similar en un contexto diferente, utilizando nuevos datos o métodos ajustados. Este concepto evalúa la capacidad de generalización de los hallazgos y su aplicabilidad en distintos escenarios, lo que resulta fundamental para validar la robustez de los resultados científicos (National Academies of Sciences, Engineering, and Medicine, 2019). La replicabilidad implica el uso de datos diferentes, la adaptación de los métodos y la obtención de resultados coherentes con los del estudio original, aunque no necesariamente idénticos. Este proceso permite evaluar la generalización de los resultados, refuerza la credibilidad científica y facilita la extensión del conocimiento a nuevas aplicaciones o contextos (The Turing Way Community, 2023).

1.3.4 Beneficios de utilizar R para la ciencia reproducible

El uso de R en la investigación científica ofrece ventajas significativas para la reproducibilidad y la replicabilidad. El código generado en R es accesible para la revisión por pares, lo que incrementa la transparencia de los análisis y permite la identificación de posibles errores o mejoras (The Turing Way Community, 2023). Además, los métodos desarrollados pueden ser reutilizados y adaptados en nuevos estudios, optimizando recursos y tiempo (Gentleman & Temple Lang, 2007). R también facilita el cumplimiento de los principios FAIR (Findable, Accessible, Interoperable, Reusable), promoviendo una gestión adecuada y responsable de los datos científicos, lo que contribuye a la apertura y reutilización de los resultados de investigación (Wilkinson et al., 2016).

2 Instalación y configuración

Antes de comenzar a trabajar con R y RStudio, es fundamental realizar la instalación y configuración de ambos programas. R es un lenguaje de programación y entorno computacional ampliamente utilizado en el análisis estadístico, la visualización de datos y la investigación reproducible (Ihaka & Gentleman, 1996). Por su parte, RStudio es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) diseñado específicamente para trabajar con R, proporcionando una interfaz amigable y herramientas avanzadas que optimizan el flujo de trabajo (Xie et al., 2018). Este capítulo detalla los pasos necesarios para descargar, instalar y configurar ambos programas, asegurando un entorno de trabajo funcional y eficiente.

2.1 Descarga de R y RStudio

Para utilizar R y RStudio, es necesario descargar ambos programas de sus sitios oficiales. R proporciona el núcleo del lenguaje y las herramientas computacionales fundamentales, mientras que RStudio actúa como una interfaz que simplifica el uso de R y mejora la experiencia del usuario, integrando funciones para la gestión de proyectos, edición de scripts y visualización de resultados (Xie et al., 2018).

2.1.1 Descarga de R

Se recomienda descargar una versión estable de R para evitar posibles incompatibilidades con paquetes que aún no han sido actualizados para las versiones más recientes. Por ejemplo, la versión R 4.4.3 es reconocida por su estabilidad y amplio soporte dentro de la comunidad de usuarios (R Core Team, 2023). El repositorio oficial de R se encuentra disponible en https://cran.r-project.org/bin/windows/base/old/, donde es posible acceder a todas las versiones publicadas. Para descargar una versión específica, se debe seleccionar el nombre de la versión deseada y hacer clic en el archivo con terminación -win.exe, lo que iniciará la descarga del instalador correspondiente (R Core Team, 2023).

2.1.2 Descarga de RStudio

La descarga de RStudio se realiza desde su página oficial, donde se encuentra disponible la versión más reciente para los principales sistemas operativos. Para usuarios de Windows, se debe seleccionar la opción “Download RStudio Desktop for Windows”, mientras que para quienes utilizan macOS o Linux, la misma página ofrece las versiones correspondientes para estos sistemas (Xie et al., 2018). Es importante asegurarse de descargar la versión adecuada según el sistema operativo del equipo para garantizar la compatibilidad y el correcto funcionamiento del entorno.

2.2 Instalación de R y RStudio

La instalación de R y RStudio debe realizarse siguiendo un orden específico para evitar conflictos y asegurar que ambos programas funcionen correctamente. A continuación, se describen los pasos detallados para cada uno:

2.2.1 Instalación de R

Una vez descargado el instalador de R, se debe ejecutar el archivo .exe y seguir las instrucciones proporcionadas por el asistente de instalación. En la mayoría de los casos, es suficiente con aceptar las configuraciones predeterminadas, a menos que se requiera una configuración personalizada para necesidades específicas del usuario o del proyecto (R Core Team, 2023).

2.2.2 Instalación de RStudio

Después de instalar R, se procede a ejecutar el instalador de RStudio previamente descargado. Al igual que en el caso de R, se pueden aceptar las opciones predeterminadas durante la instalación. Es relevante destacar que RStudio permite gestionar múltiples versiones de R en un mismo dispositivo, lo que resulta especialmente útil para trabajar en proyectos que requieren versiones específicas del lenguaje. Esta selección puede realizarse desde la configuración de RStudio, facilitando así la administración de entornos de trabajo diferenciados (Xie et al., 2018; R Core Team, 2023).

2.3 Configuración inicial

Tras completar la instalación de R y RStudio, es recomendable realizar una configuración inicial que permita personalizar el entorno de trabajo, mejorar la organización y facilitar el desarrollo de análisis estadísticos. Estas configuraciones contribuyen a optimizar la experiencia del usuario y a establecer un flujo de trabajo más eficiente y productivo (Xie et al., 2018). A continuación, se describen los pasos esenciales para configurar RStudio de manera adecuada.

2.3.1 Seleccionar la versión de R

RStudio permite elegir la versión de R que se utilizará, lo cual es especialmente útil si se tienen múltiples versiones instaladas en el mismo dispositivo. Esta funcionalidad garantiza la compatibilidad con proyectos que requieren versiones específicas del lenguaje (R Core Team, 2023). Para seleccionar la versión de R en RStudio, se deben seguir estos pasos:

1. Ir al menú **Tools** y seleccionar **Global Options**.
2. En la ventana emergente, dirigirse a la pestaña **General**.
3. En el apartado **R version**, elegir la versión deseada de R.

2.3.2 Configurar la apariencia de RStudio

RStudio ofrece opciones de personalización para adaptar su apariencia a las preferencias del usuario, lo que puede mejorar la experiencia de trabajo y reducir la fatiga visual durante sesiones prolongadas (Xie et al., 2018). Para cambiar el tema de la interfaz y ajustar la fuente, se deben seguir los siguientes pasos:

1. Acceder al menú **Tools** y seleccionar **Global Options**.
2. En la ventana emergente, ir a la pestaña **Appearance**.
3. Elegir el tema preferido, ya sea claro u oscuro (por ejemplo, el tema Cobalt para reducir la fatiga visual).
4. Ajustar el tamaño y el tipo de fuente según las preferencias personales.

2.3.3 Configurar el panel de trabajo

La interfaz de RStudio está organizada en cuatro paneles principales: editor de scripts, consola, entorno/archivos y gráficos/ayuda. Estos paneles pueden reorganizarse para optimizar el flujo de trabajo. Para modificar la disposición de los paneles, se deben seguir estos pasos:

1. Ir al menú **Tools** y seleccionar **Global Options**.
2. Acceder a la sección **Pane Layout**.
3. Ajustar la ubicación de los paneles según las necesidades, por ejemplo, colocando el editor de scripts en la parte superior izquierda y la consola en la parte inferior.
4. Guardar los cambios para aplicar la nueva disposición.

2.3.4 Habilitar el número de líneas en el editor de scripts

La numeración de líneas en el editor de scripts facilita la navegación y depuración del código. Para habilitar esta opción, se deben seguir los siguientes pasos:

1. Acceder al menú **Tools** y seleccionar **Global Options**.
2. Ir a la pestaña **Code** y luego a **Display**.
3. Marcar la casilla **Show line numbers** para activar la numeración de líneas.

2.4 Organización de proyectos

La organización adecuada de proyectos en RStudio es esencial para establecer un flujo de trabajo eficiente, reproducible y estructurado. Una gestión ordenada de archivos y scripts no solo facilita el desarrollo de los análisis, sino que también mejora la colaboración y la reproducibilidad de los resultados (Xie et al., 2018).

2.4.1 Crear un proyecto en RStudio

Para organizar los archivos, datos y scripts de un análisis específico, RStudio permite crear proyectos siguiendo estos pasos:

1. En la barra de menú, seleccionar **File > New Project**.
2. Elegir una de las siguientes opciones:

   New Directory: para crear un proyecto desde cero en una nueva carpeta.

   Existing Directory: para convertir una carpeta existente en un proyecto de RStudio.

   Version Control: para clonar un repositorio de Git y trabajar en un proyecto con control de versiones.
3. Configurar el nombre y la ubicación del proyecto según las necesidades del análisis.
4. Hacer clic en **Create Project** para finalizar la configuración.

El uso de proyectos en RStudio permite mantener una estructura clara y organizada, facilitando la gestión de los recursos necesarios para el análisis y promoviendo la reproducibilidad (Xie et al., 2018).

2.4.2 Establecer un directorio de trabajo

El directorio de trabajo es la carpeta donde R buscará los archivos y guardará los resultados generados durante el análisis. Para establecerlo manualmente, se puede utilizar la función `setwd()`, como se muestra a continuación:

```
# Establecer directorio de trabajo
setwd("ruta/del/directorio")
```

Sin embargo, al trabajar con proyectos en RStudio, el directorio de trabajo se configura automáticamente al abrir el archivo del proyecto, lo que elimina la necesidad de establecerlo manualmente y reduce errores relacionados con rutas incorrectas (R Core Team, 2023).

2.4.3 Uso de archivos .Rproj

El archivo `.Rproj` es el elemento central de cada proyecto en RStudio. Este archivo almacena las configuraciones específicas del proyecto, como el directorio de trabajo, las opciones de visualización y otros ajustes personalizados. Al abrir un archivo `.Rproj`, se carga automáticamente el entorno de trabajo asociado, lo que facilita la continuidad y la gestión del análisis (Xie et al., 2018).

2.4.4 Beneficios de la organización de proyectos

La correcta organización de proyectos en RStudio ofrece varios beneficios clave:

1. **Reproducibilidad**: Facilita que otros usuarios (o el propio usuario en el futuro) comprendan y reproduzcan el análisis, asegurando que los resultados sean consistentes.
2. **Eficiencia**: Reduce el tiempo perdido buscando archivos o configurando rutas manualmente, permitiendo un enfoque más directo en el análisis.
3. **Colaboración**: Mejora la comunicación y el trabajo en equipo al mantener una estructura clara y consistente, especialmente en proyectos compartidos.
4. **Optimización del flujo de trabajo**: La combinación de una apariencia personalizada y una estructura organizada permite al usuario enfocarse en el análisis de datos de manera más eficiente y profesional (Xie et al., 2018).

Capítulo II

Conceptos básicos de R

3 Primeros pasos en R

Iniciar el trabajo en R y RStudio puede resultar desafiante para quienes no están familiarizados con estos entornos, pero una orientación adecuada facilita considerablemente el proceso. Esta sección guía al usuario en los aspectos fundamentales para comenzar a programar en R, desde la creación de scripts hasta la comprensión de los objetos básicos del lenguaje. Estos conocimientos iniciales son esenciales para establecer un flujo de trabajo eficiente y reproducible (Xie et al., 2018).

3.1 Creación de scripts en RStudio

El script es el archivo principal donde se escribe, guarda y ejecuta el código en R. Utilizar scripts no solo permite desarrollar análisis de datos, sino también documentar cada paso del proceso, lo que contribuye a la reproducibilidad y la organización del trabajo (Xie et al., 2018). Para crear un script en RStudio, se pueden seguir los siguientes métodos:

1. Desde la barra de menú, seleccionar **File > New File > R Script**.
2. Utilizar el atajo de teclado **Ctrl + Shift + N** para abrir un nuevo script de manera rápida.

Una vez creado, el script se convierte en el espacio central para el desarrollo de los análisis. Es recomendable guardar el archivo desde el inicio, utilizando **File > Save** o el atajo **Ctrl + S**, para evitar la pérdida de información y facilitar la gestión de versiones (Xie et al., 2018).

3.2 Guardado y organización de archivos

Una gestión adecuada de los archivos es esencial para mantener la eficiencia y la reproducibilidad en el trabajo con R y RStudio. El uso de nombres descriptivos, la organización en carpetas y la documentación clara de los scripts contribuyen a un entorno de trabajo ordenado y profesional (Xie et al., 2018).

3.2.1 Guardado de scripts y archivos

Para guardar un script en RStudio, se deben seguir estos pasos:

1. Seleccionar la opción **File > Save As...** en la barra de menú.
2. Definir la ubicación y el nombre del archivo en la ventana emergente.

3. Utilizar nombres que reflejen el contenido y propósito del archivo, por ejemplo: `analisis_rendimiento.R` para scripts o `datos_suelo_2023.csv` para archivos de datos.
4. Evitar espacios y caracteres especiales en los nombres, empleando guiones bajos (_) o guiones medios (-) para separar palabras.
5. Incluir fechas en formato estándar (YYYY-MM-DD) para facilitar la identificación de versiones, como en `2023-10-15_importacion_datos.R`.

Estas prácticas previenen errores en la ejecución del código y facilitan la gestión de versiones y actualizaciones (Xie et al., 2018).

3.2.2 Organización de directorios y proyectos

La estructura de carpetas es clave para mantener el orden en los proyectos. Para organizar los archivos de manera eficiente, se recomienda:

1. Crear una carpeta específica para cada proyecto, agrupando en ella todos los scripts, datos y resultados relacionados.
2. Utilizar archivos de proyecto `.Rproj`, ya que RStudio configura automáticamente el directorio de trabajo al abrir el proyecto, simplificando la gestión de archivos y reduciendo errores asociados a rutas incorrectas (R Core Team, 2023).

3.2.3 Buenas prácticas para la organización de archivos

Para optimizar la organización y facilitar la colaboración, se aconseja:

1. Estandarizar los nombres de archivos, siguiendo un formato uniforme y descriptivo.
2. Documentar los pasos del análisis mediante comentarios claros en los scripts, lo que ayuda a comprender y reproducir el trabajo en el futuro.
3. Utilizar proyectos de RStudio (`.Rproj`) para asegurar que el entorno de trabajo esté correctamente configurado y todos los archivos relevantes se encuentren en la misma ubicación.
4. Realizar copias de seguridad periódicas, ya sea mediante sistemas de control de versiones como Git o almacenando archivos importantes en ubicaciones seguras.

La aplicación de estas prácticas contribuye a un flujo de trabajo más eficiente, facilita la colaboración y asegura la reproducibilidad de los análisis realizados en RStudio (Xie et al., 2018; R Core Team, 2023).

3.3 Introducción a los objetos en R

En R, la manipulación y el análisis de datos se basan en el uso de objetos, que son entidades capaces de almacenar información de diversos tipos. Cada objeto tiene un nombre, un tipo de dato y, en algunos casos, dimensiones o atributos adicionales. Esta estructura flexible permite organizar y gestionar datos de manera eficiente, lo que convierte a los objetos en el elemento central del trabajo en R (R Core Team, 2023).

3.3.1 Creación de objetos en R

Para crear un objeto en R, se utiliza un operador de asignación. Aunque existen dos opciones (`=` y `<-`), la convención más extendida y recomendada es el uso de `<-`, ya que mejora la legibilidad y sigue las normas del lenguaje (Ihaka & Gentleman, 1996). El valor o expresión a la derecha del operador se asigna al nombre del objeto a la izquierda.

```
# Asignación de un valor numérico a un objeto
x <- 10  # El objeto x almacena el valor 10

# Asignación de un texto a un objeto
nombre <- "Ana"  # El objeto nombre almacena la cadena de texto "Ana"
```

Este método facilita la organización y claridad del código, permitiendo reutilizar y modificar los valores almacenados en los objetos según las necesidades del análisis.

3.3.2 Buenas prácticas y documentación

La claridad en el código es fundamental para la reproducibilidad y el mantenimiento de los análisis. En R, los comentarios se introducen utilizando el símbolo numeral `#`. Los comentarios no son ejecutados por el intérprete y sirven para explicar el propósito de cada línea o bloque de código, facilitando la comprensión tanto para el autor como para otros usuarios. Por ejemplo:

```
# Este es un comentario que explica la siguiente línea
y <- 5  # Asigna el valor 5 al objeto y
```

Se recomienda documentar los scripts de manera clara y consistente, describiendo los pasos principales y las decisiones tomadas durante el análisis. Esta práctica es esencial para la colaboración y para la revisión futura del trabajo (Ihaka & Gentleman, 1996).

3.4 Tipos principales de objetos en R

R permite trabajar con diferentes tipos de objetos, cada uno diseñado para almacenar y manipular distintos tipos de datos. Los más comunes son los siguientes (R Core Team, 2023):

3.4.1 Objetos Numéricos

Los objetos numéricos son fundamentales en R y pueden almacenar tanto números enteros como decimales (también llamados de punto flotante). Existen dos subtipos principales:

1. **Números enteros (integer)**: Almacenan valores sin decimales
2. **Números de punto flotante (double)**: Almacenan valores con decimales

Ejemplo de creación de objetos numéricos:

```
# Creación de objetos numéricos
edad <- 21        # Número entero
altura_m <- 1.70  # Número decimal (punto flotante)
peso_lb <- 145    # Número entero

# Exploración de objetos numéricos
class(edad)      # Muestra "numeric"
```

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

```
typeof(altura_m)  # Muestra "double"
```

```
[1] "double"
```

```
str(peso_lb)     # Muestra la estructura completa
```

```
 num 145
```

```
# Operaciones matemáticas básicas
imc <- peso_lb/2.2046/(altura_m^2)  # Cálculo del IMC
print(imc)  # Muestra el resultado del cálculo
```

```
[1] 22.75833
```

Los objetos numéricos permiten realizar operaciones matemáticas y son esenciales en análisis estadísticos, cálculos y modelado de datos.

3.4.2 Objetos de Texto

Los objetos de texto, también conocidos como objetos de tipo carácter, almacenan cadenas de texto. Estos se escriben entre comillas dobles (") o simples ('). Son útiles para representar información cualitativa, como nombres, descripciones o etiquetas.

Ejemplo de creación de objetos de texto:

```r
# Creación de objetos de texto
nombre <- "Juan"           # Usando comillas dobles
apellido <- 'García'       # Usando comillas simples
nombre_completo <- paste(nombre, apellido)  # Concatenación de textos

# Exploración de objetos de texto
class(nombre)          # Muestra "character"
```

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

```r
str(nombre_completo)   # Muestra la estructura
```

```
 chr "Juan García"
```

```r
nchar(nombre)          # Muestra el número de caracteres
```

```
[1] 4
```

```r
# Manipulación de texto
nombre_mayusculas <- toupper(nombre)  # Convierte a mayúsculas
print(nombre_mayusculas)
```

```
[1] "JUAN"
```

3.4.3 Objetos de Tipo Factor

Los objetos de tipo factor se utilizan para almacenar variables categóricas con niveles definidos. Estos niveles representan categorías discretas, como escalas, estados o clasificaciones. Los factores son especialmente útiles en análisis estadísticos, ya que permiten manejar variables categóricas de manera eficiente.

Ejemplo de creación de objetos tipo factor:

```r
# Creación de factores simples
estado_civil <- factor("soltero")
sexo <- factor("masculino")

# Creación de factores con múltiples niveles
estado_civil <- factor("soltero",
                       levels = c("soltero", "casado", "divorciado"))
nivel_educativo <- factor("licenciatura",
                          levels = c("bachillerato",
                                     "licenciatura",
                                     "posgrado"),
                          ordered = TRUE)  # Factor ordenado

# Exploración de factores
class(estado_civil)     # Muestra "factor"
```

```
[1] "factor"
```

```
levels(estado_civil)    # Muestra los niveles definidos
```

```
[1] "soltero"    "casado"     "divorciado"
```

```
str(nivel_educativo)    # Muestra la estructura completa
```

```
 Ord.factor w/ 3 levels "bachillerato"<..: 2
```

```
is.ordered(nivel_educativo)  # Verifica si el factor es ordenado
```

```
[1] TRUE
```

Los factores son especialmente útiles en análisis estadísticos porque:

1. Permiten especificar un orden en las categorías.
2. Facilitan la creación de gráficos categóricos.
3. Son esenciales en modelos estadísticos.

3.4.4 Objetos Lógicos

Los objetos lógicos almacenan valores `TRUE` o `FALSE`, que resultan de comparaciones lógicas. Estos objetos son esenciales para realizar análisis condicionales, aplicar filtros y evaluaciones condicionales.

Ejemplo de creación de objetos lógicos:

```
# Creación de objetos lógicos mediante comparaciones
mayoria_de_edad <- edad >= 18


# Exploración de objetos lógicos
class(mayoria_de_edad)   # Muestra "logical"
```

```
[1] "logical"
```

```
str(mayoria_de_edad)      # Muestra la estructura
```

```
 logi TRUE
```

Los objetos lógicos son esenciales para:

1. Filtrar datos basados en condiciones.
2. Realizar evaluaciones condicionales.
3. Crear subconjuntos de datos.
4. Programación de control de flujo (if/else).

3.5 Exploración de objetos: funciones útiles

Una vez creados los objetos, es importante poder identificar su naturaleza y estructura. R ofrece varias funciones para este propósito:

1. `class()`: Devuelve la clase del objeto, como "numeric", "character", "factor" o "logical".
2. `typeof()`: Indica el tipo interno de almacenamiento del objeto, como "double", "integer" o "character".
3. `str()`: Muestra la estructura interna del objeto, proporcionando un resumen compacto de su contenido.
4. `levels()`: Específica para factores, devuelve los niveles o categorías posibles del objeto.

Ejemplo de uso de estas funciones:

```
# Exploración de un objeto numérico
class(x)      # Devuelve "numeric"
typeof(x)     # Devuelve "double"
str(x)        # Muestra la estructura del objeto

# Exploración de un objeto de texto
class(nombre) # Devuelve "character"
str(nombre)   # Muestra la estructura del objeto

# Exploración de un factor
estado_civil <- factor("soltero", levels = c("soltero",
                                             "casado",
                                             "divorciado"))
class(estado_civil)   # Devuelve "factor"
typeof(estado_civil)  # Devuelve "integer"
str(estado_civil)     # Muestra la estructura del factor
levels(estado_civil)  # Devuelve los niveles posibles
```

4 Estructuras de datos en R

Las estructuras de datos representan el pilar esencial para el análisis estadístico y científico en el entorno R, ya que posibilitan la organización, almacenamiento y manipulación de información de manera sistemática y eficiente. Estas estructuras permiten gestionar desde datos simples, como valores individuales, hasta conjuntos complejos y heterogéneos, adaptándose a los requerimientos de diversos tipos de análisis y facilitando la reproducibilidad de los resultados (Ihaka & Gentleman, 1996; R Core Team, 2023).

En R, las principales estructuras de datos incluyen los vectores, matrices, data frames y listas. Cada una de estas estructuras ha sido diseñada para resolver problemas específicos y se adapta a diferentes escenarios analíticos. Los vectores constituyen la unidad básica y homogénea de almacenamiento, mientras que las matrices permiten organizar datos en dos dimensiones bajo la restricción de homogeneidad de tipo. Los data frames, por su parte, ofrecen una estructura tabular flexible, capaz de contener columnas de distintos tipos de datos, lo que resulta especialmente útil en el análisis de datos reales y heterogéneos. Finalmente, las listas proporcionan una solución versátil para almacenar colecciones de objetos de diferentes tipos y longitudes, facilitando la gestión de resultados complejos y la integración de diversas fuentes de información (R Core Team, 2023; Wickham & Grolemund, 2017).

4.1 Vectores

En el lenguaje R, los vectores constituyen la estructura de datos más elemental y versátil, sirviendo como base para la construcción de estructuras más complejas, tales como matrices y data frames. Un vector se define como una secuencia ordenada y unidimensional de elementos que comparten el mismo tipo de dato, ya sea numérico, de texto (caracter), o lógico. Esta homogeneidad en el tipo de datos asegura tanto la eficiencia computacional como la coherencia en las operaciones analíticas, facilitando la manipulación y el análisis de grandes volúmenes de información (Ihaka & Gentleman, 1996; R Core Team, 2023).

4.1.1 Tipos de Vectores y su creación

La función principal para la creación de vectores en R es `c()`, abreviatura de "concatenar". Esta función permite agrupar elementos individuales o incluso otros vectores en una sola estructura (Wickham & Grolemund, 2017). Los tipos de vectores más comunes incluyen los numéricos, de texto y lógicos, como se ilustra a continuación:

```r
# Vectores numéricos
edades <- c(17, 20, 18, 25)          # Enteros
alturas <- c(1.75, 1.68, 1.82, 1.65) # Decimales

# Vectores de texto (character)
nombres <- c("Juan", "Ana", "Luis", "María")

# Vectores lógicos
# Creados usando la función c()
mayores_de_edad <- c(FALSE, TRUE, TRUE, TRUE)
# O mediante una comparación empleando operadores lógicos
mayores_de_edad <- edades >= 18
```

En estos ejemplos, el vector `edades` almacena valores numéricos, `nombres` contiene cadenas de texto, y `mayores_de_edad` almacena valores lógicos (TRUE o FALSE) derivados de una comparación. Esta flexibilidad permite adaptar los vectores a diversas necesidades analíticas (Grolemund & Wickham, 2017).

4.1.2 Coerción de Tipos de Datos en Vectores

Una característica fundamental de los vectores en R es la coerción automática de tipos de datos. Cuando se intenta combinar elementos de diferentes tipos en un mismo vector, R convierte todos los elementos al tipo más general que pueda contener a todos ellos, siguiendo una jerarquía: carácter > numérico > lógico. Por ejemplo:

```r
# Mezcla de números y texto
vector_mixto <- c(1, 2, "tres")
# Resultado: "1" "2" "tres"
```

En este caso, todos los elementos se convierten a texto (character), ya que es el tipo más general capaz de representar cualquier valor. Este comportamiento, conocido como coerción implícita, es esencial para evitar errores en la manipulación de datos, pero requiere atención para no perder información relevante o introducir inconsistencias (R Core Team, 2023; Grolemund & Wickham, 2017).

4.1.3 Operaciones con Vectores

Los vectores en R permiten realizar una amplia gama de operaciones matemáticas, lógicas y de manipulación de datos, fundamentales para el análisis estadístico y la transformación de información. A continuación, se describen las operaciones más comunes:

4.1.3.1 Acceso a elementos específicos

El acceso a elementos individuales o múltiples de un vector se realiza mediante índices entre corchetes, comenzando en 1:

```r
# Acceder a elementos individuales
primer_nombre <- nombres[1]    # "Juan"
ultima_edad <- edades[4]       # 25

# Acceder a múltiples elementos
nombres_seleccionados <- nombres[c(1, 3)]  # "Juan" "Luis"
```

4.1.3.2 Filtrado de elementos

El filtrado de elementos se logra aplicando condiciones lógicas, lo que permite seleccionar subconjuntos de datos de manera eficiente:

```r
# Filtrar personas mayores de 20 años
mayores_20 <- edades[edades > 20]

# Obtener nombres de personas mayores de 20
nombres_mayores_20 <- nombres[edades > 20]
```

Aquí, la condición `edades > 20` genera un vector lógico que selecciona únicamente los valores que cumplen el criterio especificado (Field, 2013).

4.1.3.3 Combinación de vectores

La función `c()` también permite combinar varios vectores en uno solo:

```r
# Combinar dos vectores
nuevo_vector <- c(edades, c(22, 21))
nuevo_vector
```

```
[1] 17 20 18 25 22 21
```

Aquí, el vector `nuevo_vector` combina los elementos del vector `edades` con los valores 22 y 21, generando un nuevo vector.

4.1.3.4 Funciones Útiles para Vectores

R ofrece una variedad de funciones para analizar y manipular vectores, tales como:

```r
# Estadísticas básicas
promedio_edades <- mean(edades)       # Media
edad_maxima <- max(edades)            # Valor máximo
edad_minima <- min(edades)            # Valor mínimo
total_elementos <- length(edades)      # Número de elementos

# Ordenamiento
edades_ordenadas <- sort(edades)      # Orden ascendente
edades_descendente <- sort(edades, decreasing = TRUE)  # Orden descendente
```

4.2 Matrices

Las matrices en R representan estructuras de datos bidimensionales que organizan información en filas y columnas, manteniendo la homogeneidad en el tipo de datos (numérico, lógico o de texto). Esta característica fundamental garantiza la integridad y eficiencia en las operaciones matemáticas y estadísticas, siendo particularmente relevantes en el análisis multivariado y la computación científica (Ihaka & Gentleman, 1996; Venables & Ripley, 2002).

A diferencia de los vectores unidimensionales, las matrices proporcionan un marco más sofisticado para la representación y manipulación de datos estructurados, facilitando la implementación de algoritmos estadísticos complejos y el análisis de datos experimentales (Montgomery et al., 2012).

4.2.1 Creación de Matrices y Argumentos de la Función matrix()

La función principal para la creación de matrices en R es `matrix()`. Sus argumentos más relevantes son:

1. **data**: Vector de datos a organizar en la matriz. Es el único argumento obligatorio.
2. **nrow**: Número de filas de la matriz. Opcional; si se omite, R lo infiere a partir de la longitud de `data` y el valor de `ncol`.
3. **ncol**: Número de columnas de la matriz. Opcional; si se omite, R lo infiere a partir de la longitud de `data` y el valor de `nrow`.
4. **byrow**: Lógico. Indica si los datos se llenan por filas (`TRUE`) o por columnas (`FALSE`, valor por defecto).
5. **dimnames**: Lista de dos vectores de caracteres para asignar nombres a filas y columnas.

El comportamiento de estos argumentos se ilustra en los siguientes ejemplos:

```r
# Ejemplo 1: Solo se especifica data y nrow
# R calcula automáticamente el número de columnas
matriz1 <- matrix(1:6, nrow = 2)
print(matriz1)
```

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

```r
# Ejemplo 2: Solo se especifica data y ncol
# R calcula automáticamente el número de filas
matriz2 <- matrix(1:6, ncol = 2)
print(matriz2)
```

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

```r
# Ejemplo 3: Se especifican nrow y ncol
matriz3 <- matrix(1:6, nrow = 2, ncol = 3)
print(matriz3)
```

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

```r
# Ejemplo 4: Uso del argumento byrow
matriz4 <- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
print(matriz4)
```

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

```r
# Ejemplo 5: Asignación de nombres a filas y columnas con dimnames
matriz5 <- matrix(1:4, nrow = 2,
                  dimnames = list(c("Fila1", "Fila2"),
                                  c("Col1", "Col2")))
print(matriz5)
```

```
      Col1 Col2
Fila1    1    3
Fila2    2    4
```

Si la longitud del vector `data` no coincide exactamente con el producto de `nrow` y `ncol`, R reciclará los valores del vector para completar la matriz. Este comportamiento, conocido como reciclaje de datos, puede ser útil en ciertos contextos, pero también puede generar resultados inesperados si no se verifica la consistencia de los datos (R Core Team, 2023).

```r
# Ejemplo de reciclaje de datos
matriz6 <- matrix(1:3, nrow = 2, ncol = 4)
print(matriz6) # R repite los valores de 1:3 hasta llenar la matriz
```

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

4.2.2 Propiedades y Atributos de las Matrices

Las matrices en R poseen atributos específicos que pueden ser consultados y modificados mediante funciones especializadas. Es posible obtener y modificar las dimensiones, así como asignar nombres a filas y columnas para mejorar la legibilidad y trazabilidad de los datos (Venables & Ripley, 2002; Grolemund & Wickham, 2017).

```
# Dimensiones de la matriz
dim(matriz1)      # Devuelve (filas, columnas)
```

```
[1] 2 3
```

```
nrow(matriz1)     # Número de filas
```

```
[1] 2
```

```
ncol(matriz1)     # Número de columnas
```

```
[1] 3
```

```
# Asignación de nombres a filas y columnas
rownames(matriz1) <- c("Fila1", "Fila2")
colnames(matriz1) <- c("Col1", "Col2", "Col3")
print(matriz1)
```

```
      Col1 Col2 Col3
Fila1    1    3    5
Fila2    2    4    6
```

El acceso a los elementos de una matriz se realiza mediante la notación `[fila, columna]`. Esta sintaxis permite extraer elementos individuales, filas completas, columnas completas o subconjuntos específicos de la matriz. Además, es posible aplicar condiciones lógicas para filtrar elementos, lo que resulta fundamental en la exploración y transformación de datos (Field, 2013; Grolemund & Wickham, 2017).

```
# Acceso a un elemento específico
elemento <- matriz1[2, 1]     # Elemento en fila 2, columna 1
print(elemento)
```

```
[1] 2
```

```
# Acceso a una fila completa
fila_completa <- matriz1[1, ] # Primera fila completa
print(fila_completa)
```

```
Col1 Col2 Col3
   1    3    5
```

```r
# Acceso a una columna completa
col_completa <- matriz1[, 2]  # Segunda columna completa
print(col_completa)
```

```
Fila1 Fila2
    3     4
```

```r
# Filtrado condicional
elementos_mayores_3 <- matriz1[matriz1 > 3]
print(elementos_mayores_3)  # Devuelve: 4 5 6
```

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

4.2.3 Combinación de Matrices

La combinación de matrices es una operación fundamental en la gestión y consolidación de datos, especialmente cuando se trabaja con información proveniente de diferentes fuentes o experimentos. R proporciona funciones específicas para unir matrices de manera eficiente y controlada: `cbind()` para combinar por columnas y `rbind()` para combinar por filas. Es indispensable que las dimensiones sean compatibles; de lo contrario, R generará un error. Además, la homogeneidad de tipo de datos se mantiene, y si se combinan tipos distintos, R aplicará coerción automática al tipo más general (R Core Team, 2023; Field, 2013).

```r
# Crear dos matrices compatibles para combinar
matrizA <- matrix(1:6, nrow = 3)
matrizB <- matrix(7:12, nrow = 3)

# Combinación por columnas
matriz_columnas <- cbind(matrizA, matrizB)
print(matriz_columnas)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```r
# Combinación por filas
matriz_filas <- rbind(matrizA, matrizB)
print(matriz_filas)
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
[4,]    7   10
[5,]    8   11
[6,]    9   12
```

La correcta combinación de matrices permite consolidar conjuntos de datos, preparar información para análisis multivariados y estructurar resultados experimentales de manera eficiente. Este proceso es especialmente relevante en contextos de análisis de grandes volúmenes de datos y en la integración de resultados de diferentes experimentos o fuentes (Kutner et al., 2005; Hernández, Usuga & Mazo, 2024).

4.2.4 Operaciones y aplicaciones de las matrices

Las matrices en R permiten realizar una amplia variedad de operaciones algebraicas y estadísticas, fundamentales para el análisis de datos y la modelización matemática. Entre las operaciones más relevantes se encuentran la suma y multiplicación elemento a elemento, la multiplicación matricial, la transposición, el cálculo de matrices de correlación y la descomposición en valores singulares. Estas operaciones son esenciales en el desarrollo de modelos estadísticos avanzados, análisis multivariados y procedimientos de álgebra lineal (Montgomery et al., 2012; Venables & Ripley, 2002).

```r
# Operaciones aritméticas elemento a elemento
matriz_C <- matrix(1:4, nrow = 2)
matriz_D <- matrix(5:8, nrow = 2)

suma <- matriz_C + matriz_D
producto <- matriz_C * matriz_D

# Multiplicación matricial
producto_matricial <- matriz_C %*% matriz_D

# Transposición de matrices
matriz_transpuesta <- t(matriz_C)
```

Para profundizar en el uso de matrices y su aplicación en el análisis estadístico con R, se recomienda consultar el capítulo 20 del libro *Modelos de Regresión con R* de Hernández, Usuga y Mazo (2024), donde se aborda el álgebra matricial de manera detallada y aplicada.

4.3 Data frames

El data frame constituye una de las estructuras de datos más relevantes y versátiles en el entorno de R, permitiendo la organización de información en un formato tabular bidimensional, donde las filas representan observaciones individuales y las columnas corresponden a variables específicas. Esta estructura es análoga a una hoja de cálculo en Excel o a una tabla en una base de datos relacional, lo que facilita la transición de datos entre diferentes plataformas y sistemas de análisis (R Core Team, 2023).

Una característica distintiva de los data frames es la posibilidad de que cada columna almacene un tipo de dato diferente, como valores numéricos, cadenas de texto, valores lógicos o factores. Esta flexibilidad resulta fundamental para el manejo de datos heterogéneos, permitiendo la integración y el análisis eficiente de información proveniente

de encuestas, experimentos científicos, registros administrativos y otros contextos donde la diversidad de variables es común. Además, los data frames son ampliamente compatibles con funciones y paquetes especializados en R, como `ggplot2` y `dplyr`, lo que los convierte en la estructura de datos más utilizada en el análisis estadístico y científico con este lenguaje (Wickham & Grolemund, 2017; R Core Team, 2023).

4.3.1 Creación de data frames

La creación de un data frame en R se realiza mediante la función `data.frame()`. Esta función combina varios vectores de igual longitud en una estructura tabular. Es imprescindible que todos los vectores tengan la misma cantidad de elementos, ya que cada fila representa una observación completa. El proceso de creación puede organizarse en los siguientes pasos:

1. Definir los vectores que se desean combinar, asegurando que todos tengan la misma longitud.
2. Utilizar la función `data.frame()` para unir los vectores en una estructura tabular.
3. Asignar el resultado a un objeto para su posterior manipulación y análisis.

A continuación se muestra un ejemplo utilizando los vectores definidos previamente:

```
# 1. Definir los vectores
nombres <- c("Juan", "Ana", "Luis", "María")
edades <- c(17, 20, 18, 25)
mayores_de_edad <- edades >= 18

# 2. Crear el data frame
datos <- data.frame(nombres, edades, mayores_de_edad)

# 3. Visualizar el data frame
datos
```

```
  nombres edades mayores_de_edad
1    Juan     17           FALSE
2     Ana     20            TRUE
3    Luis     18            TRUE
4   María     25            TRUE
```

En este ejemplo, el objeto `datos` corresponde a un data frame compuesto por tres columnas: `nombres` (caracteres), `edades` (numéricos) y `mayores_de_edad` (lógicos). Esta estructura permite almacenar y analizar información de manera eficiente, facilitando la manipulación y el acceso a los datos según las necesidades del análisis (R Core Team, 2023).

4.3.2 Ventajas de un data frame

Los data frames presentan múltiples ventajas que los hacen indispensables en el análisis de datos con R. Entre las principales ventajas se destacan:

1. **Estructura clara:** Cada fila representa una observación y cada columna una variable, lo que facilita la interpretación y el manejo de la información.
2. **Compatibilidad:** Los data frames funcionan con funciones estadísticas, herramientas de visualización y paquetes populares como `ggplot2` y `dplyr`, ampliando significativamente las posibilidades de análisis y presentación de resultados.
3. **Flexibilidad:** Es posible almacenar diferentes tipos de datos en las columnas, como números, texto y factores, lo que resulta esencial para el análisis de datos heterogéneos.
4. **Facilidad de manipulación:** Existen numerosas funciones y herramientas para filtrar, seleccionar, transformar y resumir la información contenida en un data frame, lo que contribuye a la eficiencia y robustez del proceso analítico (Wickham & Grolemund, 2017; Field, 2013).

4.3.3 Manipulación de data frames

R ofrece diversas formas de manipular data frames, tanto mediante funciones básicas como a través de herramientas avanzadas de paquetes especializados. Entre las operaciones más comunes se encuentran:

4.3.3.1 Acceso a columnas

Para acceder a una columna específica de un data frame, se utiliza el operador `$` seguido del nombre de la columna. Esta operación devuelve el vector correspondiente a la variable seleccionada.

```
# Acceso a la columna 'nombres'
datos$nombres
```

```
[1] "Juan"  "Ana"   "Luis"  "María"
```

4.3.3.2 Filtrado de filas

Es posible seleccionar filas que cumplan ciertas condiciones lógicas, lo que resulta fundamental para el análisis exploratorio y la segmentación de datos. Por ejemplo, para obtener únicamente las observaciones donde la edad es mayor a 20 años:

```
# Filtrar filas donde la edad sea mayor a 20
datos_filtrados <- datos[datos$edades > 20, ]
datos_filtrados
```

```
  nombres edades mayores_de_edad
4   María     25            TRUE
```

4.3.3.3 Agregar nuevas columnas

Se pueden agregar nuevas variables a un data frame asignando un vector a un nuevo nombre de columna. Por ejemplo, para añadir la altura de cada persona:

```r
# Agregar una columna llamada 'altura' al data frame
datos$altura <- c(1.75, 1.60, 1.80, 1.65)
```

Después de esta operación, el data frame `datos` tendrá una columna adicional llamada `altura`, donde cada valor corresponde a la altura de la persona en la misma fila.

4.3.3.4 Seleccionar varias columnas

Para trabajar con un subconjunto de variables, la función `subset()` permite crear un nuevo data frame que contiene únicamente las columnas seleccionadas:

```r
# Crear un nuevo data frame solo con las columnas 'nombres' y 'edades'
subgrupo <- subset(datos, select = c(nombres, edades))
```

4.3.3.5 Resumir información

La función `summary()` genera un resumen estadístico de cada columna del data frame, proporcionando información relevante como el valor mínimo, máximo, media, mediana y, en el caso de variables categóricas, la frecuencia de cada categoría. Esta función resulta esencial para la exploración inicial y la comprensión de la estructura de los datos antes de realizar análisis más detallados (Field, 2013; R Core Team, 2023).

```r
# Obtener un resumen estadístico de todas las columnas del data frame
summary(datos)
```

```
   nombres              edades      mayores_de_edad     altura
 Length:4           Min.   :17.00   Mode :logical   Min.   :1.600
 Class :character   1st Qu.:17.75   FALSE:1         1st Qu.:1.637
 Mode  :character   Median :19.00   TRUE :3         Median :1.700
                    Mean   :20.00                   Mean   :1.700
                    3rd Qu.:21.25                   3rd Qu.:1.762
                    Max.   :25.00                   Max.   :1.800
```

4.4 Listas

Las listas en R son estructuras de datos sumamente flexibles y potentes, ya que permiten almacenar elementos de diferentes tipos y longitudes dentro de un mismo objeto. A diferencia de los data frames, donde todas las columnas deben tener la misma longitud y cada columna representa una variable, en una lista cada elemento puede ser un vector, un data frame, una matriz, una función, o incluso otra lista. Esta característica hace que las listas sean ideales para guardar resultados complejos, como salidas de modelos estadísticos, colecciones de datos heterogéneos o cualquier conjunto de información que no encaje en una estructura tabular tradicional (R Core Team, 2023).

4.4.1 Creación de listas

La creación de una lista en R se realiza mediante la función `list()`. Cada elemento puede tener un nombre y puede ser de cualquier tipo de objeto. El proceso de creación puede organizarse en los siguientes pasos:

1. Definir los elementos que se desean incluir en la lista, pudiendo ser de cualquier tipo y longitud.
2. Asignar nombres a los elementos para facilitar su identificación y acceso posterior.
3. Utilizar la función `list()` para agrupar los elementos en un solo objeto.

Por ejemplo:

```
# 1. Definir los elementos
nombres <- c("Juan", "Ana")      # Vector de texto
edades <- c(18, 20)              # Vector numérico
datos_completos <- datos         # Data frame

# 2. Crear la lista con nombres para cada elemento
mi_lista <- list(
  nombres = nombres,
  edades = edades,
  datos_completos = datos_completos
)
```

En este ejemplo, la lista `mi_lista` contiene tres elementos:

1. El elemento `nombres` es un vector de texto.
2. El elemento `edades` es un vector numérico.
3. El elemento `datos_completos` corresponde a un data frame.

Esta estructura permite almacenar y organizar información heterogénea de manera eficiente (R Core Team, 2023).

4.4.2 Acceso a elementos de una lista

El acceso a los elementos de una lista puede realizarse de varias formas, según la necesidad del análisis:

1. **Por nombre**: Utilizando el operador `$` o corchetes dobles `[[ ]]`

```
# Acceder al elemento 'nombres' usando $
mi_lista$nombres
```

```
[1] "Juan" "Ana"
```

```
# Acceder al elemento 'nombres' usando corchetes dobles
mi_lista[["nombres"]]
```

```
[1] "Juan" "Ana"
```

Ambas formas devuelven el vector de nombres almacenado en la lista.

2. **Por índice**: Utilizando corchetes dobles `[[ ]]`:

```
# Acceder al primer elemento de la lista
mi_lista[[1]] #En este caso, el vector de nombres
```

```
[1] "Juan" "Ana"
```

Esto es útil cuando se desconoce el nombre del elemento, pero se conoce su posición dentro de la lista.

3. **Diferencia entre corchetes simples y dobles:** Si se utilizan corchetes simples `[ ]` para acceder a un elemento de la lista, el resultado será una sublista (es decir, una lista que contiene el elemento seleccionado), no el elemento en sí. Para obtener directamente el contenido, siempre utilice corchetes dobles `[[ ]]` o el operador `$` si el elemento tiene nombre.

```
# Devuelve una sublista
mi_lista[1]
```

```
$nombres
[1] "Juan" "Ana"
```

```
# Devuelve el elemento directamente
mi_lista[[1]]
```

```
[1] "Juan" "Ana"
```

Esta distinción resulta fundamental para evitar errores en la manipulación de listas y para acceder correctamente a los datos almacenados (Wickham & Grolemund, 2017).

4.4.3 Aplicaciones prácticas

Las listas en R resultan especialmente valiosas cuando se requiere almacenar y organizar resultados complejos derivados de análisis estadísticos. Por ejemplo, al ajustar un modelo de regresión, la función `lm()` genera una lista que contiene los coeficientes estimados, los residuos, los valores ajustados y otros diagnósticos relevantes. Esta estructura permite acceder fácilmente a cada componente del análisis para su interpretación o procesamiento posterior (R Core Team, 2023).

Además, las listas son ideales para agrupar diferentes tipos de datos relacionados en un solo objeto, como vectores, data frames, matrices o incluso otras listas. Esta capacidad de contener elementos heterogéneos facilita la gestión de información en proyectos de análisis de datos, donde es común trabajar con resultados de distintas etapas o fuentes (Wickham & Grolemund, 2017).

4.5 Comparación entre Data Frames y Listas

En R, los data frames y las listas constituyen estructuras de datos esenciales, pero difieren significativamente en su organización interna y en los contextos para los que resultan más apropiadas. La siguiente tabla resume las diferencias clave entre ambas estructuras (R Core Team, 2023; Wickham & Grolemund, 2017):

Característica	Data Frame	Lista
Estructura	Tabular: filas y columnas	Colección de objetos heterogéneos
Tipos de datos	Cada columna puede tener un tipo distinto, pero todos los elementos de una columna deben ser del mismo tipo	Cada elemento puede ser de cualquier tipo y longitud
Uso principal	Análisis estadístico y visualización de datos estructurados	Almacenamiento y gestión de resultados complejos o heterogéneos
Acceso a elementos	Por columnas (usando `$` o corchetes) y por índices de filas y columnas	Por nombre o por índice, utilizando corchetes dobles `[[ ]]` o el operador `$`

La elección entre data frames y listas depende del tipo de información y del objetivo del análisis. Para datos tabulares, como encuestas o resultados experimentales, se recomienda emplear data frames. Cuando se requiere almacenar y manipular resultados complejos o combinaciones de diferentes tipos de datos, las listas resultan más adecuadas.

5 Importación de datos

La importación de datos constituye una etapa esencial en el proceso de análisis estadístico, ya que posibilita el acceso y la manipulación de información proveniente de diversas fuentes, tales como archivos en formato CSV, hojas de cálculo de Excel o datos disponibles en páginas web. El entorno R ofrece un conjunto robusto de funciones y paquetes especializados que permiten realizar la importación de datos de manera eficiente, reproducible y escalable, facilitando así el manejo de grandes volúmenes de información y promoviendo la integridad y trazabilidad de los análisis (R Core Team, 2023; Grolemund & Wickham, 2017).

5.1 Configuración previa: el directorio de trabajo

Previo a la importación de datos, resulta imprescindible verificar y establecer correctamente el directorio de trabajo. El directorio de trabajo, conocido en inglés como *working directory*, corresponde a la carpeta desde la cual R accede a los archivos de entrada y en la que almacena los resultados generados. Una adecuada configuración de este directorio contribuye significativamente a la organización y eficiencia del flujo de trabajo (Bryan, 2018; Gentleman & Temple Lang, 2007).

En el caso de utilizar proyectos de RStudio (.Rproj), el directorio de trabajo se define automáticamente al abrir el proyecto. Esta acción simplifica la gestión de archivos y minimiza la probabilidad de errores relacionados con rutas de acceso. No obstante, cuando se trabaja con scripts independientes, es necesario establecer manualmente el directorio de trabajo mediante la función `setwd()`. Por ejemplo:

```
# Establecer directorio de trabajo
setwd("ruta/del/directorio")
```

La correcta configuración del directorio de trabajo previene errores frecuentes, como el mensaje "archivo no encontrado". Además, garantiza que el código sea portable y replicable en distintos sistemas o ubicaciones (Bryan, 2018).

5.1.1 Automatización del directorio de trabajo en scripts independientes

Para scripts que no están asociados a un proyecto de RStudio, es posible automatizar la configuración del directorio de trabajo utilizando el paquete `rstudioapi`. Este enfoque permite establecer como directorio de trabajo la carpeta en la que se encuentra guardado el script. Esto facilita la portabilidad y la colaboración entre diferentes usuarios y equipos (Bryan, 2018).

El siguiente fragmento de código ilustra este procedimiento:

```r
# Instalación y carga del paquete rstudioapi
if (!require("rstudioapi")) install.packages("rstudioapi")

# Linea empleada para establecer  el directorio de trabajo
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
```

Este código verifica si el paquete `rstudioapi` está instalado y, en caso contrario, lo instala automáticamente. Posteriormente, obtiene la ruta del script en ejecución y la utiliza para definir el directorio de trabajo. Esto permite el acceso a los archivos de la carpeta sin necesidad de especificar rutas absolutas.

5.1.2 Verificación y buenas prácticas

Antes de proceder con la importación de datos o el almacenamiento de resultados, se recomienda verificar el directorio de trabajo actual mediante la función `getwd()`:

```r
# Verificación del directorio de trabajo actual
getwd()
```

Adicionalmente, es aconsejable guardar el script antes de ejecutar la configuración automática del directorio. R requiere conocer la ubicación del archivo para establecer correctamente el entorno de trabajo.

Siempre que sea posible, se sugiere trabajar dentro de un proyecto de RStudio. Esta práctica automatiza la gestión del directorio de trabajo y favorece la organización de los archivos y recursos asociados al análisis. Esto promueve la reproducibilidad y la eficiencia en el desarrollo de proyectos de análisis de datos (Bryan, 2018; Gentleman & Temple Lang, 2007).

5.2 Importación de archivos CSV y Excel en R

La importación de datos tabulares es una tarea fundamental en el análisis estadístico. R facilita este proceso mediante funciones y paquetes. Estos permiten trabajar con archivos en formatos ampliamente utilizados, como CSV y Excel. Comprender cómo importar estos archivos correctamente es esencial para garantizar la integridad y reproducibilidad del análisis (Wickham, 2016).

5.2.1 Importación de archivos CSV

El formato CSV (Comma-Separated Values) se ha consolidado como uno de los estándares más utilizados para el almacenamiento y el intercambio de datos tabulares, debido a su simplicidad, legibilidad y compatibilidad con múltiples plataformas y aplicaciones. En R, la función `read.csv()` permite importar archivos CSV de manera eficiente, posibilitando la lectura de grandes volúmenes de datos sin requerir transformaciones previas (Grolemund & Wickham, 2017).

A continuación, se presenta un ejemplo básico de importación de un archivo CSV:

```r
# Importar un archivo CSV
datos <- read.csv("ruta/del/archivo/datos.csv",
                  header = TRUE,
                  sep = ",")
```

Los parámetros principales de esta función se describen a continuación:

1. **header**: Este argumento lógico indica si la primera fila del archivo CSV contiene los nombres de las columnas. Si se establece en `TRUE`, la primera fila se interpretará como los nombres de las variables; si se establece en `FALSE`, R asignará nombres genéricos a las columnas (R Core Team, 2023).
2. **sep**: Este argumento especifica el carácter que se utiliza para separar los valores en cada fila del archivo CSV. El valor predeterminado es la coma (`,`), pero puede ajustarse a otros caracteres, como el punto y coma (`;`) o la tabulación (`\t`), en función del formato del archivo (Grolemund & Wickham, 2017).

5.2.2 Importación de archivos Excel

El uso de hojas de cálculo en formato Excel (.xlsx) es frecuente en contextos profesionales y académicos, debido a la flexibilidad y las capacidades de organización que ofrece este software. Para la importación de archivos Excel en R, el paquete `readxl` proporciona funciones optimizadas que permiten acceder a los datos directamente desde las hojas de cálculo, sin necesidad de convertir los archivos a otros formatos intermedios (Wickham, 2016).

El procedimiento recomendado para la importación de archivos Excel incluye la instalación y carga del paquete `readxl`, seguido del uso de la función `read_excel()`, como se muestra a continuación:

```r
# Instalar y cargar el paquete readxl
if (!require("readxl")) install.packages("readxl")
```

```r
# Importar un archivo Excel
datos_excel <- read_excel("ruta/del/archivo/datos.xlsx",
                          sheet = "Hoja1",
                          col_names = TRUE/FALSE)
```

Los argumentos más relevantes de la función `read_excel()` se detallan a continuación:

1. **sheet**: Este argumento permite seleccionar la hoja específica que se desea importar desde el archivo Excel. Se puede especificar el nombre de la hoja como una cadena de caracteres (por ejemplo, `"Hoja1"`) o el número de la hoja (por ejemplo, `1` para la primera hoja) (Wickham, 2016).
2. **col_names**: Este argumento lógico determina si la primera fila de la hoja de cálculo debe ser utilizada como los nombres de las columnas. Si se establece en `TRUE`, la primera fila se interpretará como los nombres de las variables; si se establece en `FALSE`, R asignará nombres genéricos a las columnas (Grolemund & Wickham, 2017).

5.3 Verificación de la importación de datos

La verificación de la importación de datos representa una fase crítica en el proceso de análisis estadístico, ya que permite identificar y corregir posibles inconsistencias, errores de formato o problemas de codificación que puedan afectar la calidad y la validez de los resultados. Una revisión exhaustiva de los datos importados contribuye a la reproducibilidad y confiabilidad de los análisis, además de optimizar el flujo de trabajo y prevenir dificultades en etapas posteriores (Grolemund & Wickham, 2017; R Core Team, 2023).

5.3.1 Inspección preliminar de los datos

La inspección preliminar consiste en una revisión rápida del contenido y la estructura del conjunto de datos recién importado. R proporciona funciones específicas para este propósito:

1. **head()**: Permite visualizar las primeras filas del data frame, facilitando la comprobación de la correcta lectura de los encabezados, la alineación de las columnas y la presencia de datos esperados.
2. **tail()**: Muestra las últimas filas del conjunto de datos, útil para verificar la integridad de los registros al final del archivo.
3. **dim()**: Informa sobre el número de filas y columnas, lo que ayuda a confirmar que la cantidad de observaciones y variables coincide con lo esperado (Venables & Ripley, 2002).

5.3.2 Evaluación de la estructura y los tipos de variables

La correcta interpretación de los tipos de variables es fundamental para evitar errores en el análisis estadístico. R ofrece herramientas para examinar la estructura interna del objeto de datos:

1. **str()**: Proporciona información detallada sobre el tipo de cada variable (numérica, carácter, factor, etc.), la cantidad de observaciones y la organización de las columnas. Esta función resulta esencial para detectar conversiones automáticas no deseadas, como la transformación de variables numéricas en factores o viceversa (Grolemund & Wickham, 2017).
2. **names()**: Permite consultar los nombres de las columnas, lo que facilita la identificación de posibles errores en la lectura de los encabezados o la presencia de nombres duplicados.

5.3.3 Resumen estadístico y detección de inconsistencias

El análisis exploratorio inicial incluye la obtención de resúmenes estadísticos básicos, que permiten identificar valores atípicos, rangos inusuales o la presencia de datos faltantes:

1. **summary()**: Genera un resumen estadístico para cada variable, incluyendo medidas de tendencia central, dispersión y frecuencia de valores nulos. Esta función resulta útil para detectar anomalías y orientar las primeras etapas de limpieza de datos (Wickham, 2016).
2. **table()**: Permite examinar la frecuencia de los valores en variables categóricas, facilitando la identificación de categorías inesperadas o errores de codificación.

5.3.4 Verificación de la codificación de caracteres

En contextos donde los datos contienen caracteres especiales, como tildes o la letra "ñ", es fundamental asegurarse de que la codificación utilizada durante la importación sea la adecuada. El argumento `encoding` en funciones como `read.csv()` permite especificar la codificación, por ejemplo, `"UTF-8"`, para evitar la aparición de símbolos incorrectos o pérdida de información (R Core Team, 2023).

5.3.5 Identificación de valores faltantes y duplicados

La presencia de valores faltantes o registros duplicados puede afectar la validez de los análisis. R ofrece funciones para detectar y cuantificar estos casos:

1. **is.na()**: Permite identificar valores ausentes en el conjunto de datos.
2. **anyDuplicated()**: Informa sobre la existencia de filas duplicadas, lo que resulta relevante para garantizar la unicidad de las observaciones (Venables & Ripley, 2002).

5.3.6 Importancia de la verificación sistemática

La aplicación sistemática de estas herramientas y procedimientos permite detectar de manera temprana problemas que, de no ser corregidos, pueden comprometer la interpretación y la validez de los resultados. La verificación de la importación de datos debe considerarse una buena práctica en la gestión de información y un paso indispensable en cualquier proyecto de análisis estadístico (Bryan, 2018; R Core Team, 2023).

6 Operadores en R

En el lenguaje de programación R, los operadores constituyen herramientas esenciales que permiten ejecutar cálculos matemáticos, realizar comparaciones lógicas, efectuar asignaciones de valores y manipular estructuras de datos. Su dominio es fundamental para desarrollar análisis estadísticos robustos y para implementar flujos de trabajo reproducibles y eficientes en programación estadística (R Core Team, 2023). Los operadores pueden considerarse los instrumentos básicos de un entorno de trabajo analítico, ya que su correcta aplicación posibilita la construcción de soluciones complejas a partir de operaciones elementales.

La clasificación de los operadores en R se realiza en función de la naturaleza de las operaciones que permiten ejecutar. A continuación, se describen las principales categorías de operadores, acompañadas de ejemplos prácticos que ilustran su uso y aplicación en el contexto del análisis de datos.

Tipo de Operador	Ejemplo	Descripción
Aritméticos	`+`, `-`, `*`, `/`	Realizan operaciones matemáticas básicas como suma, resta, multiplicación, entre otras
Lógicos	`>`, `<`, `==`, `!=`	Comparan valores y devuelven un resultado lógico (`TRUE` o `FALSE`)
Asignación	`<-`, `=`, `->`	Asignan valores a objetos
Manipulación de datos	`$`, `[]`, `:`	Acceden o manipulan elementos dentro de estructuras de datos

6.1 Operadores de Asignación

En R, los operadores de asignación cumplen la función de crear objetos y almacenar valores o resultados en ellos, lo que representa uno de los pilares de la programación y la manipulación de datos en este lenguaje. Los dos operadores de asignación más empleados son `<-` y `=`, ambos válidos para asignar valores a objetos. Sin embargo, la convención ampliamente aceptada en la comunidad de R es utilizar el operador `<-`, ya que este evita ambigüedades con el operador lógico de igualdad (`==`) y contribuye a mantener la claridad y coherencia en el código (Ihaka & Gentleman, 1996).

Por ejemplo, la instrucción:

```
x <- 10
```

asigna el valor numérico 10 al objeto denominado `x`. De manera similar, la instrucción:

```
y = 20
```

asigna el valor 20 al objeto `y`, aunque esta forma es menos recomendada en contextos profesionales y académicos. Una vez creados, estos objetos pueden ser utilizados en operaciones posteriores, como se muestra a continuación:

```
# Asignación de valores a objetos
x <- 10
y = 20

# Uso de objetos
x + y    # Resultado: 30
```

La salida generada por R será:

```
[1] 30
```

Es relevante señalar que, aunque el operador `=` puede emplearse para asignar valores, su uso puede inducir a confusiones, especialmente en contextos donde se emplean expresiones lógicas o en la definición de argumentos dentro de funciones, ya que `=` también se utiliza para asociar valores a parámetros. Por esta razón, se recomienda preferir el uso de `<-` para la asignación de valores en la mayoría de los casos, siguiendo las mejores prácticas de programación en R (Ihaka & Gentleman, 1996).

6.2 Operadores aritméticos

Los operadores aritméticos son elementos fundamentales en R, ya que posibilitan la ejecución de operaciones matemáticas tanto básicas como avanzadas. Estos operadores son esenciales para la manipulación de datos numéricos y la realización de cálculos en el ámbito del análisis estadístico. Actúan sobre valores numéricos y producen resultados numéricos, permitiendo así la transformación y el análisis cuantitativo de la información (R Core Team, 2023).

La siguiente tabla resume los principales operadores aritméticos disponibles en R, junto con su función, un ejemplo de uso y el resultado esperado:

Operador	**Acción**	**Ejemplo**	**Resultado**
`+`	Suma	`5 + 3`	`8`
`-`	Resta	`10 - 4`	`6`
`*`	Multiplicación	`6 * 2`	`12`
`/`	División	`15 / 3`	`5`
`^`	Potencia	`2 ^ 3`	`8`
`%/%`	División entera	`17 %/% 5`	`3`
`%%`	Módulo o residuo	`17 %% 5`	`2`

6.2.1 Ejemplo práctico

El siguiente ejemplo ilustra el uso de operadores aritméticos en R para efectuar cálculos matemáticos básicos entre dos variables numéricas. Cada operación se acompaña de una breve explicación sobre su funcionamiento y utilidad.

```
# Definición de variables numéricas
a <- 15
b <- 4

# Suma de a y b
suma <- a + b           # Resultado: 19

# Resta de b a a
resta <- a - b          # Resultado: 11

# Multiplicación de a por b
multiplicacion <- a * b # Resultado: 60

# División de a entre b
division <- a / b       # Resultado: 3.75

# Potencia: a elevado a la b
potencia <- a ^ b       # Resultado: 50625

# División entera de a entre b
division_entera <- a %/% b  # Resultado: 3

# Módulo o residuo de la división de a entre b
residuo <- a %% b           # Resultado: 3
```

En este bloque de código, se definen dos variables numéricas, `a` y `b`, y se aplican distintos operadores aritméticos. El operador `+` realiza la suma, `-` la resta, `*` la multiplicación y `/` la división estándar. El operador `^` calcula la potencia, es decir, eleva `a` a la `b`. Por su parte, `%/%` efectúa la división entera, devolviendo únicamente la parte entera del cociente, mientras que `%%` retorna el residuo de la división.

Este ejemplo evidencia cómo los operadores aritméticos en R permiten efectuar operaciones matemáticas básicas de manera directa y eficiente, facilitando la obtención de resultados numéricos en el análisis de datos (R Core Team, 2023).

6.3 Operadores lógicos

Los operadores lógicos desempeñan un papel crucial en la evaluación de condiciones y la toma de decisiones dentro del código en R. Estos operadores permiten comparar valores y establecer reglas condicionales, lo cual es esencial para tareas como el filtrado de datos, la selección de subconjuntos y la implementación de estructuras de control. Los operadores

lógicos trabajan con valores booleanos (`TRUE` o `FALSE`), y su correcta utilización facilita la construcción de análisis estadísticos robustos y flexibles (R Core Team, 2023).

A continuación, se presenta una tabla que resume los principales operadores lógicos en R, junto con su función, un ejemplo de uso y el resultado esperado:

Operador	Acción	Ejemplo	Resultado
`>`	Mayor que	`5 > 3`	`TRUE`
`<`	Menor que	`5 < 3`	`FALSE`
`>=`	Mayor o igual que	`5 >= 5`	`TRUE`
`<=`	Menor o igual que	`5 <= 4`	`FALSE`
`==`	Igualdad	`5 == 5`	`TRUE`
`!=`	Desigualdad	`5 != 3`	`TRUE`
`&`	Y lógico (AND)	`(5 > 3) & (4 > 2)`	`TRUE`
`|`	O lógico (OR)	`(4 < 2) | (5 > 3)`	`TRUE`
`!`	Negación lógica	`!(5 > 3)`	`FALSE`

6.3.1 Ejemplo práctico

Para ilustrar el uso de los operadores lógicos en R, se presenta un ejemplo práctico que emplea fragmentos de código comentados y explicados paso a paso. Estas operaciones son fundamentales para realizar comparaciones y evaluaciones condicionales en el contexto del análisis de datos.

Primero, se muestran comparaciones simples utilizando operadores lógicos:

```
# Comparaciones simples
edad <- 25
es_mayor <- edad > 18      # TRUE, porque 25 es mayor que 18
es_menor <- edad < 30      # TRUE, porque 25 es menor que 30
es_igual <- edad == 25     # TRUE, porque 25 es igual a 25
```

En este bloque de código, se asigna el valor 25 a la variable `edad` y se realizan diversas comparaciones lógicas. El operador `>` evalúa si `edad` es mayor que 18, el operador `<` evalúa si `edad` es menor que 30 y el operador `==` evalúa si `edad` es igual a 25. Cada una de estas comparaciones devuelve un valor lógico (`TRUE` o `FALSE`) que se almacena en las variables correspondientes.

A continuación, se ejemplifican operaciones lógicas compuestas mediante el cálculo del índice de masa corporal (IMC) y la evaluación de si una persona se encuentra en un rango de peso normal o en sobrepeso:

```
# Operaciones lógicas compuestas
peso_Kg <- 70
altura <- 1.75
# Cálculo del índice de masa corporal
imc <- peso_Kg / (altura^2)
```

```r
sobrepeso <- imc >= 25 & imc < 30
sobrepeso  # FALSE, el IMC está fuera del rango de sobrepeso
```

```
[1] FALSE
```

```r
peso_normal <- imc >= 18.5 & imc < 25
peso_normal # TRUE, el IMC está en el rango de peso normal
```

```
[1] TRUE
```

En este caso, se calculó el IMC dividiendo el peso en kilogramos por el cuadrado de la altura en metros. Luego, se utilizan operadores lógicos compuestos (`&`) para evaluar si el IMC se encuentra dentro de los rangos de sobrepeso (mayor o igual a 25 y menor que 30) y peso normal (mayor o igual a 18.5 y menor que 25). El resultado de estas evaluaciones lógicas se almacena en las variables `sobrepeso` y `peso_normal`, respectivamente.

En resumen, este ejemplo ilustra cómo los operadores lógicos permiten evaluar condiciones tanto simples como compuestas, facilitando la clasificación de datos y la toma de decisiones dentro del análisis estadístico en R (R Core Team, 2023). El cálculo del IMC y la posterior evaluación de si una persona se encuentra en un rango de peso normal o en sobrepeso son ejemplos claros de la aplicación práctica de estos operadores.

6.4 Operadores de Manipulación de Datos

En R, los operadores de manipulación de datos desempeñan una función esencial en el acceso, selección y modificación de elementos dentro de diversas estructuras de datos como vectores, matrices, listas y data frames. El dominio de estos operadores resulta indispensable para trabajar con datos organizados y ejecutar análisis estadísticos de manera eficiente, ya que permiten extraer, transformar y analizar información específica de grandes conjuntos de datos (R Core Team, 2023).

La siguiente tabla resume los principales operadores de manipulación de datos en R, su función, un ejemplo de uso y el resultado esperado:

Operador	Acción	Ejemplo	Resultado
`[]`	Acceso a elementos por posición	`vector[1]`	Primer elemento del vector
`[ , ]`	Acceso a filas y columnas en un data frame	`data[1, 2]`	Elemento en la fila 1, columna 2
`$`	Acceso a una columna específica en un data frame	`data$columna`	Columna seleccionada
`:`	Creación de secuencias	`1:10`	Secuencia del 1 al 10

Estos operadores constituyen herramientas fundamentales para la manipulación de datos en R, permitiendo a los analistas y científicos de datos acceder con precisión a los elementos que necesitan procesar. Su correcta aplicación facilita la implementación de análisis estadísticos complejos y la generación de visualizaciones informativas (Wickham & Grolemund, 2017).

6.4.1 Ejemplo Práctico

Para ilustrar el uso de los operadores de manipulación de datos en R, se expone un ejemplo práctico que emplea fragmentos de código comentados y explicados paso a paso. Estas operaciones constituyen la base para el manejo eficiente de información en el entorno estadístico.

Primero, se muestra cómo crear un vector numérico y acceder a sus elementos individuales:

```
# Crear un vector numérico
vector <- c(10, 20, 30, 40, 50)

# Acceder al primer elemento del vector
vector[1]  # Resultado: 10
```

```
[1] 10
```

El operador de corchetes `[ ]` permite seleccionar elementos específicos de un vector. En este caso, `vector[1]` retorna el primer valor almacenado, que es 10.

A continuación, se ejemplifica la creación de un data frame, estructura fundamental para el análisis de datos en R, y la selección de columnas y elementos particulares:

```
# Crear un data frame con variables nombre, edad y peso
data <- data.frame(
  nombre = c("Juan", "Ana", "Luis"),
  edad = c(25, 30, 22),
  peso = c(70, 65, 80)
)

# Acceder a la columna 'edad'
data$edad  # Resultado: 25, 30, 22
```

```
[1] 25 30 22
```

```
# Acceder a un elemento específico: fila 2, columna 3 (peso de Ana)
data[2, 3]  # Resultado: 65
```

```
[1] 65
```

El uso del signo de dólar `$` permite extraer una columna completa por su nombre, mientras que la notación `[fila, columna]` posibilita acceder a un valor específico dentro del data frame.

Por último, se muestra cómo generar una secuencia de números utilizando el operador de dos puntos `:`, que es útil para crear rangos de valores:

```
# Crear una secuencia de números del 1 al 10
secuencia <- 1:10
secuencia  # Resultado: 1, 2, 3, ..., 10
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

En resumen, estos ejemplos demuestran que los operadores de manipulación de datos en R permiten seleccionar elementos individuales, columnas completas o secuencias de valores dentro de las estructuras de datos más utilizadas. Estas operaciones son esenciales para filtrar, transformar y analizar información de manera precisa y eficiente (R Core Team, 2023).

6.4.2 Aplicaciones Avanzadas

Los operadores de manipulación de datos en R pueden combinarse para realizar selecciones más complejas y adaptadas a necesidades específicas de análisis. A continuación, se presentan algunos ejemplos que ilustran aplicaciones avanzadas:

```
# Seleccionar múltiples elementos de un vector
# Se seleccionan los elementos en las posiciones 1, 3 y 5
vector[c(1, 3, 5)]  # Resultado: 10, 30, 50
```

```
[1] 10 30 50
```

```
# Seleccionar un subconjunto de filas y columnas en un data frame
# Se seleccionan el nombre y el peso de las personas mayores de 25 años
data[data$edad > 25, c("nombre", "peso")]
```

```
  nombre peso
2    Ana   65
```

```
# Utilizar secuencias para seleccionar rangos de elementos
# Se seleccionan los elementos desde la posición 2 hasta la 4
vector[2:4]  # Resultado: 20, 30, 40
```

```
[1] 20 30 40
```

En el primer caso, se utiliza un vector de índices para seleccionar posiciones específicas dentro del vector original. En el segundo ejemplo, se emplea una condición lógica para filtrar filas de un data frame y, simultáneamente, se seleccionan columnas específicas por nombre. Finalmente, el uso de secuencias permite extraer rangos continuos de elementos de un vector.

Estas aplicaciones avanzadas evidencian la flexibilidad y potencia de los operadores de manipulación de datos en R, facilitando la realización de selecciones precisas y complejas mediante una sintaxis clara y concisa (Wickham, 2016).

7 Funciones en R

Las funciones constituyen uno de los componentes esenciales y estructurales de la programación en R, desempeñando un papel central en la automatización de tareas, la reducción de la redundancia y la mejora de la legibilidad y mantenibilidad del código. Una función puede entenderse como un bloque de instrucciones encapsuladas que, al ser invocadas, ejecutan una tarea específica sobre uno o varios valores de entrada, denominados argumentos, y devuelven un resultado. El dominio en la creación y utilización de funciones es indispensable para aprovechar plenamente las capacidades de R en el análisis estadístico, la manipulación de datos y el desarrollo de soluciones eficientes (R Core Team, 2023; Wickham & Grolemund, 2017).

El presente capítulo explora en profundidad el concepto de función en R, su estructura fundamental, los tipos existentes y el proceso para definir funciones personalizadas que respondan a necesidades analíticas particulares.

7.1 Definición y características de las funciones en R

En R, una función se define formalmente como un objeto que recibe uno o más argumentos, ejecuta una secuencia de operaciones sobre ellos y retorna un valor como resultado. El uso de funciones permite modularizar el código, facilitando su reutilización, mantenimiento y escalabilidad, aspectos cruciales en el desarrollo de proyectos de análisis de datos de cualquier envergadura (Chambers, 2008).

Toda función en R está compuesta por los siguientes elementos fundamentales:

1. **Nombre:** Identificador único que permite invocar la función dentro del entorno de trabajo.
2. **Argumentos:** Valores de entrada que la función requiere para ejecutar sus operaciones internas. Los argumentos pueden tener valores por defecto, lo que otorga flexibilidad y adaptabilidad a la función.
3. **Cuerpo:** Conjunto de instrucciones que definen el comportamiento de la función y especifican las operaciones a realizar sobre los argumentos recibidos.
4. **Valor de retorno:** Resultado que la función entrega tras la ejecución de su cuerpo. En R, este valor se especifica mediante la instrucción `return()`, aunque si no se utiliza explícitamente, la función devolverá el último valor evaluado.

La correcta estructuración de estos elementos permite desarrollar funciones robustas, reutilizables y fácilmente integrables en flujos de trabajo complejos.

7.1.1 Tipos de funciones

Las funciones en R pueden clasificarse en dos grandes categorías, según su origen y propósito:

1. **Funciones predefinidas (built-in):** Son aquellas incluidas en el núcleo del lenguaje R o en paquetes adicionales ampliamente utilizados. Estas funciones abarcan una extensa variedad de tareas, que van desde operaciones matemáticas y estadísticas básicas, hasta la manipulación avanzada de datos y la generación de gráficos. Su uso es fundamental para la resolución eficiente de problemas comunes y para la implementación de análisis estándar (R Core Team, 2023).
2. **Funciones personalizadas (user-defined):** Son funciones definidas por el usuario para abordar necesidades específicas que no pueden resolverse mediante las funciones predefinidas. La creación de funciones personalizadas resulta especialmente útil para automatizar procesos repetitivos, encapsular procedimientos complejos y adaptar el análisis a contextos particulares. El desarrollo de funciones propias fomenta la modularidad y la reutilización del código, contribuyendo a la eficiencia y escalabilidad de los proyectos (Wickham & Grolemund, 2017; Chambers, 2008).

7.1.2 Funciones predefinidas en R

Las funciones predefinidas en R representan un conjunto de herramientas esenciales que facilitan la ejecución eficiente y directa de operaciones comunes. Integradas tanto en el núcleo del lenguaje como en diversos paquetes complementarios, estas funciones permiten realizar cálculos estadísticos, manipular datos y obtener resúmenes informativos sin la necesidad de definir procedimientos adicionales. Su uso simplifica la resolución de tareas habituales y contribuye a la claridad y concisión del código (R Core Team, 2023).

A continuación, se presentan algunos ejemplos de funciones predefinidas ampliamente utilizadas en R, junto con fragmentos de código que ilustran su aplicación:

```
# Definición de un vector de datos
datos <- c(1:25)

# Cálculo de la media aritmética
mean(datos)  # Resultado: 13
```

```
[1] 13
```

```
# Suma de los elementos del vector
sum(datos)  # Resultado: 325
```

```
[1] 325
```

```
# Cálculo de la desviación estándar
sd(datos)  # Resultado: 7.359801
```

```
[1] 7.359801
```

```r
# Resumen estadístico del conjunto de datos
summary(datos) # Resultado:
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      1       7      13      13      19      25 
```

En este ejemplo, se define un vector de datos llamado `datos` que contiene los números enteros del 1 al 25. Luego, se utilizan diversas funciones predefinidas para calcular estadísticas descriptivas básicas. La función `mean()` calcula la media aritmética, `sum()` calcula la suma de los elementos, `sd()` calcula la desviación estándar y `summary()` proporciona un resumen estadístico que incluye el mínimo, el primer cuartil, la mediana, la media, el tercer cuartil y el máximo.

Estas funciones están disponibles de forma predeterminada en R y permiten realizar análisis estadísticos básicos de manera inmediata, sin requerir definiciones adicionales por parte del usuario. Su dominio es esencial para el trabajo cotidiano con datos en R (R Core Team, 2023). Además de las funciones mostradas, R ofrece una amplia gama de funciones predefinidas para tareas como la manipulación de cadenas de texto (`grep`, `gsub`), la gestión de fechas y horas (`Sys.Date`, `strptime`), la generación de números aleatorios (`runif`, `rnorm`) y la creación de gráficos (`plot`, `hist`), entre muchas otras.

7.1.3 Funciones personalizadas en R

Las funciones personalizadas permiten al usuario definir procedimientos específicos para resolver problemas que no están cubiertos por las funciones predefinidas. Este tipo de funciones resulta especialmente útil para automatizar tareas repetitivas o implementar cálculos complejos adaptados a necesidades particulares. La creación de funciones personalizadas contribuye a la organización y reutilización del código, facilitando el desarrollo de análisis más eficientes (R Core Team, 2023).

A continuación, se muestra un ejemplo de cómo definir y utilizar una función personalizada en R para calcular el área de un círculo a partir de su radio:

```r
# Función personalizada para calcular el área de un círculo
area_circulo <- function(radio) {
  area <- pi * radio^2
  return(area)
}

# Uso de la función personalizada
area <- area_circulo(5)   # Resultado: 78.53982
```

En este ejemplo, el usuario define la lógica de la función, especifica el argumento necesario (radio) y utiliza la función para obtener el resultado deseado. La función `area_circulo` encapsula la fórmula matemática para el cálculo del área, lo que permite reutilizarla fácilmente con diferentes valores de radio sin necesidad de repetir el código.

7.1.4 Diferencias entre funciones predefinidas y personalizadas

Las principales diferencias entre funciones predefinidas y personalizadas en R se resumen en la siguiente tabla:

Característica	Funciones predefinidas	Funciones personalizadas
Disponibilidad	Incluidas en R o en paquetes	Creadas por el usuario
Flexibilidad	Limitada a las tareas para las que fueron diseñadas	Totalmente adaptables a las necesidades del usuario
Ejemplos	`mean()`, `sum()`, `sd()`, `summary()`	`area_circulo()`
Reutilización	Reutilizables en cualquier script	Reutilizables si se definen en el entorno o se importan
Mantenimiento	Actualizadas por los desarrolladores de R o del paquete	Mantenidas por el usuario
Documentación	Generalmente bien documentadas en la ayuda de R	Requieren documentación explícita por parte del usuario

Esta distinción permite seleccionar el tipo de función más adecuado según el contexto y los objetivos del análisis (R Core Team, 2023). En general, se recomienda utilizar funciones predefinidas siempre que sea posible, ya que suelen estar optimizadas y bien probadas. Sin embargo, cuando se requiere una funcionalidad específica que no está disponible en las funciones predefinidas, la creación de funciones personalizadas es la solución más adecuada.

7.2 Usos y beneficios de las funciones en R

El uso de funciones en R representa una estrategia fundamental para optimizar el desarrollo y la gestión de proyectos de análisis de datos. Una de las ventajas más destacadas es la reutilización del código. Cuando se define una función, esta puede emplearse en diferentes partes de un mismo proyecto o incluso en proyectos distintos, lo que reduce la duplicación de instrucciones y facilita el mantenimiento del código. Esta reutilización contribuye a minimizar errores, ya que la lógica centralizada en una función puede ser actualizada o corregida en un solo lugar, propagando automáticamente los cambios a todas las instancias donde se utiliza.

Otro beneficio esencial es la modularidad. Las funciones permiten descomponer problemas complejos en componentes más simples y manejables. Esta descomposición facilita la organización y la estructura del código, haciendo que cada función se enfoque en una tarea específica. Como resultado, el proceso de depuración y mejora del código se vuelve más eficiente, ya que es posible identificar y corregir errores en módulos independientes sin afectar el resto del programa.

La legibilidad y la mantenibilidad del código también se ven favorecidas por el uso de funciones. Encapsular operaciones dentro de funciones contribuye a que el código sea

más claro y comprensible, lo que facilita su revisión y la colaboración entre diferentes usuarios o equipos de trabajo. Además, la automatización de tareas repetitivas mediante funciones incrementa la eficiencia y ahorra tiempo en la ejecución de procesos rutinarios, permitiendo que el analista se concentre en aspectos más estratégicos del análisis (R Core Team, 2023; Wickham & Grolemund, 2017).

7.2.1 Ejemplo de automatización con funciones

Para ilustrar estos beneficios, se puede considerar el caso en el que se requiere calcular el área de varios círculos con diferentes radios. En lugar de repetir manualmente el cálculo para cada radio, basta con aplicar una función personalizada a un vector de radios. Por ejemplo:

```
radios <- c(1, 2, 3, 4, 5)
areas <- area_circulo(radios)
areas
```

El resultado será un vector con las áreas correspondientes a cada radio:

```
[1]  3.141593 12.566371 28.274334 50.265482 78.539816
```

Este ejemplo demuestra cómo el uso de funciones personalizadas permite automatizar cálculos y trabajar de manera más eficiente con conjuntos de datos, reafirmando la importancia de las funciones en la programación con R (R Core Team, 2023).

7.3 Creación de funciones en R: Sintaxis y elementos fundamentales

La definición de funciones en R se realiza mediante una sintaxis clara y estructurada, lo que facilita la creación de procedimientos personalizados para resolver tareas específicas. Comprender la estructura básica de una función es fundamental para aprovechar al máximo la modularidad y reutilización del código en R (R Core Team, 2023; Wickham & Grolemund, 2017).

7.3.1 Sintaxis general de una función en R

La sintaxis básica para crear una función en R consiste en asignar un nombre descriptivo a la función y utilizar la palabra reservada `function`, seguida de una lista de argumentos entre paréntesis. El cuerpo de la función, delimitado por llaves, contiene las instrucciones y operaciones que se ejecutarán al llamar a la función. El valor de retorno se especifica mediante la instrucción `return()`, aunque si no se utiliza explícitamente, R devolverá automáticamente el último valor evaluado en el cuerpo de la función. La estructura general es la siguiente:

```r
nombre_funcion <- function(argumento1, argumento2, ...) {
  # Instrucciones y operaciones
  return(resultado)
}
```

7.3.2 Elementos clave de una función

Cada función en R se compone de los siguientes elementos:

1. **Nombre de la función**: que debe ser descriptivo y reflejar claramente la tarea que realiza.
2. **Argumentos**: representan los valores de entrada requeridos por la función. Es posible asignar valores por defecto a estos argumentos para hacer la función más flexible.
3. **Cuerpo de la función**: contiene la lógica y las operaciones principales, y puede incluir validaciones y manejo de errores para asegurar la robustez del código.
4. **Valor de retorno**: es el resultado que la función entrega tras la ejecución de sus operaciones; este valor puede ser un dato simple o una estructura más compleja, dependiendo del propósito de la función.

7.3.3 Ejemplo de función personalizada

A continuación, se muestra un ejemplo de una función personalizada que convierte temperaturas de grados Celsius a Fahrenheit:

```r
# función para convertir temperaturas de Celsius a Fahrenheit
celsius_a_fahrenheit <- function(celsius) {
    # Validación del tipo de dato del argumento de entrada
    if (!is.numeric(celsius)) {
        # Si el argumento no es numérico,
        # detiene la ejecución y muestra un mensaje de error
        stop("El argumento 'celsius' debe ser numérico")
    }
    # Cálculo de la conversión de Celsius a Fahrenheit
    fahrenheit <- (celsius * 9/5) + 32
    # Devuelve el resultado de la conversión
    return(fahrenheit)
}

# Ejemplo de uso de la función
# Se convierte 25 grados Celsius a Fahrenheit
temperatura_celsius <- 25
resultado <- celsius_a_fahrenheit(temperatura_celsius)
```

Esta función demuestra varios conceptos importantes en la programación con R:

1. **Validación de datos:** La función verifica que el argumento de entrada sea del tipo correcto antes de realizar los cálculos, lo que previene errores y mejora la robustez del código.
2. **Claridad en la estructura:** La función sigue una estructura lógica clara: validación, cálculo y retorno del resultado.
3. **Documentación interna:** Los comentarios explican el propósito de cada sección del código, facilitando su comprensión y mantenimiento.
4. **Reutilización:** La función puede ser utilizada con diferentes valores de entrada, incluyendo vectores de temperaturas, gracias a la vectorización inherente de R.

Para ilustrar la versatilidad de la función, se puede utilizar con múltiples valores simultáneamente:

```
# Ejemplo de uso con múltiples temperaturas
temperaturas_celsius <- c(0, 25, 100)
temperaturas_fahrenheit <- celsius_a_fahrenheit(temperaturas_celsius)
temperaturas_fahrenheit # Vector con los resultados:
```

```
[1]  32  77 212
```

La inclusión de comentarios detallados y ejemplos de uso hace que el código sea más accesible para otros usuarios y facilita su mantenimiento a largo plazo, aspectos fundamentales en el desarrollo de software científico y análisis de datos (R Core Team, 2023).

8 Paquetes en R

Los paquetes en R constituyen una de las herramientas más potentes del lenguaje, ya que permiten ampliar sus funcionalidades básicas y abordar tareas especializadas de manera eficiente. Gracias a los paquetes, es posible acceder a funciones, datos y documentación desarrollados por expertos, lo que facilita la realización de análisis estadísticos, manipulación de datos y visualización avanzada, entre otras aplicaciones. Esta arquitectura modular posibilita que R se adapte a las necesidades de usuarios de distintos campos, promoviendo la reutilización de código y la colaboración científica (R Core Team, 2023).

8.1 Definición y Alcance de los Paquetes en R

Un paquete en R se define como una colección estructurada de funciones, conjuntos de datos y documentación que extiende las capacidades del entorno base. Estos paquetes, desarrollados tanto por la comunidad como por equipos especializados, están orientados a resolver problemas concretos en áreas como la estadística, la ciencia de datos, la visualización gráfica y la programación. La modularidad de los paquetes permite que los usuarios seleccionen e instalen únicamente las herramientas que requieren para sus proyectos, optimizando así el uso de recursos y facilitando la actualización de funcionalidades (Wickham, 2016; R Core Team, 2023).

8.1.1 Atributos Distintivos de los Paquetes en R

Los paquetes en R presentan una serie de atributos fundamentales que los distinguen y potencian su utilidad:

1. **Funciones Especializadas:** Cada paquete contiene funciones diseñadas para abordar tareas concretas, como la generación de gráficos, la realización de análisis estadísticos avanzados o la manipulación de grandes volúmenes de datos.
2. **Documentación Integral:** Los paquetes incluyen documentación detallada que describe el propósito de cada función, sus argumentos y ejemplos de uso, lo que facilita el aprendizaje autónomo y la correcta aplicación de las herramientas.
3. **Conjuntos de Datos Ilustrativos:** Muchos paquetes incorporan conjuntos de datos de ejemplo, que permiten a los usuarios practicar y comprender la funcionalidad ofrecida, así como reproducir ejemplos y casos de estudio presentados en la documentación (Grolemund & Wickham, 2017; Xie et al., 2018).

8.1.2 Beneficios y Aplicaciones de los Paquetes en R

El aprovechamiento de los paquetes es esencial para explotar al máximo el potencial de R, ya que proporcionan extensibilidad, eficiencia y especialización. Los paquetes permiten realizar tareas que no están disponibles en el entorno base, simplifican procesos complejos y ofrecen soluciones adaptadas a necesidades específicas en disciplinas como la agronomía, la biología, la economía y muchas otras. La existencia de una comunidad activa y colaborativa asegura la actualización constante y el soporte de una amplia variedad de paquetes, lo que contribuye a mantener a R como una herramienta de referencia en el análisis de datos y la investigación reproducible (Wickham & Grolemund, 2017; R Core Team, 2023). Esta dinámica de desarrollo y mantenimiento colectivo fomenta la innovación y la rápida incorporación de nuevas metodologías y tecnologías en el ecosistema de R.

8.2 Gestión de Paquetes en R: Instalación y Carga

La gestión de paquetes en R es un proceso esencial para acceder a herramientas especializadas y ampliar las capacidades del entorno base. La mayoría de los paquetes se obtienen desde CRAN (Comprehensive R Archive Network), el repositorio oficial que alberga una amplia variedad de recursos para diferentes áreas de aplicación (R Core Team, 2023). Este proceso asegura que los usuarios puedan acceder a las funcionalidades necesarias para sus análisis y proyectos.

8.2.1 Procedimiento de Instalación de Paquetes

Para instalar un paquete desde CRAN, se utiliza la función `install.packages()`. Este proceso descarga e instala el paquete y sus dependencias en el sistema. Por ejemplo, para instalar el paquete `ggplot2`, ampliamente utilizado para la visualización de datos, se emplea la siguiente instrucción:

```
# Instalación del paquete ggplot2
install.packages("ggplot2")
```

La instalación de un paquete es un proceso que solo debe realizarse una vez en el sistema, a menos que se requiera una versión específica o se reinstale el sistema operativo.

8.2.2 Activación de Paquetes: Carga en la Sesión de Trabajo

Después de instalar un paquete, es necesario cargarlo en cada nueva sesión de trabajo para poder utilizar sus funciones. Esto se realiza mediante la función `library()`:

```
# Cargar el paquete ggplot2
library(ggplot2)
```

La carga de paquetes debe repetirse cada vez que se inicia una nueva sesión en R, ya que los paquetes no se cargan automáticamente al abrir el entorno. Este paso es crucial para que las funciones y los datos del paquete estén disponibles para su uso.

8.2.3 Automatización de la instalación y carga

Para asegurar que un paquete esté disponible y evitar errores al compartir scripts, es recomendable automatizar el proceso de verificación, instalación y carga. La siguiente estructura permite comprobar si el paquete está instalado y, en caso contrario, instalarlo y cargarlo automáticamente:

```
# Verificar e instalar automáticamente un paquete
if (!require("ggplot2")) install.packages("ggplot2")
```

Este enfoque contribuye a la reproducibilidad del código y facilita el intercambio de scripts entre usuarios, garantizando que todas las dependencias necesarias estén disponibles en el entorno de trabajo (R Core Team, 2023).

8.2.4 Alternativas para la Gestión de Paquetes

Además de las funciones básicas `install.packages()` y `library()`, existen alternativas para la gestión de paquetes que ofrecen funcionalidades adicionales:

1. **pacman:** Este paquete simplifica la instalación y carga de múltiples paquetes con una sintaxis más concisa. Por ejemplo, `p_load(ggplot2, dplyr)` instala y carga ambos paquetes simultáneamente.
2. **renv:** Este paquete permite crear entornos de proyectos reproducibles, registrando las versiones exactas de los paquetes utilizados. Esto asegura que el código funcione correctamente incluso en diferentes sistemas o en el futuro.
3. **devtools:** Este paquete facilita la instalación de paquetes desde GitHub u otras fuentes no oficiales, lo que es útil para acceder a versiones en desarrollo o paquetes personalizados.

El uso de estas herramientas puede mejorar significativamente la eficiencia y la reproducibilidad en el manejo de paquetes en R (R Core Team, 2023).

8.3 Paquetes recomendados para tareas específicas en R

En el contexto del análisis estadístico y la manipulación de datos, R dispone de una amplia variedad de paquetes que optimizan tareas especializadas y permiten realizar análisis complejos de manera eficiente. La selección adecuada de paquetes facilita la automatización de procesos, la obtención de resultados reproducibles y la adaptación a diferentes áreas de aplicación (R Core Team, 2023).

A continuación, se presenta una clasificación de los paquetes más relevantes, organizados por su área de aplicación y acompañados de una breve descripción de sus funcionalidades principales:

Área	Paquete	Descripción
Manipulación de datos	`dplyr`	Facilita la transformación y manipulación de datos mediante funciones intuitivas
	`tidyr`	Permite reorganizar datos entre formatos ancho y largo
	`data.table`	Optimizado para el manejo de grandes conjuntos de datos
Análisis exploratorio	`DataExplorer`	Automatiza el análisis exploratorio de datos
	`summarytools`	Genera resúmenes estadísticos detallados
	`psych`	Proporciona funciones para análisis psicométricos y estadística descriptiva
Análisis estadístico	`stats`	Incluye funciones base para pruebas estadísticas comunes
	`agricolae`	Especializado en diseños experimentales y análisis agrícolas
	`AgroR`	Proporciona funciones y herramientas para análisis estadísticos en agronomía
	`car`	Facilita análisis de regresión avanzados
Visualización	`ggplot2`	Permite crear gráficos personalizados de alta calidad
	`plotly`	Genera gráficos interactivos

Esta clasificación permite identificar rápidamente los paquetes más adecuados para cada etapa del análisis de datos, desde la manipulación inicial hasta la visualización y el análisis especializado.

8.3.1 Instalación y carga de paquetes esenciales

Para facilitar el inicio de un proyecto de análisis estadístico, se recomienda instalar y cargar un conjunto básico de paquetes que cubran las principales necesidades de manipulación, exploración, análisis y visualización de datos. El siguiente código muestra cómo automatizar este proceso:

```
# Paquetes para manipulación y visualización de datos
if (!require("tidyverse")) install.packages("tidyverse")
```

```r
if (!require("data.table")) install.packages("data.table")

# Paquetes para análisis exploratorio
if (!require("DataExplorer")) install.packages("DataExplorer")
if (!require("psych")) install.packages("psych")

# Paquetes para análisis estadísticos especializados
if (!require("agricolae")) install.packages("agricolae")
if (!require("AgroR")) install.packages("AgroR")
if (!require("car")) install.packages("car")

# Paquetes para manejo de archivos
if (!require("readxl")) install.packages("readxl")
if (!require("writexl")) install.packages("writexl")
```

Este conjunto de instrucciones garantiza que los paquetes esenciales estén disponibles en el entorno de trabajo, contribuyendo a la reproducibilidad y facilitando el intercambio de scripts entre usuarios. Además, la automatización de la instalación y carga de paquetes minimiza errores y asegura que todas las dependencias necesarias se encuentren correctamente configuradas (R Core Team, 2023).

Capítulo III

Manipulación de datos

9 Introducción a la manipulación de datos en R

La manipulación de datos representa una etapa crítica en el proceso de análisis estadístico, ya que los datos raramente se encuentran en condiciones óptimas para su análisis inmediato. Es común que los conjuntos de datos contengan errores, valores faltantes, duplicados o estén organizados de manera poco conveniente para los objetivos del estudio. Por ello, la manipulación de datos es esencial para transformar los datos crudos en información útil, confiable y lista para el análisis estadístico y la visualización. Sin una adecuada manipulación, los resultados pueden ser erróneos o difíciles de interpretar, lo que afecta la validez y la reproducibilidad de los análisis (Wickham & Grolemund, 2017; R Core Team, 2023).

9.1 Principales tareas de manipulación de datos

La manipulación de datos en R abarca un conjunto de tareas fundamentales que permiten preparar la información para su análisis estadístico. Estas tareas son necesarias para garantizar que los datos sean consistentes, completos y estén organizados de acuerdo con los requerimientos del análisis a realizar (Wickham & Grolemund, 2017).

9.1.1 Filtrado de datos

El filtrado consiste en seleccionar subconjuntos de datos que cumplen ciertas condiciones específicas. Esta tarea es fundamental para enfocar el análisis en grupos de interés, eliminar registros no relevantes o excluir observaciones que puedan distorsionar los resultados. Por ejemplo, se puede filtrar una base de datos para analizar únicamente los registros de un grupo demográfico particular o eliminar casos con información incompleta (R Core Team, 2023).

9.1.2 Selección de variables

La selección de variables implica elegir únicamente las columnas o variables relevantes para el análisis. Esta tarea simplifica el conjunto de datos, reduce la complejidad del análisis y facilita la interpretación de los resultados. Seleccionar las variables adecuadas es especialmente importante cuando se trabaja con bases de datos extensas o con información redundante (Wickham & Grolemund, 2017).

9.1.3 Transformación de datos

La transformación de datos abarca la creación de nuevas variables, la modificación de valores existentes o la recodificación de categorías. Estas operaciones permiten adaptar los datos a los requerimientos del análisis estadístico, por ejemplo, convirtiendo variables categóricas en numéricas, calculando índices o agrupando categorías similares. La transformación es clave para preparar los datos antes de aplicar técnicas estadísticas específicas (R Core Team, 2023).

9.1.4 Agregación de información

La agregación consiste en resumir la información contenida en los datos, calculando medidas como promedios, totales, conteos o proporciones por grupo. Esta tarea es fundamental para comparar tendencias, identificar patrones y realizar análisis descriptivos o inferenciales. La agregación permite sintetizar grandes volúmenes de datos en resúmenes comprensibles y útiles para la toma de decisiones (Wickham & Grolemund, 2017).

9.1.5 Reestructuración de datos

La reestructuración de datos implica cambiar la forma en que los datos están organizados, por ejemplo, convirtiendo un conjunto de datos de formato ancho a largo o viceversa. Esta tarea es necesaria cuando la estructura original de los datos no es compatible con las técnicas estadísticas o de visualización que se desean aplicar. La reestructuración facilita la aplicación de modelos y la generación de gráficos adecuados (Wickham & Grolemund, 2017).

9.2 Enfoques disponibles en R para la manipulación de datos

R ofrece dos enfoques principales para la manipulación de datos, cada uno con características y ventajas particulares que se adaptan a diferentes necesidades y niveles de experiencia del usuario (R Core Team, 2023).

9.2.1 Herramientas base de R

Las herramientas base de R incluyen funciones integradas como el uso de corchetes para seleccionar filas y columnas, así como funciones como subset(), aggregate() y tapply(). Estas herramientas permiten realizar operaciones fundamentales de manipulación de datos de manera flexible y eficiente. Sin embargo, la sintaxis puede resultar menos intuitiva para quienes se inician en R, y las operaciones complejas pueden requerir múltiples pasos o combinaciones de funciones (R Core Team, 2023).

9.2.2 El enfoque tidyverse

El tidyverse es un conjunto de paquetes desarrollados para simplificar y estandarizar la manipulación de datos en R. Entre estos paquetes destacan dplyr y tidyr, que ofrecen funciones específicas para filtrar, seleccionar, transformar y reestructurar datos de manera clara y legible. El uso del tidyverse facilita la construcción de flujos de trabajo reproducibles y eficientes, y su sintaxis está diseñada para ser accesible tanto para principiantes como para usuarios avanzados. Además, el tidyverse promueve el principio de "datos ordenados" (tidy data), que facilita la aplicación de técnicas estadísticas y la generación de visualizaciones (Wickham & Grolemund, 2017).

10 Manipulación de Datos con Herramientas Base de R

La manipulación de datos con herramientas base de R constituye una etapa esencial en el flujo de trabajo estadístico clásico. Antes de aplicar técnicas como el análisis de varianza, la regresión lineal o la comparación de medias, es necesario preparar los datos para asegurar su integridad y adecuación al análisis. Las funciones básicas de R permiten seleccionar, filtrar, transformar, agrupar y limpiar datos de manera eficiente, facilitando la obtención de resultados estadísticos válidos y reproducibles (R Core Team, 2023).

10.1 Datos de ejemplo

Para ilustrar las técnicas de manipulación, se emplea un conjunto de datos simulado que representa un experimento agrícola. Este conjunto contiene variables numéricas y categóricas, así como algunos valores faltantes, lo que permite demostrar operaciones comunes en la estadística clásica. El uso de datos simulados garantiza la reproducibilidad de los ejemplos y facilita la comprensión de los procedimientos (Wickham & Grolemund, 2017).

El objetivo de este ejemplo es mostrar cómo crear y explorar un conjunto de datos en R, identificando la estructura de las variables y la presencia de valores faltantes, aspectos fundamentales en la preparación de datos para el análisis estadístico.

Creación del conjunto de datos simulado

```
# Establecer una semilla para que el usuario pueda replicar el ejemplo
set.seed(123) # Garantiza reproducibilidad

# Simular los resultados de un experimento
# con el diseño bloques completos al azar
datos_cultivo <- data.frame(
  parcela = 1:20,
  tratamiento = rep(c("Control",
                      "Fertilizante A",
                      "Fertilizante B",
                      "Fertilizante C"), each = 5),
  bloque = rep(1:5, times = 4),
  altura_cm = round(rnorm(20, mean = 65, sd = 10), 1),
  peso_gr = round(rnorm(20, mean = 120, sd = 25), 1),
  daño_plaga = sample(c("Alto", "Medio", "Bajo"), 20, replace = TRUE),
  fecha_siembra = as.Date("2024-01-01") + sample(1:10, 20, replace = TRUE)
)
```

En este bloque de código se crea un data frame denominado `datos_cultivo` que simula los resultados de un experimento agrícola bajo un diseño de bloques completos al azar. Las variables incluyen identificadores de parcela, tipo de tratamiento aplicado, bloque experimental, altura y peso de las plantas, nivel de daño por plaga y fecha de siembra. La función `set.seed(123)` asegura que los resultados sean reproducibles, permitiendo que cualquier usuario obtenga el mismo conjunto de datos al ejecutar el código.

Simulación de valores faltantes

```
# Simular la presencia de datos faltantes en los resultados del experimento
datos_cultivo$altura_cm[c(3, 15)] <- NA
datos_cultivo$peso_gr[c(7, 18)] <- NA
```

Se introducen valores faltantes (NA) en las variables `altura_cm` y `peso_gr` para reflejar situaciones reales en las que los datos pueden estar incompletos. Este paso es fundamental para demostrar técnicas de manejo de datos faltantes en secciones posteriores.

Visualización preliminar de los datos

```
# Visualizar las primeras filas del data frame con los datos simulados
head(datos_cultivo)
```

```
  parcela    tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
1       1        Control      1      59.4    93.3      Medio    2024-01-10
2       2        Control      2      62.7   114.6      Medio    2024-01-08
3       3        Control      3        NA    94.3       Bajo    2024-01-04
4       4        Control      4      65.7   101.8      Medio    2024-01-09
5       5        Control      5      66.3   104.4      Medio    2024-01-10
6       6 Fertilizante A      1      82.2    77.8       Bajo    2024-01-04
```

La función `head()` permite observar las primeras seis filas del data frame, facilitando la verificación de la correcta creación de las variables y la identificación de valores faltantes.

Es importante revisar la estructura y el contenido de los datos antes de proceder con cualquier análisis, ya que la presencia de valores faltantes o inconsistencias puede afectar la validez de los resultados estadísticos. Para ello también se puede usar la función `view()`, que nos permite visualizar un objeto en su totalidad en formato tabular (R Core Team, 2023).

10.2 Selección y filtrado de datos en data frames

La selección y el filtrado de datos constituyen operaciones fundamentales en la manipulación de data frames en R, ya que permiten enfocar el análisis en subconjuntos de información relevantes para los objetivos estadísticos planteados. Estas tareas se realizan mediante la indexación y el uso de condiciones lógicas, lo que facilita la extracción precisa de variables y observaciones de interés (Wickham & Grolemund, 2017; R Core Team, 2023).

10.2.1 Selección de columnas

En R, la selección de columnas dentro de un data frame se efectúa utilizando la notación de corchetes `[, ]`, donde el primer argumento corresponde a las filas y el segundo a las columnas. Por ejemplo, para extraer únicamente las variables de altura y peso del conjunto de datos simulado, se emplea el siguiente código:

```
# Seleccionar solo las columnas de altura y peso del data frame
datos_mediciones <- datos_cultivo[, c("altura_cm", "peso_gr")]

# Visualizar las primeras filas para verificar la selección
head(datos_mediciones)
```

```
  altura_cm peso_gr
1      59.4    93.3
2      62.7   114.6
3        NA    94.3
4      65.7   101.8
5      66.3   104.4
6      82.2    77.8
```

En este caso, se especifican los nombres de las columnas deseadas dentro de un vector, lo que permite obtener un nuevo data frame que contiene exclusivamente las mediciones de altura y peso. Esta práctica es recomendable, ya que el uso de nombres de columnas, en lugar de posiciones numéricas, previene errores en caso de que la estructura del data frame cambie posteriormente.

La exclusión de columnas también puede realizarse de manera eficiente. Por ejemplo, si se requiere eliminar la variable de fecha de siembra, se puede utilizar la función `names()` junto con el operador `%in%` para identificar y excluir la columna correspondiente:

```
# Excluir la columna fecha_siembra del data frame
datos_sin_fecha <- datos_cultivo[, !names(datos_cultivo)
                                  %in% "fecha_siembra"]

# Verificar que la columna fecha_siembra ha sido eliminada
head(datos_sin_fecha)
```

```
  parcela    tratamiento bloque altura_cm peso_gr daño_plaga
1       1        Control      1      59.4    93.3      Medio
2       2        Control      2      62.7   114.6      Medio
3       3        Control      3        NA    94.3       Bajo
4       4        Control      4      65.7   101.8      Medio
5       5        Control      5      66.3   104.4      Medio
6       6 Fertilizante A      1      82.2    77.8       Bajo
```

Este procedimiento genera un nuevo data frame en el que la columna “fecha_siembra” ha sido removida, manteniendo el resto de las variables intactas. La visualización de las primeras filas permite comprobar que la exclusión se ha realizado correctamente.

Asimismo, la selección de columnas puede basarse en la posición que ocupan dentro del data frame:

```
# Seleccionar las tres primeras columnas usando índices numéricos
primeras_tres_columnas <- datos_cultivo[, 1:3]

# Mostrar el resultado para verificar la selección
head(primeras_tres_columnas)
```

```
  parcela    tratamiento bloque
1       1        Control      1
2       2        Control      2
3       3        Control      3
4       4        Control      4
5       5        Control      5
6       6 Fertilizante A      1
```

Este método resulta útil en situaciones donde se conoce la estructura del data frame y se requiere trabajar con un subconjunto de variables contiguas. Sin embargo, es importante considerar que la selección por posición puede ser menos robusta ante modificaciones en el orden de las columnas, por lo que se recomienda preferir la selección por nombre cuando sea posible.

10.2.2 Filtrado de filas por condiciones lógicas

El filtrado de filas en un data frame se realiza especificando una condición lógica en el argumento correspondiente a las filas dentro de la notación de corchetes. Por ejemplo, para seleccionar únicamente las observaciones que corresponden al tratamiento "Control", se utiliza la siguiente instrucción:

```
# Filtrar las filas donde el tratamiento es "Control"
datos_control <- datos_cultivo[datos_cultivo$tratamiento == "Control", ]

# Verificar el filtrado mostrando las primeras filas
head(datos_control)
```

```
  parcela tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
1       1     Control      1      59.4    93.3      Medio    2024-01-10
2       2     Control      2      62.7   114.6      Medio    2024-01-08
3       3     Control      3        NA    94.3       Bajo    2024-01-04
4       4     Control      4      65.7   101.8      Medio    2024-01-09
5       5     Control      5      66.3   104.4      Medio    2024-01-10
```

En este caso, la condición `datos_cultivo$tratamiento == "Control"` evalúa cada fila del data frame y selecciona aquellas en las que la variable "tratamiento" coincide con el valor especificado. El resultado es un nuevo data frame que contiene únicamente las

observaciones del grupo de control, lo que facilita la comparación o el análisis específico de este subconjunto.

El filtrado puede combinar múltiples condiciones lógicas para refinar aún más la selección de datos. Por ejemplo, si se desea obtener las observaciones en las que la altura es mayor a 65 centímetros y el tratamiento es distinto de "Control", se puede emplear el siguiente código:

```
# Filtrar filas que cumplen dos condiciones simultáneamente
datos_altos <- datos_cultivo[datos_cultivo$altura_cm > 65 &
                             datos_cultivo$tratamiento != "Control", ]

# Verificar el resultado del filtrado
head(datos_altos)
```

```
   parcela    tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
6        6 Fertilizante A      1      82.2    77.8       Bajo    2024-01-04
7        7 Fertilizante A      2      69.6      NA       Bajo    2024-01-08
11      11 Fertilizante B      1      77.2   130.7       Alto    2024-01-11
12      12 Fertilizante B      2      68.6   112.6      Medio    2024-01-06
13      13 Fertilizante B      3      69.0   142.4       Alto    2024-01-06
14      14 Fertilizante B      4      66.1   142.0       Alto    2024-01-09
```

Aquí, la condición compuesta utiliza el operador lógico `&` para exigir que ambas condiciones se cumplan simultáneamente. De este modo, se obtiene un subconjunto de datos que cumple criterios específicos de interés para el análisis.

Para mejorar la legibilidad y evitar la repetición del nombre del data frame en cada condición, R ofrece la función `subset()`, que permite expresar las condiciones de filtrado de manera más clara. Por ejemplo, para seleccionar las observaciones correspondientes al tratamiento "Fertilizante A" y a los bloques 2, 3 o 4, se puede utilizar la siguiente instrucción:

```
# Usar subset() para filtrar datos de manera más legible
datos_fertilizante_A <- subset(datos_cultivo,
                               tratamiento == "Fertilizante A" &
                               bloque %in% c(2,3,4))

# Mostrar el resultado del filtrado
head(datos_fertilizante_A)
```

```
  parcela    tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
7       7 Fertilizante A      2      69.6      NA       Bajo    2024-01-08
8       8 Fertilizante A      3      52.3   123.8       Alto    2024-01-04
9       9 Fertilizante A      4      58.1    91.5      Medio    2024-01-08
```

La función `subset()` interpreta las condiciones dentro del contexto del data frame especificado, lo que simplifica la sintaxis y reduce la posibilidad de errores. Esta función

resulta especialmente útil cuando se trabaja con condiciones complejas o con múltiples variables.

La correcta aplicación de estas técnicas de selección y filtrado permite preparar conjuntos de datos ajustados a los requerimientos de los análisis estadísticos, optimizando la eficiencia y la claridad en la manipulación de la información (R Core Team, 2023; Wickham & Grolemund, 2017).

10.3 Modificación de variables

La modificación de variables constituye una etapa crucial en la preparación de datos para análisis estadísticos, ya que permite adaptar la información a los requerimientos específicos de diferentes métodos analíticos. Esta transformación de datos puede incluir la creación de nuevas variables derivadas, la recodificación de variables categóricas y la aplicación de transformaciones matemáticas para cumplir con los supuestos de las técnicas estadísticas (Wickham & Grolemund, 2017; R Core Team, 2023).

10.3.1 Creación de nuevas columnas

En R, la creación de nuevas variables se realiza mediante operaciones aritméticas o lógicas sobre las columnas existentes del data frame. Estas transformaciones permiten generar índices, categorizar datos continuos o aplicar transformaciones matemáticas para mejorar la normalidad de las distribuciones:

```
# Crear un índice que relaciona altura y peso
# La división genera una nueva variable que es el indice de crecimiento
datos_cultivo$indice_crecimiento <-
  datos_cultivo$altura_cm/datos_cultivo$peso_gr

# Categorizar el peso en dos niveles usando una condición lógica
# ifelse() evalúa cada valor y asigna una categoría según la condición
datos_cultivo$categoria_peso <- ifelse(datos_cultivo$peso_gr > 120,
                                       "Alto", "Bajo")

# Aplicar transformación logarítmica para normalizar la distribución
# log() es útil para variables con asimetría positiva
datos_cultivo$log_peso <- log(datos_cultivo$peso_gr)

# Verificar las nuevas variables creadas
head(datos_cultivo)
```

```
  parcela    tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
1       1        Control      1      59.4    93.3      Medio    2024-01-10
2       2        Control      2      62.7   114.6      Medio    2024-01-08
3       3        Control      3        NA    94.3       Bajo    2024-01-04
4       4        Control      4      65.7   101.8      Medio    2024-01-09
5       5        Control      5      66.3   104.4      Medio    2024-01-10
```

```
6       6 Fertilizante A      1      82.2    77.8       Bajo    2024-01-04
  indice_crecimiento categoria_peso log_peso
1          0.6366559           Bajo 4.535820
2          0.5471204           Bajo 4.741448
3                 NA           Bajo 4.546481
4          0.6453831           Bajo 4.623010
5          0.6350575           Bajo 4.648230
6          1.0565553           Bajo 4.354141
```

La creación de estas nuevas variables permite enriquecer el análisis y adaptar los datos a diferentes necesidades analíticas. Por ejemplo, la transformación logarítmica es especialmente útil cuando se requiere normalizar distribuciones asimétricas para análisis paramétricos, mientras que la categorización facilita la comparación entre grupos.

10.3.2 Recodificación de variables categóricas

La recodificación de variables categóricas es fundamental para establecer un orden específico en los niveles de los factores o para simplificar categorías, lo cual es particularmente relevante en análisis como ANOVA o regresión logística:

```
# Recodificar los niveles de daño por plaga como valores numéricos
## factor() permite especificar el orden de los niveles
## y asignar nuevas etiquetas
datos_cultivo$nivel_daño <- factor(datos_cultivo$daño_plaga,
                                   levels = c("Bajo", "Medio", "Alto"),
                                   labels = c("1", "2", "3"))

# Crear una variable indicadora (dummy) para el tratamiento control
# Esta transformación es útil para análisis de regresión
datos_cultivo$es_control <-
  ifelse(datos_cultivo$tratamiento == "Control", 1, 0)

# Verificar la recodificación
head(datos_cultivo)
```

```
  parcela    tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
1       1        Control      1      59.4    93.3      Medio    2024-01-10
2       2        Control      2      62.7   114.6      Medio    2024-01-08
3       3        Control      3        NA    94.3       Bajo    2024-01-04
4       4        Control      4      65.7   101.8      Medio    2024-01-09
5       5        Control      5      66.3   104.4      Medio    2024-01-10
6       6 Fertilizante A      1      82.2    77.8       Bajo    2024-01-04
  indice_crecimiento categoria_peso log_peso nivel_daño es_control
1          0.6366559           Bajo 4.535820          2          1
2          0.5471204           Bajo 4.741448          2          1
3                 NA           Bajo 4.546481          1          1
4          0.6453831           Bajo 4.623010          2          1
5          0.6350575           Bajo 4.648230          2          1
6          1.0565553           Bajo 4.354141          1          0
```

La función `factor()` es especialmente útil en este contexto, ya que permite no solo recodificar los valores, sino también establecer un orden específico en los niveles de la variable categórica. Esto es crucial en análisis donde el orden de los niveles afecta la interpretación de los resultados, como en el caso de variables ordinales.

10.4 Ordenamiento y agrupamiento de datos

El ordenamiento y agrupamiento de datos constituyen operaciones fundamentales en el análisis exploratorio de datos, facilitando la identificación de patrones, la detección de valores atípicos y el cálculo de estadísticas descriptivas por grupos. Estas operaciones son especialmente relevantes en la estadística clásica, donde la estructura y organización de los datos influyen directamente en la calidad de los análisis posteriores (Wickham & Grolemund, 2017; R Core Team, 2023).

10.4.1 Ordenamiento de datos

R proporciona herramientas eficientes para ordenar datos mediante la función `order()`, que genera índices de ordenamiento que pueden aplicarse a las filas del data frame. Este ordenamiento puede realizarse por una o múltiples variables, en orden ascendente o descendente.

```
# Ordenar el data frame por altura de manera ascendente
# order() genera índices basados en los valores de altura_cm
# Los índices se utilizan para reordenar todas las filas
datos_ordenados <- datos_cultivo[order(datos_cultivo$altura_cm), ]

# Verificar el ordenamiento mostrando las primeras filas
head(datos_ordenados)
```

```
   parcela    tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
18      18 Fertilizante C      3      45.3      NA       Alto    2024-01-11
8        8 Fertilizante A      3      52.3   123.8       Alto    2024-01-04
9        9 Fertilizante A      4      58.1    91.5      Medio    2024-01-08
1        1        Control      1      59.4    93.3      Medio    2024-01-10
20      20 Fertilizante C      5      60.3   110.5       Alto    2024-01-11
10      10 Fertilizante A      5      60.5   151.3      Medio    2024-01-07
   indice_crecimiento categoria_peso log_peso nivel_daño es_control
18                 NA           <NA>       NA          3          0
8           0.4224556           Alto 4.818667          3          0
9           0.6349727           Bajo 4.516339          2          0
1           0.6366559           Bajo 4.535820          2          1
20          0.5457014           Bajo 4.705016          3          0
10          0.3998678           Alto 5.019265          2          0
```

En este ejemplo, la función `order(datos_cultivo$altura_cm)` genera un vector de índices que indica el orden en el que deben organizarse las filas del data frame

`datos_cultivo` para que la columna `altura_cm` quede ordenada de manera ascendente. Estos índices se utilizan para reordenar las filas del data frame, y el resultado se almacena en `datos_ordenados`. La función `head()` muestra las primeras filas del data frame ordenado para verificar el resultado.

```
# Ordenar por múltiples criterios:
# Por tratamiento y por peso descendente
datos_ordenados_multi <- datos_cultivo[order(
  datos_cultivo$tratamiento,
  -datos_cultivo$peso_gr), ]

# Verificar el ordenamiento múltiple
head(datos_ordenados_multi)
```

```
   parcela    tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
2        2        Control      2      62.7   114.6      Medio    2024-01-08
5        5        Control      5      66.3   104.4      Medio    2024-01-10
4        4        Control      4      65.7   101.8      Medio    2024-01-09
3        3        Control      3        NA    94.3       Bajo    2024-01-04
1        1        Control      1      59.4    93.3      Medio    2024-01-10
10      10 Fertilizante A      5      60.5   151.3      Medio    2024-01-07
   indice_crecimiento categoria_peso log_peso nivel_daño es_control
2           0.5471204           Bajo 4.741448          2          1
5           0.6350575           Bajo 4.648230          2          1
4           0.6453831           Bajo 4.623010          2          1
3                  NA           Bajo 4.546481          1          1
1           0.6366559           Bajo 4.535820          2          1
10          0.3998678           Alto 5.019265          2          0
```

En este caso, se realiza un ordenamiento por múltiples criterios. Primero, se ordena por la columna `tratamiento` de manera ascendente (orden alfabético). Luego, dentro de cada grupo de tratamiento, se ordena por la columna `peso_gr` de manera descendente. El signo negativo `-` delante de `datos_cultivo$peso_gr` indica que el ordenamiento debe ser descendente para esta variable.

El ordenamiento por múltiples variables es particularmente útil cuando se necesita establecer una jerarquía en la organización de los datos, como por ejemplo, ordenar primero por tratamiento y luego por una variable de respuesta.

10.4.2 Cálculo de estadísticas por grupo

El análisis por grupos es esencial en la estadística experimental, permitiendo comparar tratamientos y evaluar la variabilidad dentro y entre grupos. R ofrece varias funciones para realizar estos cálculos:

```
# Calcular medias por tratamiento usando tapply
# tapply aplica la función mean a cada subconjunto definido por tratamiento
medias_altura <- tapply(datos_cultivo$altura_cm,
```

```
                        datos_cultivo$tratamiento,
                        mean,
                        na.rm = TRUE)
```

La salida muestra las medias de altura para cada tratamiento:

```
# Mostrar las medias por tratamiento
print(medias_altura)
```

```
       Control Fertilizante A Fertilizante B Fertilizante C
        63.525         64.540         70.225         66.100
```

Para un análisis más completo, se pueden calcular múltiples estadísticas simultáneamente utilizando la función `aggregate()`:

```
# Calcular media y desviación estándar por tratamiento
# aggregate permite trabajar con múltiples variables y funciones
estadisticas_grupo <- aggregate(cbind(altura_cm, peso_gr) ~ tratamiento,
                                data = datos_cultivo,
                                FUN = function(x)
                                  c(media = mean(x, na.rm = TRUE),
                                    sd = sd(x, na.rm = TRUE)))
```

Los resultados muestran estadísticas descriptivas por tratamiento:

```
# Mostrar los resultados del análisis por grupo
print(estadisticas_grupo)
```

```
     tratamiento altura_cm.media altura_cm.sd peso_gr.media peso_gr.sd
1        Control       63.525000     3.168990    103.525000   8.773967
2 Fertilizante A       63.275000    13.077812    111.100000  33.017066
3 Fertilizante B       70.225000     4.823812    131.925000  13.978406
4 Fertilizante C       71.300000     9.268945    123.475000  13.976021
```

Esta salida proporciona una visión completa de la variabilidad en las mediciones de altura y peso para cada tratamiento, revelando que:

1. El Fertilizante B y C muestran las mayores medias en altura
2. El Fertilizante A presenta la mayor variabilidad en altura (SD = 13.08)
3. Los tratamientos con fertilizantes muestran mayores pesos medios que el control

Estos resultados preliminares son fundamentales para guiar análisis estadísticos más detallados, como ANOVA o comparaciones múltiples, y para identificar posibles patrones o anomalías en los datos experimentales (R Core Team, 2023).

La combinación de técnicas de ordenamiento y agrupamiento proporciona una base sólida para la exploración inicial de datos experimentales, permitiendo identificar patrones, detectar valores atípicos y generar hipótesis para análisis posteriores. Esta preparación cuidadosa de los datos es esencial para garantizar la validez y robustez de las conclusiones estadísticas.

10.5 Manejo de valores faltantes y duplicados

El tratamiento adecuado de valores faltantes y registros duplicados constituye un paso fundamental en la preparación de datos para análisis estadísticos. Esta etapa es crucial para garantizar la validez de los resultados y evitar sesgos en las estimaciones estadísticas. Los valores faltantes pueden afectar significativamente los análisis si no se manejan apropiadamente, mientras que los duplicados pueden inflar artificialmente el tamaño de la muestra y distorsionar las conclusiones (Wickham & Grolemund, 2017; R Core Team, 2023).

10.5.1 Identificación y manejo de valores faltantes

En R, existen diversas herramientas para identificar y tratar los valores faltantes (NA). La estrategia de manejo debe seleccionarse cuidadosamente según el contexto del estudio y el impacto potencial en los análisis posteriores:

```
# Realizar un diagnóstico inicial de valores faltantes por columna
# is.na() identifica valores NA en cada celda
# colSums() suma los valores TRUE (NA) en cada columna
na_por_columna <- colSums(is.na(datos_cultivo))

# Mostrar el resultado del diagnóstico
print(na_por_columna)
```

```
          parcela       tratamiento            bloque         altura_cm
                0                 0                 0                 2
         peso_gr        daño_plaga     fecha_siembra indice_crecimiento
                2                 0                 0                 4
   categoria_peso          log_peso       nivel_daño        es_control
                2                 2                 0                 0
```

Este diagnóstico inicial permite identificar las variables más afectadas por valores faltantes y determinar la estrategia más apropiada para su tratamiento. Una vez identificados los valores faltantes, se pueden aplicar diferentes métodos de manejo:

```
# Eliminar filas que contengan cualquier valor faltante
# na.omit() crea un nuevo data frame excluyendo filas con NA
datos_completos <- na.omit(datos_cultivo)

# Imputar valores faltantes utilizando la media de la variable
# Este método es común pero debe usarse con precaución
datos_cultivo$altura_cm[is.na(datos_cultivo$altura_cm)] <-
  mean(datos_cultivo$altura_cm, na.rm = TRUE)

# Verificar el resultado de la imputación
summary(datos_cultivo$altura_cm)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  45.30   60.45   66.06   66.01   69.70   82.90
```

La eliminación de filas con valores faltantes (caso completo) es una solución simple pero puede resultar en pérdida significativa de información. La imputación con la media es una alternativa común, aunque puede subestimar la variabilidad real de los datos. En estudios más rigurosos, se pueden considerar métodos de imputación más sofisticados basados en modelos estadísticos.

10.5.2 Identificación y manejo de duplicados

Los registros duplicados pueden surgir por diversos motivos, desde errores en la entrada de datos hasta repeticiones intencionales en el diseño experimental. Su identificación y manejo apropiado es esencial para mantener la integridad del análisis:

```
# Identificar duplicados basados en variables específicas
# duplicated() evalúa si cada fila es una duplicación de una fila anterior
duplicados <- duplicated(datos_cultivo[, c("tratamiento", "bloque")])

# Analizar la presencia de duplicados
summary(duplicados)
```

```
   Mode   FALSE
logical      20
```

```
# Mostrar las filas duplicadas para su inspección
datos_cultivo[duplicados, ]
```

```
 [1] parcela            tratamiento        bloque             altura_cm
 [5] peso_gr            daño_plaga         fecha_siembra      indice_crecimiento
 [9] categoria_peso     log_peso           nivel_daño         es_control
<0 rows> (o 0- extensión row.names)
```

```
# Eliminar duplicados manteniendo la primera ocurrencia
# unique() conserva solo las filas únicas del data frame
datos_sin_duplicados <- unique(datos_cultivo)

# Verificar la reducción en el número de filas
nrow(datos_cultivo) - nrow(datos_sin_duplicados)
```

```
[1] 0
```

La identificación de duplicados puede realizarse considerando todas las variables o solo un subconjunto relevante para el análisis. Es importante distinguir entre duplicados verdaderos (errores) y repeticiones válidas en el diseño experimental.

10.5.3 Verificación de la integridad de los datos

Después de abordar los valores faltantes y duplicados, es crucial verificar la integridad general de los datos para asegurar la calidad y fiabilidad del análisis posterior. Esta verificación permite identificar posibles sesgos o inconsistencias introducidas durante el proceso de limpieza y transformación de los datos.

```
# Realizar un resumen estadístico completo
summary(datos_sin_duplicados)
```

```
    parcela      tratamiento            bloque    altura_cm        peso_gr
 Min.   : 1.00   Length:20          Min.   :1   Min.   :45.30   Min.   : 77.8
 1st Qu.: 5.75   Class :character   1st Qu.:2   1st Qu.:60.45   1st Qu.:102.5
 Median :10.50   Mode  :character   Median :3   Median :66.06   Median :113.6
 Mean   :10.50                      Mean   :3   Mean   :66.01   Mean   :117.5
 3rd Qu.:15.25                      3rd Qu.:4   3rd Qu.:69.70   3rd Qu.:136.3
 Max.   :20.00                      Max.   :5   Max.   :82.90   Max.   :151.3
                                                                NA's   :2
  daño_plaga        fecha_siembra        indice_crecimiento categoria_peso
 Length:20          Min.   :2024-01-03   Min.   :0.3999     Length:20
 Class :character   1st Qu.:2024-01-04   1st Qu.:0.5135     Class :character
 Mode  :character   Median :2024-01-08   Median :0.5974     Mode  :character
                    Mean   :2024-01-07   Mean   :0.5901
                    3rd Qu.:2024-01-10   3rd Qu.:0.6355
                    Max.   :2024-01-11   Max.   :1.0566
                                         NA's   :4
    log_peso     nivel_daño   es_control
 Min.   :4.354   1:5        Min.   :0.00
 1st Qu.:4.629   2:9        1st Qu.:0.00
 Median :4.733   3:6        Median :0.00
 Mean   :4.750              Mean   :0.25
 3rd Qu.:4.915              3rd Qu.:0.25
 Max.   :5.019              Max.   :1.00
 NA's   :2
```

La función `summary()` proporciona un resumen estadístico de cada variable en el data frame `datos_sin_duplicados`. Este resumen incluye información como el mínimo, el primer cuartil, la mediana, la media, el tercer cuartil y el máximo para las variables numéricas, así como la frecuencia de cada categoría para las variables categóricas. La presencia de valores `NA` (Not Available) indica la cantidad de valores faltantes en cada variable.

```
# Verificar la estructura del data frame resultante
str(datos_sin_duplicados)
```

```
'data.frame':   20 obs. of  12 variables:
 $ parcela           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ tratamiento       : chr  "Control" "Control" "Control" "Control" ...
```

```
 $ bloque            : int  1 2 3 4 5 1 2 3 4 5 ...
 $ altura_cm         : num  59.4 62.7 66 65.7 66.3 ...
 $ peso_gr           : num  93.3 114.6 94.3 101.8 104.4 ...
 $ daño_plaga        : chr  "Medio" "Medio" "Bajo" "Medio" ...
 $ fecha_siembra     : Date, format: "2024-01-10" "2024-01-08" ...
 $ indice_crecimiento: num  0.637 0.547 NA 0.645 0.635 ...
 $ categoria_peso    : chr  "Bajo" "Bajo" "Bajo" "Bajo" ...
 $ log_peso          : num  4.54 4.74 4.55 4.62 4.65 ...
 $ nivel_daño        : Factor w/ 3 levels "1","2","3": 2 2 1 2 2 1 1 3 2 2 ...
 $ es_control        : num  1 1 1 1 1 0 0 0 0 0 ...
```

La función `str()` muestra la estructura interna del data frame, incluyendo el tipo de cada variable (numérica, carácter, factor, fecha, etc.) y las primeras observaciones. Esta función es útil para confirmar que las variables tienen el tipo de dato esperado y que no hay errores en la importación o transformación de los datos.

```r
# Calcular el porcentaje de datos completos por variable
porcentaje_completos <- colMeans(!is.na(datos_sin_duplicados)) * 100
print(porcentaje_completos)
```

```
          parcela       tratamiento            bloque          altura_cm 
              100               100               100               100 
         peso_gr        daño_plaga     fecha_siembra indice_crecimiento 
              90               100               100                80 
   categoria_peso         log_peso       nivel_daño        es_control 
              90                90               100               100 
```

En este bloque de código, se calcula el porcentaje de datos completos (no faltantes) para cada variable. La función `is.na()` devuelve un valor lógico (`TRUE` o `FALSE`) indicando si cada valor es faltante o no. El operador `!` invierte estos valores lógicos, de modo que `!is.na()` devuelve `TRUE` para los valores no faltantes y `FALSE` para los valores faltantes. La función `colMeans()` calcula la media de estos valores lógicos por columna, lo que equivale al porcentaje de valores no faltantes. Finalmente, se multiplica por 100 para expresar el resultado como un porcentaje.

Esta verificación final permite asegurar que el tratamiento de valores faltantes y duplicados no ha introducido sesgos o inconsistencias en los datos. Es fundamental documentar todas las decisiones tomadas en este proceso, ya que pueden afectar la interpretación de los resultados posteriores.

La gestión adecuada de valores faltantes y duplicados es un paso crítico en la preparación de datos para análisis estadísticos. Las decisiones tomadas en esta etapa deben basarse en un entendimiento profundo del contexto del estudio y documentarse apropiadamente para garantizar la reproducibilidad del análisis (R Core Team, 2023).

11 Manipulación de datos con dplyr y tidyr

11.1 Introducción a los paquetes dplyr y tidyr

Los paquetes dplyr y tidyr constituyen componentes fundamentales del ecosistema tidyverse, diseñados específicamente para la manipulación y transformación eficiente de datos en R. Estos paquetes implementan una filosofía de programación que prioriza la claridad y consistencia en el código, facilitando el desarrollo de análisis estadísticos reproducibles (Wickham & Grolemund, 2017).

El paquete dplyr se especializa en la manipulación de datos tabulares, proporcionando un conjunto coherente de verbos (funciones) que corresponden a las operaciones más comunes en el análisis de datos. Estas operaciones incluyen el filtrado de observaciones, la selección de variables, la creación de nuevas variables y la agregación de datos. Por su parte, tidyr se centra en la reorganización estructural de los datos, permitiendo transformaciones entre diferentes formatos según los requerimientos específicos del análisis estadístico (Wickham et al., 2023).

11.2 Configuración del Entorno y Datos de Ejemplo

Se emplea el mismo conjunto de datos simulado del experimento agrícola utilizado en el capítulo anterior, lo que permite comparar directamente los enfoques de R base y tidyverse.

```
# Configuración inicial del entorno de análisis
library(dplyr)    # Carga del paquete para manipulación de datos
library(tidyr)    # Carga del paquete para reorganización de datos

# Creación de datos simulados para experimento agrícola
set.seed(123)     # Establecimiento de semilla para reproducibilidad

datos_cultivo <- data.frame(
    parcela = 1:20,    # Identificador único de cada parcela
    tratamiento = rep(c("Control", "Fertilizante A",
                        "Fertilizante B", "Fertilizante C"), each = 5),
    bloque = rep(1:5, times = 4),    # Estructura de bloques
     # Variable respuesta 1
    altura_cm = round(rnorm(20, mean = 65, sd = 10), 1),
     # Variable respuesta 2
    peso_gr = round(rnorm(20, mean = 120, sd = 25), 1),
    daño_plaga = sample(c("Alto", "Medio", "Bajo"), 20, replace = TRUE),
    fecha_siembra = as.Date("2024-01-01") +
```

```
      sample(1:10, 20, replace = TRUE)
)

# Introducción de valores faltantes para ejemplos didácticos
datos_cultivo$altura_cm[c(3, 15)] <- NA
datos_cultivo$peso_gr[c(7, 18)] <- NA
```

El conjunto de datos simulado representa un experimento agrícola con un diseño de bloques completamente aleatorizado. Los datos incluyen mediciones de altura y peso de plantas bajo diferentes tratamientos de fertilización, organizados en bloques para controlar la variabilidad ambiental. La estructura del experimento sigue los principios fundamentales del diseño experimental descritos por Montgomery et al. (2012), donde el control local mediante bloques permite una estimación más precisa de los efectos de los tratamientos.

Las variables incluidas en el conjunto de datos son:

1. parcela: Identificador único de cada unidad experimental
2. tratamiento: Factor experimental con cuatro niveles (Control y tres tipos de fertilizantes)
3. bloque: Factor de control local con cinco niveles
4. altura_cm: Variable respuesta que mide el crecimiento vertical de las plantas
5. peso_gr: Variable respuesta que mide la biomasa de las plantas
6. daño_plaga: Evaluación categórica del daño por plagas
7. fecha_siembra: Registro temporal de la implementación del experimento

La inclusión deliberada de valores faltantes en las variables de respuesta (altura_cm y peso_gr) permite ilustrar técnicas comunes de manejo de datos incompletos, una situación frecuente en experimentos agrícolas (Field, 2013).

11.3 Operaciones básicas con dplyr

11.3.1 Filtrado de datos con filter()

La función `filter()` permite seleccionar subconjuntos de filas en un data frame o tibble, basándose en una o más condiciones lógicas. Esta función es esencial para enfocar el análisis en observaciones específicas que cumplen con criterios predefinidos (Wickham & Grolemund, 2017).

```
# Sintaxis general de la función filter()
filter(.data, ...)
```

Donde:

1. `.data`: Especifica el data frame o tibble sobre el cual se aplicará el filtrado.

2. ...: Representa una o más expresiones lógicas que deben evaluarse como `TRUE` para que una fila sea seleccionada.

Ejemplos:

```
# Ejemplo 1: Filtrar parcelas con tratamiento "Control"
datos_control <- filter(datos_cultivo, tratamiento == "Control")
head(datos_control) # Visualizar las primeras filas del resultado
```

```
  parcela tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
1       1     Control      1      59.4    93.3      Medio    2024-01-10
2       2     Control      2      62.7   114.6      Medio    2024-01-08
3       3     Control      3        NA    94.3       Bajo    2024-01-04
4       4     Control      4      65.7   101.8      Medio    2024-01-09
5       5     Control      5      66.3   104.4      Medio    2024-01-10
```

En este ejemplo, se crea un nuevo data frame llamado `datos_control` que contiene únicamente las filas donde la columna `tratamiento` es igual a "Control". La función `head()` se utiliza para mostrar las primeras filas del resultado, permitiendo una rápida verificación del filtrado.

```
# Ejemplo 2: Filtrar parcelas con altura mayor a 65 cm y
           # tratamiento distinto de "Control"
datos_altos <- filter(datos_cultivo,
                      altura_cm > 65, tratamiento != "Control")
head(datos_altos) # Visualizar las primeras filas del resultado
```

```
  parcela    tratamiento bloque altura_cm peso_gr daño_plaga fecha_siembra
1       6 Fertilizante A      1      82.2    77.8       Bajo    2024-01-04
2       7 Fertilizante A      2      69.6      NA       Bajo    2024-01-08
3      11 Fertilizante B      1      77.2   130.7       Alto    2024-01-11
4      12 Fertilizante B      2      68.6   112.6      Medio    2024-01-06
5      13 Fertilizante B      3      69.0   142.4       Alto    2024-01-06
6      14 Fertilizante B      4      66.1   142.0       Alto    2024-01-09
```

En este caso, se combinan dos condiciones lógicas: la altura debe ser mayor a 65 cm y el tratamiento no debe ser "Control". El resultado es un data frame que contiene solo las parcelas que cumplen ambos criterios. Es importante notar que, a diferencia de R base, no es necesario repetir el nombre del data frame en cada condición, lo que simplifica la sintaxis y mejora la legibilidad del código (Wickham et al., 2023).

11.3.2 Selección de columnas con select()

La función `select()` permite extraer un subconjunto de columnas de un data frame o tibble. Esta función es útil para simplificar el análisis, enfocándose únicamente en las variables relevantes para la investigación (Wickham & Grolemund, 2017).

```r
# Sintaxis general de la función select()
select(.data, ...)
```

Donde:

1. `.data`: Especifica el data frame o tibble de entrada.
2. `...`: Representa los nombres de las columnas a seleccionar, o funciones auxiliares que permiten patrones de selección más complejos.

Ejemplos:

```r
# Ejemplo 1: Seleccionar las columnas altura_cm y peso_gr
datos_mediciones <- select(datos_cultivo, altura_cm, peso_gr)
head(datos_mediciones) # Visualizar las primeras filas del resultado
```

```
  altura_cm peso_gr
1      59.4    93.3
2      62.7   114.6
3        NA    94.3
4      65.7   101.8
5      66.3   104.4
6      82.2    77.8
```

En este ejemplo, se crea un nuevo data frame llamado `datos_mediciones` que contiene únicamente las columnas `altura_cm` y `peso_gr` del data frame original.

```r
# Ejemplo 2: Excluir la columna fecha_siembra
datos_sin_fecha <- select(datos_cultivo, -fecha_siembra)
head(datos_sin_fecha) # Visualizar las primeras filas del resultado
```

```
  parcela    tratamiento bloque altura_cm peso_gr daño_plaga
1       1        Control      1      59.4    93.3      Medio
2       2        Control      2      62.7   114.6      Medio
3       3        Control      3        NA    94.3       Bajo
4       4        Control      4      65.7   101.8      Medio
5       5        Control      5      66.3   104.4      Medio
6       6 Fertilizante A      1      82.2    77.8       Bajo
```

En este caso, se utiliza el operador `-` para excluir la columna `fecha_siembra` del resultado. El nuevo data frame `datos_sin_fecha` contiene todas las columnas del original, excepto la columna excluida.

```r
# Ejemplo 3: Seleccionar columnas que terminan en "cm" o "gr"
datos_numericos <- select(datos_cultivo, ends_with("cm"), ends_with("gr"))
head(datos_numericos) # Visualizar las primeras filas del resultado
```

```
  altura_cm peso_gr
1      59.4    93.3
2      62.7   114.6
3        NA    94.3
4      65.7   101.8
5      66.3   104.4
6      82.2    77.8
```

Este ejemplo demuestra el uso de la función auxiliar `ends_with()` para seleccionar columnas cuyos nombres terminan con "cm" o "gr". El resultado es un data frame que contiene únicamente las columnas que cumplen con este patrón. El uso de funciones auxiliares permite seleccionar columnas de manera flexible, lo que resulta útil en bases de datos extensas (Wickham et al., 2023).

11.3.3 Creación y transformación de variables con mutate()

La función `mutate()` permite crear nuevas columnas o modificar las existentes en un data frame o tibble. Esta función es fundamental para la ingeniería de variables, que consiste en transformar los datos originales para generar nuevas variables que capturen información relevante para el análisis (Wickham & Grolemund, 2017).

```r
# Sintaxis general de la función mutate()
mutate(.data, ...)
```

1. `.data`: Especifica el data frame o tibble de entrada.
2. `...`: Representa una o más expresiones que definen las nuevas columnas o transformaciones.

Ejemplos:

```r
# Ejemplo 1: Crear una nueva variable: índice de crecimiento
datos_cultivo <- mutate(datos_cultivo,
                        indice_crecimiento =
                          altura_cm / peso_gr)
```

En este ejemplo, se crea una nueva columna llamada `indice_crecimiento` que se calcula como la razón entre la altura y el peso de cada planta. La nueva columna se añade al data frame original `datos_cultivo`.

```r
# Ejemplo 2: Crear varias variables nuevas
datos_cultivo <- mutate(datos_cultivo,
                        altura_m = altura_cm / 100,
                        peso_kg = peso_gr / 1000,
                        categoria_altura = ifelse(
                          altura_cm > 65, "Alto", "Bajo"))
```

En este caso, se crean tres nuevas columnas: `altura_m` (altura en metros), `peso_kg` (peso en kilogramos) y `categoria_altura` (categoría de altura basada en un umbral). La función `ifelse(condición, valor_si_verdadero, valor_si_falso)` permite crear variables categóricas a partir de condiciones lógicas, lo que es común en la estadística clásica para definir grupos o categorías (Wickham & Grolemund, 2017).

11.3.4 Agrupamiento y resumen con group_by() y summarize()

Las funciones `group_by()` y `summarize()` son herramientas poderosas para el análisis exploratorio de datos y la generación de estadísticas descriptivas por grupos. La función `group_by()` permite dividir un data frame en grupos basados en los valores de una o más variables, mientras que `summarize()` calcula estadísticas resumen para cada grupo (Wickham & Grolemund, 2017).

```
# Sintaxis general de las funciones group_by() y summarize()
group_by(.data, ...)
summarize(.data, ...)
```

Donde:

1. `.data`: Especifica el data frame o tibble de entrada.
2. `...`: Representa las variables de agrupamiento (en `group_by()`) o las expresiones de resumen (en `summarize()`).

Ejemplos:

```
# Ejemplo: Agrupar por tratamiento y calcular estadísticas descriptivas
resumen_tratamiento <- datos_cultivo %>%
    group_by(tratamiento) %>%
    summarize(
        media_altura = mean(altura_cm, na.rm = TRUE),
        sd_altura = sd(altura_cm, na.rm = TRUE),
        n = n()
    )
resumen_tratamiento
```

```
# A tibble: 4 x 4
  tratamiento    media_altura sd_altura     n
  <chr>                 <dbl>     <dbl> <int>
1 Control                63.5      3.17     5
2 Fertilizante A         64.5     11.7      5
3 Fertilizante B         70.2      4.82     5
4 Fertilizante C         66.1     14.1      5
```

En este ejemplo, se utiliza el operador pipe (`%>%`) para encadenar las funciones `group_by()` y `summarize()`. Primero, se agrupan los datos por la variable `tratamiento`. Luego, para cada grupo, se calculan las siguientes estadísticas:

1. `media_altura = mean(altura_cm, na.rm = TRUE)`: Calcula la media de la altura, excluyendo los valores faltantes (`NA`). El argumento `na.rm = TRUE` es crucial para evitar que los valores faltantes afecten el cálculo de la media.
2. `sd_altura = sd(altura_cm, na.rm = TRUE)`: Calcula la desviación estándar de la altura, también excluyendo los valores faltantes.
3. `n = n()`: Cuenta el número de observaciones en cada grupo. La función `n()` es una función especial de dplyr que cuenta el tamaño de cada grupo.

El resultado es un nuevo data frame llamado `resumen_tratamiento` que contiene las estadísticas descriptivas para cada tratamiento. Estas funciones son esenciales para obtener resúmenes estadísticos por grupo, como promedios por tratamiento en un experimento clásico (Wickham et al., 2023).

11.3.5 Ordenamiento de datos con arrange()

La función `arrange()` permite ordenar las filas de un data frame o tibble según los valores de una o más variables. El ordenamiento es útil para identificar valores extremos, preparar tablas para reportes o facilitar la visualización de datos (Wickham & Grolemund, 2017).

```
# Sintaxis general de la función arrange()
arrange(.data, ...)
```

Donde:

1. `.data`: Especifica el data frame o tibble de entrada.
2. `...`: Representa las variables por las que se desea ordenar. Se puede utilizar la función `desc()` para ordenar en orden descendente.

Ejemplos

```
# Ejemplo 1: Ordenar por altura de menor a mayor
datos_ordenados <- arrange(datos_cultivo, altura_cm)
```

En este ejemplo, se crea un nuevo data frame llamado `datos_ordenados` que contiene las mismas filas que `datos_cultivo`, pero ordenadas de menor a mayor según los valores de la columna `altura_cm`.

```
# Ejemplo 2: Ordenar por tratamiento y peso descendente
datos_ordenados_multi <- arrange(datos_cultivo,
                                 tratamiento,
                                 desc(peso_gr))
```

En este caso, se ordenan los datos por dos variables: primero por `tratamiento` (en orden ascendente por defecto) y luego por `peso_gr` (en orden descendente, gracias a la función `desc()`). El resultado es un data frame donde las filas están ordenadas alfabéticamente por tratamiento, y dentro de cada tratamiento, las filas están ordenadas de mayor a menor según el peso.

11.4 Introducción a los pipes (%>%)

El operador pipe (`%>%`), introducido por el paquete `magrittr` y adoptado como parte fundamental del `tidyverse`, representa una innovación significativa en la sintaxis de R. Este operador permite construir secuencias de operaciones de manera clara y lógica, siguiendo un flujo natural de procesamiento de datos. El pipe toma el resultado de una expresión a su izquierda y lo pasa como primer argumento a la función a su derecha (Wickham & Grolemund, 2017).

La sintaxis básica del operador pipe es:

```
# Estructura usando pipes
datos %>% funcion()

# Equivalente a la siguiente estructura anidada
funcion(datos)
```

11.4.1 Ventajas del uso de pipes

El uso de pipes ofrece múltiples ventajas en el análisis estadístico (Wickham et al., 2023):

1. **Legibilidad mejorada**: Las operaciones se leen de izquierda a derecha y de arriba hacia abajo, siguiendo el orden natural de lectura. Esto facilita la comprensión del flujo de trabajo y reduce la probabilidad de errores.
2. **Reducción de objetos intermedios**: No es necesario crear variables temporales para almacenar resultados intermedios. Esto simplifica el código y reduce el riesgo de errores asociados con la gestión de múltiples objetos.
3. **Facilidad de depuración**: Cada paso puede ser comentado o modificado independientemente. Esto facilita la identificación y corrección de errores en el código.
4. **Claridad en la secuencia de operaciones**: El flujo de trabajo se hace explícito y fácil de seguir. Esto mejora la mantenibilidad del código y facilita la colaboración entre analistas.

11.4.2 Ejemplo práctico

Para ilustrar las ventajas del uso de pipes, consideremos el siguiente ejemplo, donde se calcula la media de la altura por tratamiento, excluyendo los valores faltantes:

Sin pipe (anidado): En la sintaxis tradicional, las funciones deben anidarse, lo que puede dificultar la lectura:

```
# Calcular la media de altura por tratamiento, excluyendo valores NA
resumen_tratamiento <- summarize(
  group_by(
    filter(datos_cultivo, !is.na(altura_cm)),
    tratamiento
```

```
  ),
  media_altura = mean(altura_cm)
)
```

En este ejemplo, primero se filtran las filas sin valores faltantes en `altura_cm`, luego se agrupan por `tratamiento` y finalmente se calcula la media de altura para cada grupo. La anidación de funciones dificulta la lectura y comprensión del código.

Con pipe (más legible): El mismo análisis, usando pipes, resulta más claro y fácil de seguir:

```
# Calcular la media de altura por tratamiento, excluyendo valores NA
resumen_tratamiento <- datos_cultivo %>%
    # 1. Eliminar filas con NA en altura_cm
    filter(!is.na(altura_cm)) %>%
    # 2. Agrupar los datos por tratamiento
    group_by(tratamiento) %>%
    # 3. Calcular la media de altura por grupo
    summarize(media_altura = mean(altura_cm))
```

En este ejemplo, el código se lee de arriba hacia abajo, siguiendo el orden lógico de las operaciones:

1. En la primera línea, se eliminan las filas donde la altura es `NA`.
2. En la segunda línea, se agrupan los datos por el tipo de tratamiento.
3. En la tercera línea, se calcula la media de la altura para cada tratamiento.

Cada paso es explícito y se puede leer de arriba hacia abajo, lo que facilita la comprensión y depuración del análisis (Wickham & Grolemund, 2017). El uso de pipes mejora significativamente la legibilidad y mantenibilidad del código, facilitando la colaboración y reduciendo la probabilidad de errores.

11.5 Transformaciones de datos con tidyr

El paquete `tidyr` es una herramienta fundamental para la reorganización y transformación de datos en R, permitiendo adaptar la estructura de los conjuntos de datos a los requerimientos de los métodos estadísticos clásicos. Estas transformaciones son esenciales para preparar los datos antes de aplicar técnicas como ANOVA, regresión o análisis descriptivos, ya que muchos procedimientos requieren que los datos estén en un formato específico (Wickham & Grolemund, 2017).

11.5.1 Transformación de formato ancho a largo con pivot_longer()

La función `pivot_longer()` convierte varias columnas de un data frame en pares de nombre-valor, generando un formato largo. Este formato es especialmente útil en análisis estadísticos donde cada observación debe ocupar una fila y las variables medidas se representan en una columna adicional, como en el caso de ANOVA de medidas repetidas (Wickham & Grolemund, 2017).

La sintaxis principal es:

```
# Sintaxis principal de la funcion pivot_longer ()
pivot_longer(
  data,              # Data frame o tibble de entrada
  cols,              # Columnas a transformar
  # Nombre de la nueva columna para las columnas originales
  names_to = "name",
  # Nombre de la nueva columna para los valores originales
  values_to = "value"
)
```

Para ilustrar el uso de `pivot_longer()`, consideremos un ejemplo simplificado:

```
# Crear un data frame de ejemplo
datos_ancho <- data.frame(
  parcela = 1:3,
  altura_2023 = c(150, 160, 155),
  peso_2023 = c(80, 85, 82),
  altura_2024 = c(165, 170, 168),
  peso_2024 = c(88, 90, 89)
)

datos_ancho
```

```
  parcela altura_2023 peso_2023 altura_2024 peso_2024
1       1         150        80         165        88
2       2         160        85         170        90
3       3         155        82         168        89
```

Este data frame representa mediciones de altura y peso de tres parcelas en dos años diferentes. Para transformar este data frame a formato largo, podemos usar `pivot_longer()` de la siguiente manera:

```
# Transformar a formato largo
datos_largo <- datos_ancho %>%
  pivot_longer(
    cols = c(altura_2023, peso_2023, altura_2024, peso_2024),
    names_to = "variable",
    values_to = "valor"
```

```
  )

datos_largo
```

```
# A tibble: 12 x 3
   parcela variable    valor
     <int> <chr>       <dbl>
 1       1 altura_2023   150
 2       1 peso_2023      80
 3       1 altura_2024   165
 4       1 peso_2024      88
 5       2 altura_2023   160
 6       2 peso_2023      85
 7       2 altura_2024   170
 8       2 peso_2024      90
 9       3 altura_2023   155
10       3 peso_2023      82
11       3 altura_2024   168
12       3 peso_2024      89
```

En este ejemplo:

1. `cols = c(altura_2023, peso_2023, altura_2024, peso_2024)`: Especifica las columnas que se van a transformar.
2. `names_to = "variable"`: Indica que los nombres de las columnas originales se almacenarán en una nueva columna llamada "variable".
3. `values_to = "valor"`: Indica que los valores de las columnas originales se almacenarán en una nueva columna llamada "valor".

El resultado es un data frame en formato largo, donde cada fila representa una medición de altura o peso para una parcela en un año específico. Este formato es ideal para realizar análisis estadísticos que requieren que cada observación ocupe una fila, como ANOVA de medidas repetidas o modelos mixtos (Kutner et al., 2005).

11.5.2 Transformación de formato largo a ancho con pivot_wider()

La función `pivot_wider()` realiza la operación inversa a `pivot_longer()`, transformando un data frame de formato largo a formato ancho. Esta función es útil cuando se necesita organizar los datos de manera que diferentes valores de una variable se conviertan en columnas separadas, facilitando la comparación entre grupos o condiciones (Wickham & Grolemund, 2017).

La sintaxis principal es:

```r
# Sintaxis principal de la funcion pivot_wider ()
pivot_wider(
  # Data frame o tibble de entrada
  data,
  # Columna cuyos valores se usarán como nombres de las nuevas columnas
  names_from = ,
  # Columna cuyos valores se usarán para llenar las nuevas columnas
  values_from =
)
```

Para ilustrar el uso de `pivot_wider()`, consideremos el data frame `datos_largo` creado en la sección anterior. Para transformar este data frame de nuevo a formato ancho, podemos usar `pivot_wider()` de la siguiente manera:

```r
# Transformar a formato ancho
datos_ancho <- datos_largo %>%
  pivot_wider(
    names_from = variable,
    values_from = valor
  )

datos_ancho
```

```
# A tibble: 3 x 5
  parcela altura_2023 peso_2023 altura_2024 peso_2024
    <int>       <dbl>     <dbl>       <dbl>     <dbl>
1       1         150        80         165        88
2       2         160        85         170        90
3       3         155        82         168        89
```

En este ejemplo:

1. `names_from = variable`: Especifica que los valores de la columna `variable` (altura_2023, peso_2023, altura_2024, peso_2024) se utilizarán como nombres de las nuevas columnas.
2. `values_from = valor`: Especifica que los valores de la columna `valor` se utilizarán para llenar las nuevas columnas.

El resultado es un data frame en formato ancho, donde cada fila representa una parcela y cada columna representa una medición de altura o peso en un año específico. Este formato facilita la comparación directa de las mediciones entre años para cada parcela.

11.5.3 Separación y Unión de Columnas con separate() y unite()

Las funciones `separate()` y `unite()` permiten manipular variables compuestas, dividiendo una columna en varias o combinando varias columnas en una sola. Estas

funciones son útiles para limpiar y estructurar datos que contienen información combinada en una sola columna (Wickham & Grolemund, 2017).

La función `separate()` divide una columna en varias, utilizando un carácter separador. Su sintaxis principal es:

```
# Sintaxis de la funcion separate ()
separate(
  data,    # Data frame o tibble de entrada
  col,     # Columna a dividir
  into,    # Vector con los nombres de las nuevas columnas
  sep      # Carácter separador
)
```

Por ejemplo, considérese el siguiente subconjunto:

```
# Crear el dataframe para el ejemplo
mini_datos_comp <- data.frame(
  parcela_bloque = c("1-1", "2-2", "3-3"),
  altura_cm = c(70, 65, 60)
)

mini_datos_comp
```

```
  parcela_bloque altura_cm
1            1-1        70
2            2-2        65
3            3-3        60
```

Para separar la columna `parcela_bloque` en dos columnas llamadas `parcela` y `bloque`, se utiliza:

```
mini_separado <- separate(
  data = mini_datos_comp,
  col = parcela_bloque,    # Columna a dividir
  into = c("parcela", "bloque"),# Nombres de las nuevas columnas
  sep = "-"                # Carácter separador
)

mini_separado
```

```
  parcela bloque altura_cm
1       1      1        70
2       2      2        65
3       3      3        60
```

El argumento `col` indica la columna a dividir, `into` define los nombres de las nuevas columnas y `sep` especifica el carácter separador (Wickham & Grolemund, 2017).

La función `unite()` combina dos o más columnas en una sola, utilizando un carácter separador. Su sintaxis principal es:

```
# Sintaxis de la funcion unite ()
unite(
  data,    # Data frame o tibble de entrada
  col,     # Nombre de la nueva columna
  ...,     # Columnas a unir
  sep      # Carácter separador
)
```

Por ejemplo, para volver a unir las columnas `parcela` y `bloque` en una sola columna `parcela_bloque`:

```
mini_unido <- unite(
  data = mini_separado,
  col = "parcela_bloque", # Nombre de la nueva columna
  parcela, bloque,        # Columnas a unir
  sep = "-"                # Carácter separador
)

mini_unido
```

```
  parcela_bloque altura_cm
1            1-1        70
2            2-2        65
3            3-3        60
```

El argumento `col` define el nombre de la nueva columna resultante, los siguientes argumentos son las columnas a unir y `sep` indica el carácter separador (Wickham & Grolemund, 2017).

11.6 Comparación entre la manipulación de datos con R base y tidyverse

La manipulación de datos constituye una etapa esencial en el análisis estadístico clásico, ya que permite preparar, transformar y explorar la información antes de aplicar técnicas inferenciales o modelos predictivos. En el entorno R, existen dos enfoques principales para realizar estas tareas: el uso de funciones base y el empleo de paquetes del tidyverse, como dplyr y tidyr. A continuación, se presenta una comparación estructurada de ambos enfoques, considerando aspectos clave como sintaxis, legibilidad, flexibilidad y reproducibilidad (Wickham & Grolemund, 2017).

Aspecto	R base	tidyverse (dplyr/tidyr)
Sintaxis	Uso de corchetes, funciones como `subset()`, `apply()`, y anidación.	Uso de funciones verbales (`filter()`, `select()`, `mutate()`, etc.) y pipes `%>%`.
Legibilidad	El código puede ser difícil de leer, especialmente con operaciones anidadas.	El flujo de trabajo es secuencial y fácil de seguir, cada paso en una línea.
Creación de variables	Se usa `$` o `transform()`.	Se usa `mutate()`, que permite crear o modificar variables de forma clara.
Filtrado de filas	Se usan corchetes o `subset()`.	Se usa `filter()`, con sintaxis más intuitiva y sin necesidad de repetir el nombre del data frame.
Selección de columnas	Se usan corchetes o `select()`.	Se usa `select()`, con funciones auxiliares como `starts_with()`, `ends_with()`.
Agrupamiento y resumen	Se usan `tapply()`, `aggregate()`, o bucles.	Se usan `group_by()` y `summarize()`, facilitando el cálculo de estadísticas por grupo.
Transformación de formato	Se usan funciones como `reshape()`, `melt()`, `cast()`.	Se usan `pivot_longer()` y `pivot_wider()`, con sintaxis más clara y moderna.
Manejo de variables compuestas	Se requiere manipulación manual con funciones como `strsplit()`.	Se usan `separate()` y `unite()`, que simplifican la división y combinación de columnas.
Reproducibilidad	El código puede ser menos reproducible y más propenso a errores.	El uso de pipes y funciones verbales mejora la reproducibilidad y la claridad del análisis.
Curva de aprendizaje	Familiar para usuarios con experiencia previa en R, pero puede ser menos intuitivo para principiantes.	Más accesible para principiantes, especialmente por la coherencia y claridad de la sintaxis.

En síntesis, el enfoque tidyverse ofrece ventajas notables en términos de claridad, reproducibilidad y facilidad de uso, especialmente en flujos de trabajo complejos o colaborativos. Sin embargo, el conocimiento de las funciones base de R sigue siendo valioso, ya que permite comprender el funcionamiento interno del lenguaje y resolver tareas específicas de manera eficiente (Wickham & Grolemund, 2017).

Capítulo IV

Visualización de datos

12 Introducción a la visualización de datos

La visualización de datos se define como el proceso de representar información cuantitativa y cualitativa mediante gráficos, diagramas y otras formas visuales. Su objetivo principal es facilitar la comprensión, el análisis y la comunicación de los datos, permitiendo identificar patrones, tendencias, relaciones y anomalías que podrían pasar desapercibidas en tablas numéricas o descripciones textuales. En el contexto del análisis estadístico, la visualización es una herramienta esencial tanto en la fase exploratoria como en la presentación de resultados, ya que ayuda a validar supuestos, comunicar hallazgos y respaldar la toma de decisiones informadas (Wickham, 2016).

La importancia de la visualización radica en su capacidad para transformar datos complejos en representaciones accesibles y comprensibles, promoviendo la transparencia y la reproducibilidad en la investigación científica. Además, los gráficos permiten detectar errores en los datos, identificar valores atípicos y comprender la distribución de las variables antes de aplicar técnicas estadísticas formales (Tufte, 2001).

12.1 Historia y evolución de la visualización en estadística

La visualización de datos tiene una larga tradición en la historia de la estadística. Sus orígenes se remontan al siglo XVIII, cuando se comenzaron a utilizar gráficos para representar información demográfica y económica. Uno de los hitos más importantes fue la invención del gráfico de barras por William Playfair en 1786, quien también introdujo el gráfico de líneas y el gráfico circular. Posteriormente, Florence Nightingale empleó diagramas de área para comunicar la mortalidad en hospitales militares, demostrando el poder de los gráficos para influir en la opinión pública y en la toma de decisiones políticas (Friendly, 2008).

A lo largo del siglo XX, la visualización se consolidó como una disciplina fundamental en la estadística, especialmente con el desarrollo de la computación y el software estadístico, que permitieron la creación de gráficos más complejos y personalizados. En la actualidad, la visualización de datos es un componente central en el análisis exploratorio de datos (EDA) y en la comunicación científica, siendo reconocida como una herramienta indispensable para el trabajo estadístico (Wickham, 2016).

12.2 Principios básicos de la visualización efectiva

La visualización efectiva de datos es un componente esencial para asegurar que la información transmitida sea comprensible, precisa y útil en la toma de decisiones estadísticas. Para lograr este objetivo, es fundamental considerar tres principios clave: claridad, precisión y eficiencia. Estos principios han sido ampliamente discutidos en la

literatura especializada, destacando su relevancia en la comunicación gráfica de datos (Tufte, 2001; Cleveland, 1993).

12.2.1 Claridad

La claridad en la visualización implica que el gráfico sea comprensible y transmita el mensaje principal de manera directa, sin ambigüedades ni elementos distractores. Para lograr claridad, se deben considerar los siguientes aspectos (Tufte, 2001):

1. Eliminar elementos decorativos innecesarios, como fondos llamativos, sombras o efectos tridimensionales que no aportan información relevante.
2. Utilizar títulos descriptivos y etiquetas claras en los ejes, de modo que el lector comprenda de inmediato qué variables se están representando.
3. Incluir leyendas explicativas cuando se utilicen colores, símbolos o líneas para diferenciar grupos o categorías.
4. Mantener un diseño limpio y ordenado, evitando la sobrecarga visual y el uso excesivo de colores o tipografías.
5. Presentar la información de manera secuencial y lógica, facilitando la interpretación del gráfico desde el primer vistazo.

12.2.2 Precisión

La precisión se refiere a la representación fiel y exacta de los datos, evitando distorsiones que puedan inducir a interpretaciones erróneas. Para asegurar la precisión en los gráficos, se recomienda (Cleveland, 1993):

1. Utilizar escalas proporcionales y adecuadas al rango de los datos, evitando truncar ejes o manipular escalas que alteren la percepción de las diferencias o relaciones.
2. Representar todos los datos relevantes, sin omitir valores atípicos o subconjuntos importantes que puedan influir en la interpretación.
3. Seleccionar el tipo de gráfico adecuado para el tipo de variable y el objetivo del análisis, por ejemplo, no usar gráficos de barras para variables continuas.
4. Evitar la exageración visual de diferencias mediante el uso de áreas, volúmenes o efectos visuales que no correspondan a la magnitud real de los datos.
5. Revisar cuidadosamente los datos y la codificación del gráfico para prevenir errores de transcripción o interpretación.

12.2.3 Eficiencia

La eficiencia en la visualización implica transmitir la mayor cantidad de información relevante con el menor esfuerzo cognitivo posible para el usuario. Para lograr eficiencia, se deben seguir estas recomendacionesndefined(Tufte, 2001; Cleveland, 1993):

1. Resumir la información de manera que el gráfico muestre los aspectos más importantes sin saturar de detalles innecesarios.
2. Utilizar gráficos sintéticos, como diagramas de caja o gráficos de dispersión, que permiten visualizar múltiples características de los datos en una sola imagen.
3. Priorizar la información relevante para el objetivo del análisis, evitando la inclusión de variables o elementos que no aportan al mensaje principal.
4. Facilitar la comparación entre grupos o categorías mediante el uso de colores, formas o posiciones consistentes y fácilmente distinguibles.
5. Optimizar el tamaño y la resolución del gráfico para que sea legible tanto en pantalla como en impresiones.

12.2.4 Errores comunes a evitar en la visualización de datos

Existen errores frecuentes que pueden comprometer la efectividad de una visualización. Entre los más relevantes se encuentran (Tufte, 2001; Cleveland, 1993):

1. Uso excesivo de colores, degradados o efectos visuales que dificultan la interpretación y distraen del mensaje principal.
2. Omitir etiquetas, títulos o leyendas, lo que genera confusión sobre el significado de los elementos representados.
3. Elegir un tipo de gráfico inadecuado para el tipo de datos, como utilizar gráficos circulares para comparar muchas categorías o gráficos de líneas para variables categóricas.
4. Manipular escalas de los ejes para exagerar o minimizar diferencias, lo que puede inducir a conclusiones erróneas.
5. Presentar demasiada información en un solo gráfico, lo que sobrecarga al usuario y dificulta la extracción de conclusiones claras.

12.2.5 Recomendaciones para una visualización efectiva

Para garantizar la integridad y la transparencia en la presentación de los datos, se recomienda (Tufte, 2001; Cleveland, 1993):

1. Seleccionar el tipo de gráfico más adecuado según el objetivo del análisis y la naturaleza de las variables.
2. Mantener un diseño simple, claro y directo, priorizando la comprensión del mensaje principal.

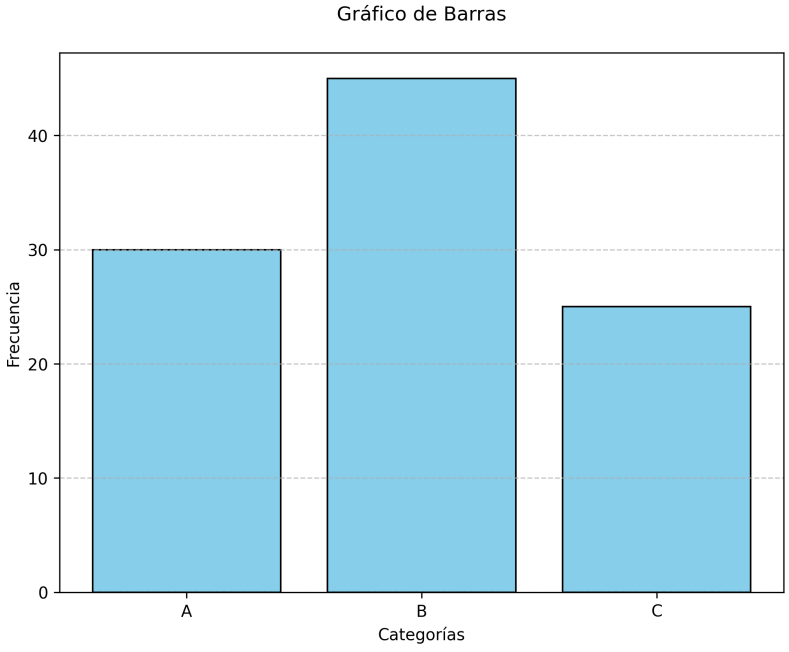
3. Revisar y validar los gráficos antes de su presentación, asegurando que representen fielmente los datos y sean interpretables por el público objetivo.
4. Utilizar recursos visuales (colores, formas, tamaños) de manera coherente y justificada, evitando la sobrecarga visual.
5. Documentar las decisiones tomadas en la construcción del gráfico, facilitando la reproducibilidad y la transparencia en el análisis.

12.3 Tipos de gráficos y su utilidad en estadística clásica

En la estadística clásica, la selección adecuada del tipo de gráfico es fundamental para explorar los datos, validar supuestos y comunicar resultados de manera efectiva. A continuación, se describen los principales tipos de gráficos utilizados, sus características y su utilidad específica en el análisis estadístico, siguiendo las recomendaciones de la literatura especializada (Venables & Ripley, 2002; Cleveland, 1993).

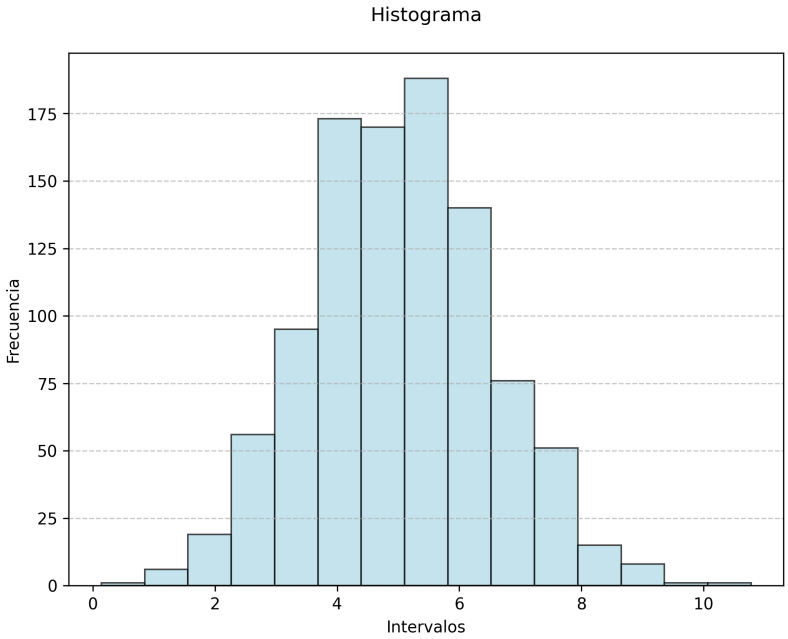
12.3.1 Gráficos de barras

Los gráficos de barras permiten comparar frecuencias o proporciones entre categorías de una variable cualitativa. Cada barra representa una categoría y su altura es proporcional a la frecuencia o porcentaje correspondiente. Este tipo de gráfico facilita la identificación de categorías dominantes o poco representadas y es especialmente útil en el análisis de variables como sexo, grupo de tratamiento o respuestas dicotómicas. Además, los gráficos de barras ayudan a detectar patrones de distribución y posibles sesgos en la recolección de datos (Cleveland, 1993).



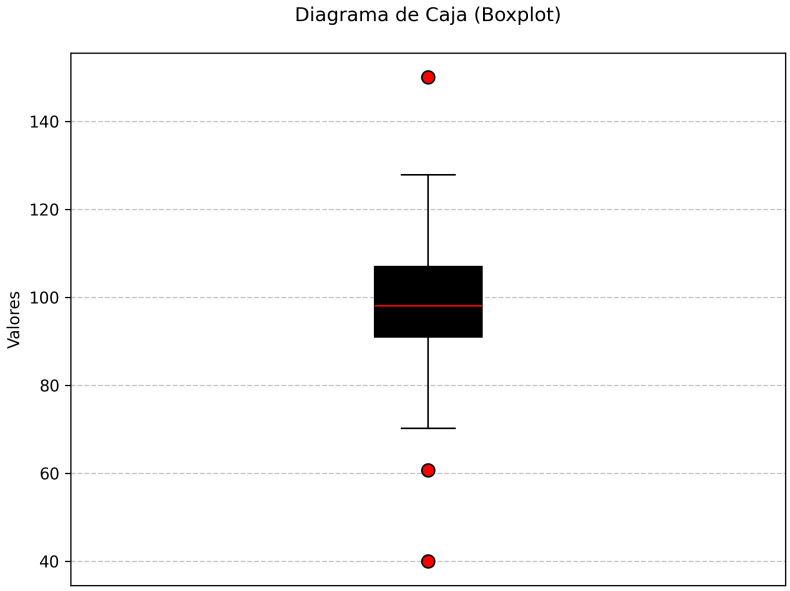
12.3.2 Histogramas

El histograma es la herramienta principal para visualizar la distribución de variables cuantitativas continuas. Agrupa los datos en intervalos y muestra la frecuencia de observaciones en cada uno. Esta representación permite identificar la forma de la distribución, detectar asimetrías, curtosis, valores atípicos y la presencia de múltiples modos. Los histogramas son esenciales para evaluar el supuesto de normalidad, requisito frecuente en pruebas como el ANOVA y la regresión lineal (Venables & Ripley, 2002).



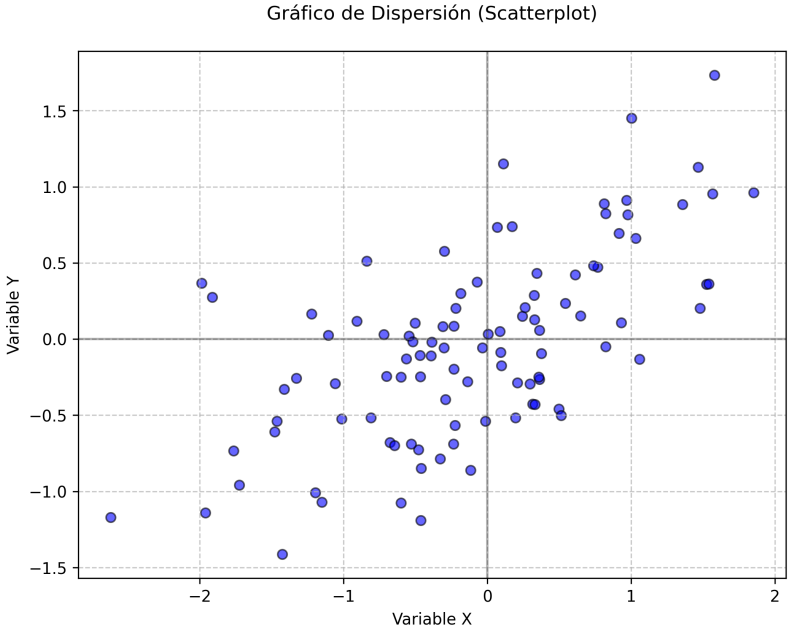
12.3.3 Diagramas de caja (boxplots)

El diagrama de caja, o boxplot, resume la distribución de una variable cuantitativa mostrando la mediana, los cuartiles y los valores extremos. Este gráfico facilita la comparación entre grupos y la identificación de valores atípicos. Además, permite evaluar la homogeneidad de la varianza, aspecto crucial en el análisis de varianza. Su interpretación sencilla y su capacidad para sintetizar información lo convierten en una herramienta indispensable en la estadística descriptiva y comparativa (Cleveland, 1993).



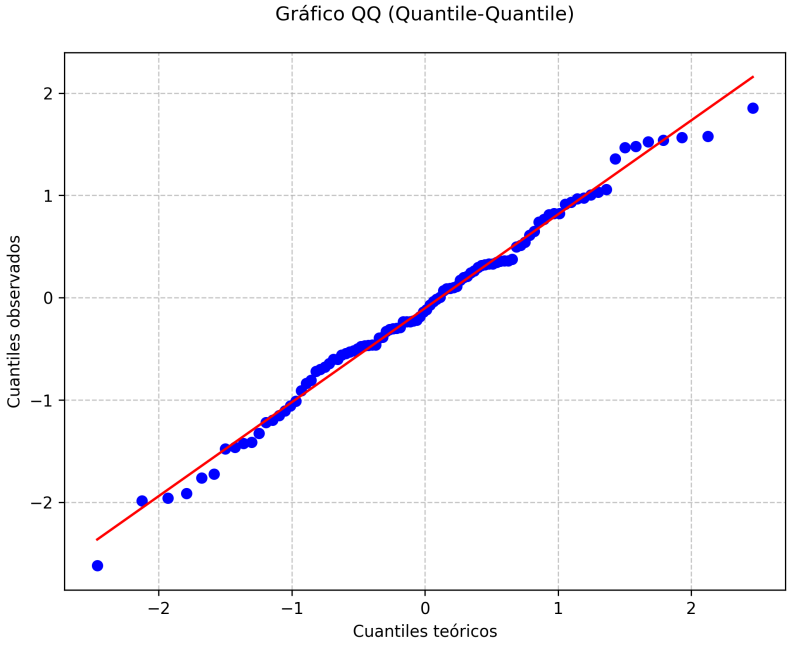
12.3.4 Gráficos de dispersión (scatterplots)

Los gráficos de dispersión se utilizan para analizar la relación entre dos variables cuantitativas. Cada punto representa una observación y su posición refleja los valores de ambas variables. Este tipo de gráfico permite identificar patrones de asociación, linealidad, presencia de valores atípicos y posibles agrupamientos. Además, es fundamental para explorar la existencia de correlaciones y para evaluar el supuesto de linealidad en modelos de regresión (Venables & Ripley, 2002).



12.3.5 Gráficos QQ (quantile-quantile)

El gráfico QQ compara la distribución de los datos observados con una distribución teórica, generalmente la normal. Si los puntos del gráfico se alinean sobre la diagonal, se puede concluir que los datos siguen la distribución teórica. Este gráfico es esencial para evaluar el supuesto de normalidad en pruebas paramétricas y para detectar desviaciones sistemáticas, colas pesadas o asimetrías en la distribución de los datos (Cleveland, 1993).



12.3.6 Gráficos de residuos

Los gráficos de residuos muestran la diferencia entre los valores observados y los valores ajustados por un modelo estadístico. Un patrón aleatorio en estos gráficos indica que el modelo es adecuado, mientras que la presencia de patrones sistemáticos sugiere problemas de especificación, heterocedasticidad o autocorrelación. Estos gráficos son fundamentales en la validación de modelos de regresión y en la toma de decisiones sobre la necesidad de transformar variables o ajustar el modelo (Venables & Ripley, 2002).

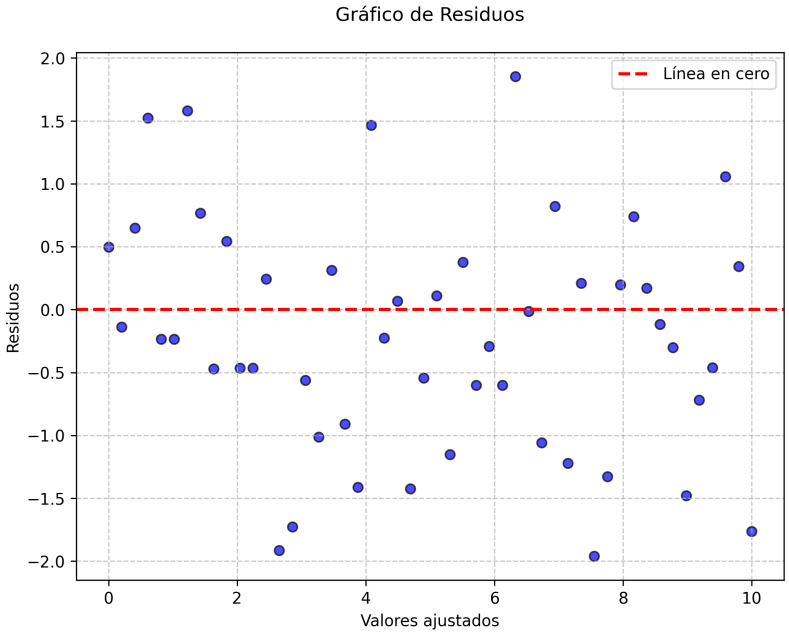


Gráfico de Residuos
Línea en cero
Residuos
Valores ajustados
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2
4
6
8
10

13 Sistema Gráfico Base de R

La representación gráfica de la información constituye un componente indispensable para la comprensión, la comunicación y la validación de resultados estadísticos. El sistema gráfico base de R ofrece un conjunto de herramientas versátiles que permiten construir visualizaciones de alta calidad siguiendo un enfoque incremental, en el cual cada elemento puede añadirse o modificarse de forma independiente (Murrell, 2018). A continuación se describe, de manera detallada y pedagógica, la arquitectura de este sistema y las funciones esenciales para el análisis exploratorio y la comprobación de supuestos en la estadística clásica.

El propósito principal de la visualización es facilitar la detección de patrones, tendencias y anomalías que resultan difíciles de advertir mediante inspección numérica (Cleveland, 1993). Además, las gráficas permiten evaluar supuestos tales como normalidad, homocedasticidad y linealidad, que son cruciales para la validez de métodos paramétricos como la regresión lineal y el ANOVA (Venables & Ripley, 2002). Bajo esta perspectiva, la elaboración de gráficos debe regirse por principios de claridad, precisión y economía visual (Tufte, 2001).

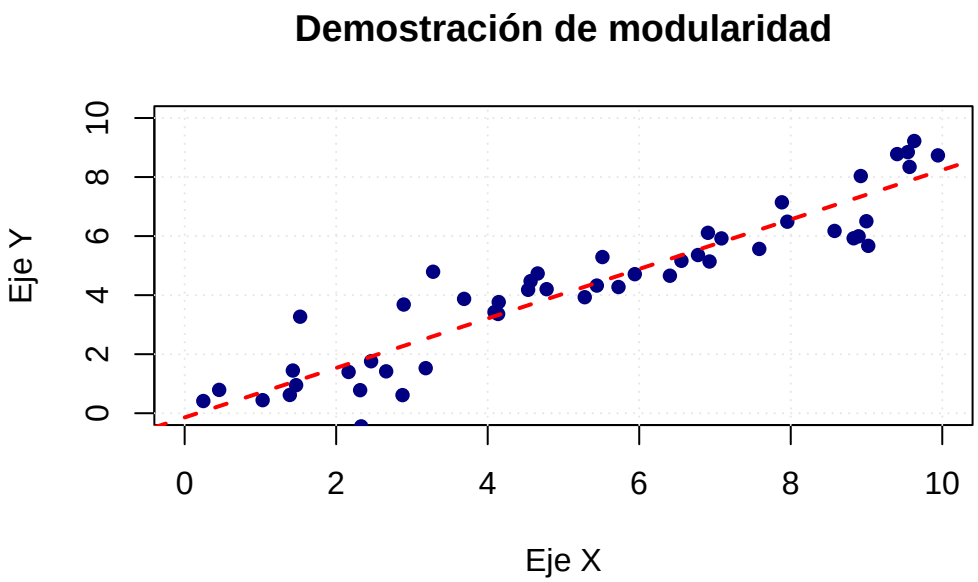
13.1 Arquitectura del sistema gráfico base

El sistema gráfico base de R se sustenta en tres principios operativos:

1. **Modularidad:** cada elemento (ejes, marcas, títulos, objetos geométricos) puede añadirse o modificarse sin rehacer el gráfico desde cero.
2. **Jerarquía:** los componentes se dibujan en capas sucesivas sobre un "lienzo" inicial.
3. **Persistencia:** las modificaciones se aplican sobre el dispositivo gráfico activo hasta que este se cierra o se restablecen los parámetros originales (Murrell, 2018).

```
# Ejemplo ilustrativo de construcción modular
plot(NULL,                          # Lienzo vacío
     xlim = c(0, 10), ylim = c(0, 10),
     xlab = "Eje X", ylab = "Eje Y",
     main = "Demostración de modularidad")

grid(col = "gray90")                # Capa 1: cuadrícula

# Capa 2: puntos de datos simulados
set.seed(123)
x <- runif(50, 0, 10)
y <- 0.8 * x + rnorm(50, 0, 1)
points(x, y, pch = 16, col = "navy")
```

```r
# Capa 3: línea de tendencia
abline(lm(y ~ x), col = "red", lwd = 2, lty = 2)
```



13.2 Funciones gráficas básicas de R

El sistema gráfico base de R constituye una de las herramientas más accesibles y versátiles para la visualización de datos en estadística clásica. Estas funciones permiten crear gráficos de manera rápida y flexible, facilitando tanto la exploración inicial de los datos como la comprobación de supuestos estadísticos fundamentales. El enfoque de R base se basa en la construcción secuencial de gráficos, donde cada elemento puede ser añadido o modificado mediante argumentos y funciones auxiliares, lo que resulta especialmente útil en el análisis exploratorio y diagnóstico (Murrell, 2018; Venables & Ripley, 2002).

Entre las funciones más utilizadas se encuentran:

1. `plot()`: función genérica para gráficos de dispersión, líneas y otros tipos de visualizaciones.
2. `hist()`: para la creación de histogramas que muestran la distribución de variables cuantitativas.
3. `boxplot()`: para diagramas de caja que resumen la dispersión y los valores atípicos.
4. `barplot()`: para gráficos de barras de frecuencias o proporciones.
5. `qqnorm()` y `qqline()`: para gráficos Q-Q que evalúan la normalidad de los datos.
6. `pairs()`: para matrices de gráficos de dispersión entre varias variables.

Estas funciones son la base para la mayoría de los análisis gráficos en estadística clásica, permitiendo una rápida inspección visual de los datos y la validación de supuestos (Venables & Ripley, 2002).

13.3 Funciones esenciales para la exploración de datos

La exploración visual de los datos es una etapa fundamental en cualquier análisis estadístico, ya que permite identificar patrones, tendencias, anomalías y posibles errores en los datos antes de aplicar modelos formales. El sistema gráfico base de R proporciona funciones versátiles y personalizables para la creación de gráficos exploratorios, facilitando la comprensión y la comunicación de los resultados (Venables & Ripley, 2002; Murrell, 2018).

13.3.1 Histogramas

El histograma es una herramienta gráfica que permite representar la distribución de una variable numérica, facilitando la identificación de asimetrías, curtosis, valores atípicos y posibles multimodalidades (Cleveland, 1993). En R, la función principal para crear histogramas es `hist()`.

Sintaxis general:

```
hist(x,
     breaks = "Sturges",
     freq = TRUE,
     col = NULL,
     border = NULL,
     main = NULL,
     xlab = NULL,
     ylab = NULL,
     ...)
```

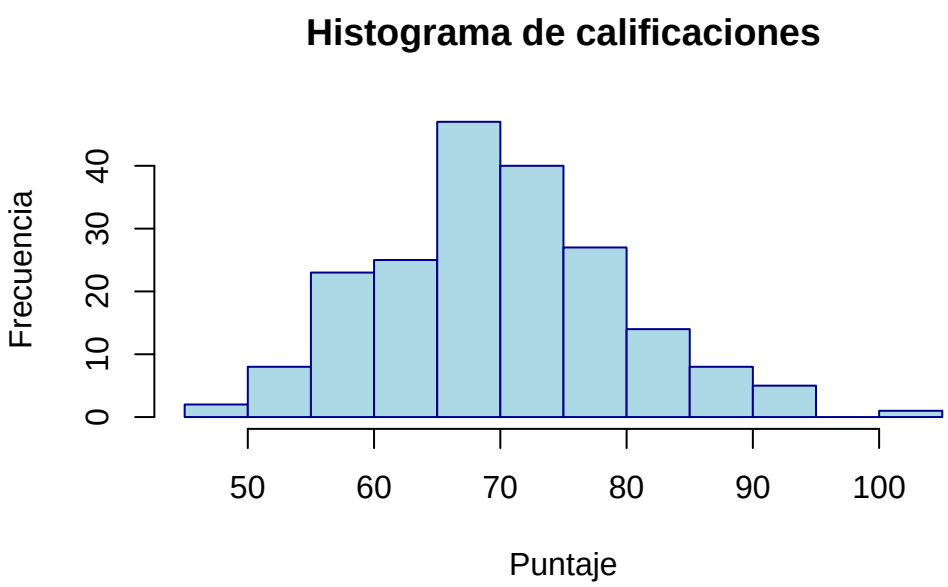
Explicación de los argumentos principales:

1. `x`: Vector numérico con los datos a graficar.
2. `breaks`: Define el número de intervalos (bins) o el método para calcularlos. Puede ser un número, un vector de puntos de corte, o un método como “Sturges”, “Scott”, “FD”.
3. `freq`: Si es TRUE, el eje Y muestra frecuencias absolutas; si es FALSE, muestra densidades.
4. `col`: Color de las barras.
5. `border`: Color del borde de las barras.
6. `main`: Título principal del gráfico.
7. `xlab`, `ylab`: Etiquetas de los ejes X e Y.

8. ...: Otros argumentos gráficos adicionales.

Ejemplo:

```
# Simulación de datos: calificaciones de 200 estudiantes
set.seed(123)
notas <- rnorm(200, mean = 70, sd = 10)

# Creación de un histograma personalizado
hist(notas,
     breaks = 15,          # Número de intervalos (bins)
     freq = TRUE,          # Mostrar frecuencias absolutas en el eje Y
     col = "lightblue",    # Color de las barras
     border = "darkblue", # Color del borde de las barras
     main = "Histograma de calificaciones", # Título principal
     xlab = "Puntaje",     # Etiqueta del eje X
     ylab = "Frecuencia") # Etiqueta del eje Y
```



La elección del número de intervalos (`breaks`) es crucial para evitar interpretaciones erróneas: intervalos muy amplios pueden ocultar detalles importantes, mientras que intervalos muy estrechos pueden generar ruido visual (Venables & Ripley, 2002).

13.3.2 Diagramas de caja (boxplots)

El diagrama de caja, o boxplot, es una herramienta gráfica que resume la dispersión, la mediana y la presencia de valores atípicos en una o varias muestras. Es especialmente útil para comparar grupos y detectar asimetrías (Tukey, 1977).

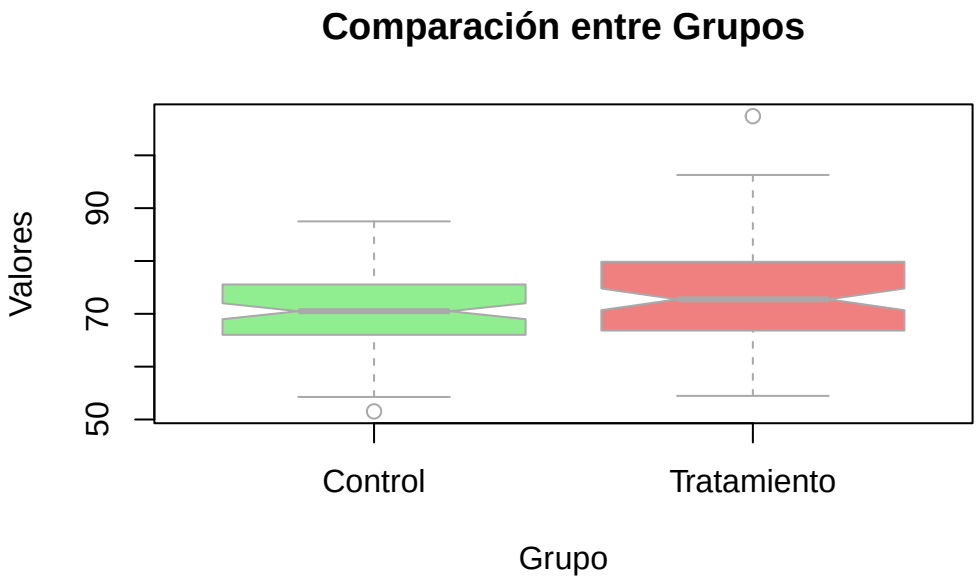
Sintaxis general:

```
boxplot(formula,
        data = NULL,
        main = NULL,
        xlab = NULL,
        ylab = NULL,
        col = NULL,
        border = NULL,
        notch = FALSE,
        outline = TRUE,
        ...)
```

1. `formula`: Expresión del tipo `y ~ grupo` para comparar grupos.
2. `data`: Data frame donde buscar las variables.
3. `main`, `xlab`, `ylab`: Títulos y etiquetas.
4. `col`: Colores de las cajas.
5. `border`: Color del borde de las cajas.
6. `notch`: Si es TRUE, añade una muesca para comparar medianas.
7. `outline`: Si es TRUE, muestra valores atípicos.
8. `...`: Otros argumentos gráficos.

Ejemplo:

```
# Simulación de datos para dos grupos
set.seed(123)
grupo <- factor(rep(c("Control", "Tratamiento"), each = 100))
valores <- c(rnorm(100, 70, 8), rnorm(100, 75, 10))

# Creación de un boxplot personalizado
boxplot(valores ~ grupo,
        main = "Comparación entre Grupos",
        xlab = "Grupo",
        ylab = "Valores",
        col = c("lightgreen", "lightcoral"), # Colores para cada grupo
        border = "darkgray",          # Color del borde
        notch = TRUE,                 # Muesca para comparar medianas
        outline = TRUE)               # Mostrar valores atípicos
```



La muesca en el boxplot ayuda a comparar visualmente las medianas: si las muescas no se superponen, existe evidencia de diferencia significativa entre los grupos (Murrell, 2018).

13.3.3 Gráficos de dispersión

El gráfico de dispersión es fundamental para analizar la relación entre dos variables cuantitativas, permitiendo identificar tendencias lineales, no lineales, agrupamientos y valores atípicos (Cleveland, 1993).

Sintaxis general:

```
plot(x, y,
     type = "p",
     main = NULL,
     sub = NULL,
     xlab = NULL,
     ylab = NULL,
     pch = 1,
     col = NULL,
     cex = 1,
     ...)
```

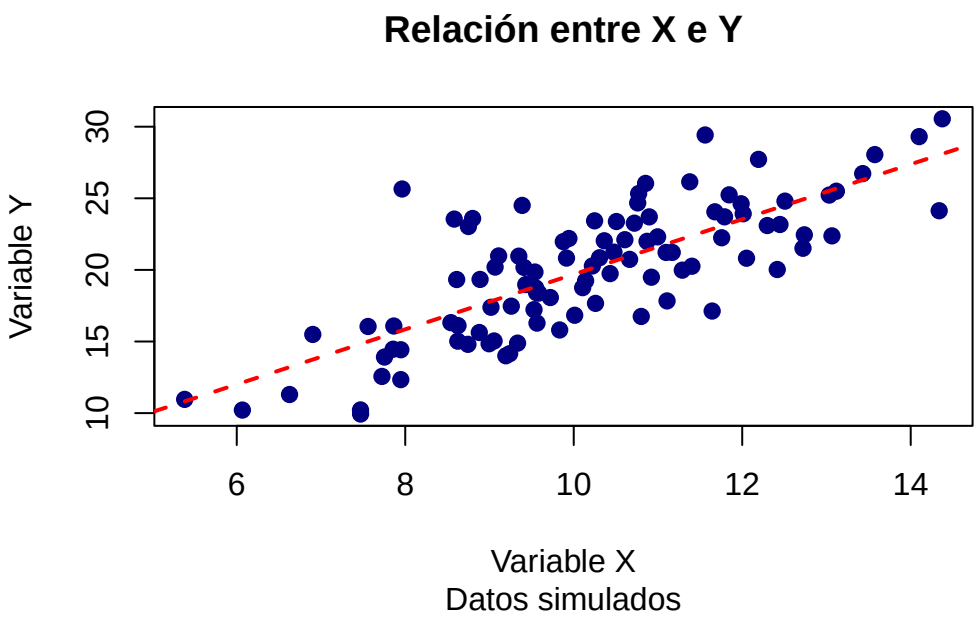
Explicación de los argumentos principales:

1. `x`, `y`: Vectores numéricos de igual longitud.
2. `type`: Tipo de gráfico (“p” para puntos, “l” para líneas, “b” para ambos).
3. `main`, `sub`: Título principal y subtítulo.
4. `xlab`, `ylab`: Etiquetas de los ejes.

5. `pch`: Tipo de símbolo para los puntos (1: círculo, 16: círculo sólido, 17: triángulo, etc.).
6. `col`: Color de los puntos.
7. `cex`: Tamaño relativo de los puntos.
8. `...`: Otros argumentos gráficos.

Ejemplo:

```r
# Simulación de datos correlacionados
set.seed(123)
x <- rnorm(100, mean = 10, sd = 2)
y <- 2 * x + rnorm(100, 0, 3)

# Gráfico de dispersión personalizado
plot(x, y,
     type = "p",                    # Tipo de gráfico: puntos
     main = "Relación entre X e Y",
     sub = "Datos simulados",
     xlab = "Variable X",
     ylab = "Variable Y",
     pch = 16,                      # Círculo sólido
     col = "navy",                  # Color de los puntos
     cex = 1.2)                     # Tamaño de los puntos

# Añadir línea de regresión lineal
abline(lm(y ~ x), col = "red", lwd = 2, lty = 2)
```



La adición de la línea de regresión ayuda a identificar la dirección y fuerza de la relación entre las variables (Venables & Ripley, 2002).

13.3.4 Gráficos de líneas

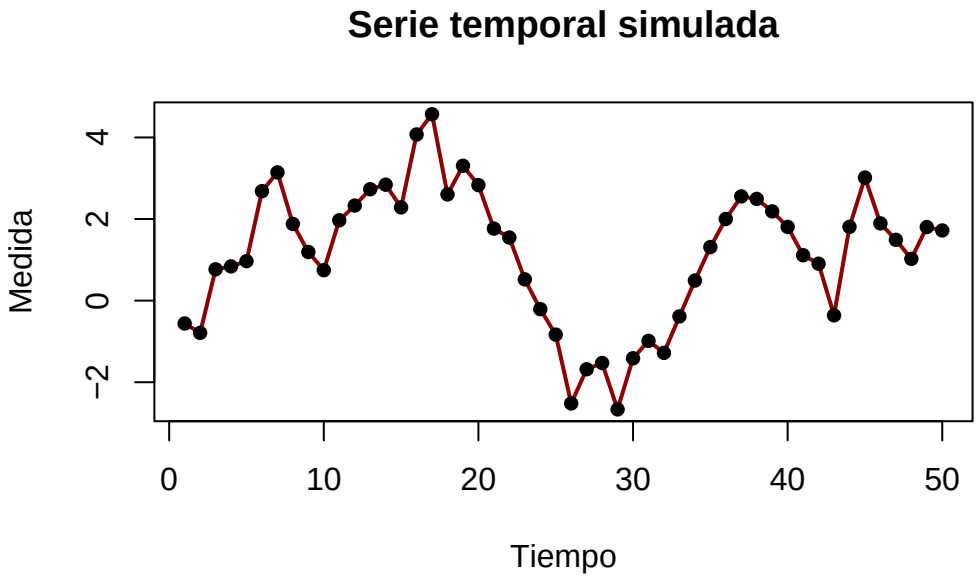
Los gráficos de líneas son ideales para representar la evolución de una variable a lo largo del tiempo o en función de un orden secuencial, permitiendo detectar tendencias, ciclos y cambios abruptos (Murrell, 2018).

```
plot(x, y,
     type = "l",
     main = NULL,
     xlab = NULL,
     ylab = NULL,
     col = NULL,
     lwd = 1,
     ...)
```

1. `type = "l"`: Dibuja una línea.
2. `lwd`: Grosor de la línea.

Ejemplo:

```
# Simulación de una serie temporal
set.seed(123)
tiempo <- 1:50
medidas <- cumsum(rnorm(50))

# Gráfico de líneas
plot(tiempo, medidas,
     type = "l",                  # Tipo de gráfico: línea
     main = "Serie temporal simulada",
     xlab = "Tiempo",
     ylab = "Medida",
     col = "darkred",
     lwd = 2)                     # Grosor de la línea

# Añadir puntos sobre la línea para enfatizar cada observación
points(tiempo, medidas, pch = 16, col = "black")
```



La combinación de líneas y puntos facilita la identificación de valores individuales y la tendencia global de la serie.

13.4 Visualización para la comprobación de supuestos estadísticos

La validación gráfica de los supuestos estadísticos es un paso esencial para garantizar la validez de los análisis en la estadística clásica. Antes de aplicar pruebas como ANOVA o modelos de regresión lineal, es fundamental verificar visualmente la normalidad, la homocedasticidad y la linealidad de los datos. El sistema gráfico base de R proporciona herramientas específicas para evaluar estos supuestos de manera eficiente y pedagógica (Venables & Ripley, 2002; Murrell, 2018).

13.4.1 Gráficos Q-Q: Evaluación visual de la normalidad

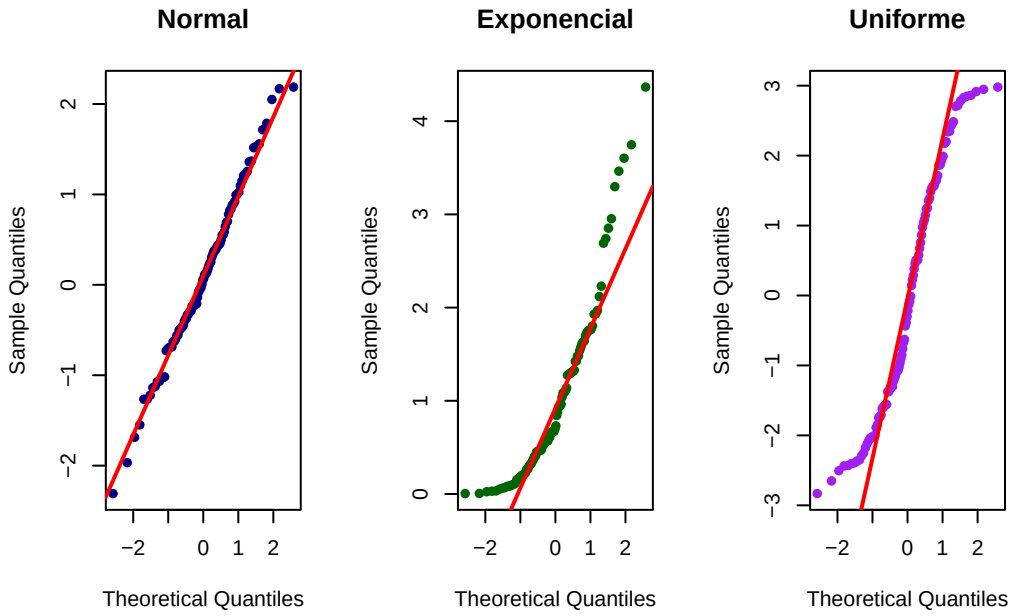
El gráfico Q-Q (quantile-quantile) es una herramienta visual poderosa para comparar la distribución de los datos observados con una distribución teórica, generalmente la normal. Si los puntos del gráfico se alinean sobre la diagonal, se puede inferir que los datos siguen la distribución de referencia. Las desviaciones sistemáticas de esta línea indican alejamientos de la normalidad, lo que puede requerir transformaciones de los datos o el uso de métodos no paramétricos (Cleveland, 1993).

Sintaxis básica y explicación:

1. `qqnorm()`: Genera el gráfico Q-Q de los datos frente a la normal.
2. `qqline()`: Añade la línea de referencia teórica.

Ejemplo:

```
# Simulación de tres conjuntos de datos con diferentes distribuciones
set.seed(123)
# Datos con distribución normal
datos_normales <- rnorm(100, mean = 0, sd = 1)
# Datos con distribución exponencial (asimétrica)
datos_asimetricos <- rexp(100, rate = 1)
# Datos con distribución uniforme
datos_uniformes <- runif(100, min = -3, max = 3)

# Configuración para mostrar tres gráficos en una fila
par(mfrow = c(1, 3))

# Gráfico Q-Q para datos normales
qqnorm(datos_normales,
       main = "Normal",
       pch = 16,                # Círculo sólido
       col = "navy")            # Color de los puntos
qqline(datos_normales, col = "red", lwd = 2)  # Línea de referencia

# Gráfico Q-Q para datos asimétricos
qqnorm(datos_asimetricos,
       main = "Exponencial",
       pch = 16,
       col = "darkgreen")
qqline(datos_asimetricos, col = "red", lwd = 2)

# Gráfico Q-Q para datos uniformes
qqnorm(datos_uniformes,
       main = "Uniforme",
       pch = 16,
       col = "purple")
qqline(datos_uniformes, col = "red", lwd = 2)
```




```
# Restaurar la configuración original de la ventana gráfica
par(mfrow = c(1, 1))
```

La interpretación de estos gráficos se basa en el patrón que forman los puntos en relación con la línea de referencia. Según Venables & Ripley (2002), las desviaciones más comunes incluyen:

1. **Colas pesadas:** cuando los extremos se alejan de la línea.
2. **Asimetría**: cuando se forma un patrón curvilíneo.
3. **Bimodalidad:** cuando aparece un patrón en forma de S.

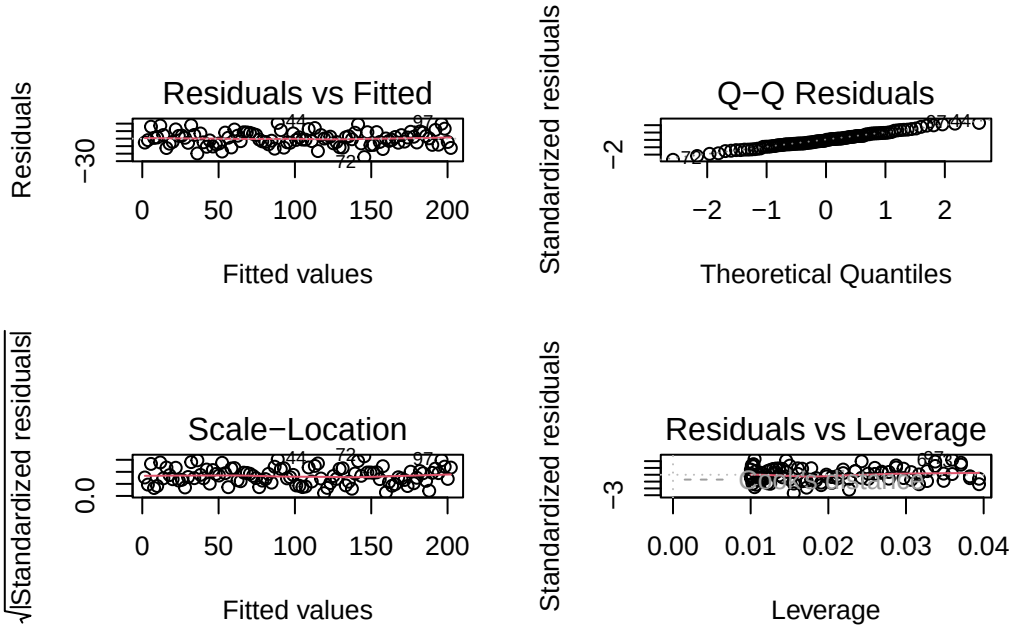
13.4.2 Gráficos de diagnóstico para modelos de regresión

La regresión lineal clásica asume linealidad, normalidad de los residuos, homocedasticidad (varianza constante) e independencia. R facilita la evaluación simultánea de estos supuestos mediante gráficos de diagnóstico automáticos generados con la función `plot()` aplicada a objetos de clase `lm` (Murrell, 2018).

Ejemplo:

```
# Simulación de datos para regresión lineal
set.seed(123)
x <- seq(1, 100)                    # Variable predictora
y <- 2 * x + rnorm(100, 0, 10)      # Variable respuesta
datos <- data.frame(x = x, y = y)   # Crear data frame

# Ajuste del modelo de regresión lineal
modelo <- lm(y ~ x, data = datos)          # Ajustar modelo
```

```r
# Configuración de la ventana gráfica para mostrar cuatro gráficos
par(mfrow = c(2, 2))
plot(modelo)                                  # Generar gráficos diagnósticos
```




```r
par(mfrow = c(1, 1))                          # Restaurar configuración
```

Descripción e interpretación de los gráficos generados:

1. **Residuos vs valores ajustados:** Permite evaluar la linealidad y la homogeneidad de la varianza. Un patrón aleatorio indica que se cumplen los supuestos; patrones sistemáticos sugieren problemas de especificación del modelo.
2. **Q-Q de residuos:** Evalúa la normalidad de los residuos. Desviaciones de la línea diagonal indican que los residuos no son normales.
3. **Scale-Location (raíz cuadrada de los residuos estandarizados vs valores ajustados):** Permite examinar la homogeneidad de la varianza. Una banda horizontal indica homogeneidad de la varianza.
4. **Residuos vs leverage:** Identifica observaciones influyentes. Puntos alejados o con gran leverage pueden indicar outliers o casos influyentes que afectan el ajuste del modelo (Murrell, 2018).

13.5 Personalización de gráficos en R base

La personalización de gráficos es un aspecto fundamental para lograr visualizaciones claras, informativas y estéticamente agradables. En el sistema gráfico base de R, la personalización se realiza mediante la modificación de los argumentos de las funciones

gráficas principales y la incorporación de elementos adicionales a través de funciones auxiliares. Esta flexibilidad permite adaptar cada gráfico a las necesidades específicas del análisis y a los estándares de comunicación científica (Murrell, 2018).

13.5.1 Argumentos y funciones clave para la personalización

A continuación se describen los argumentos y funciones más relevantes para la personalización de gráficos en R base:

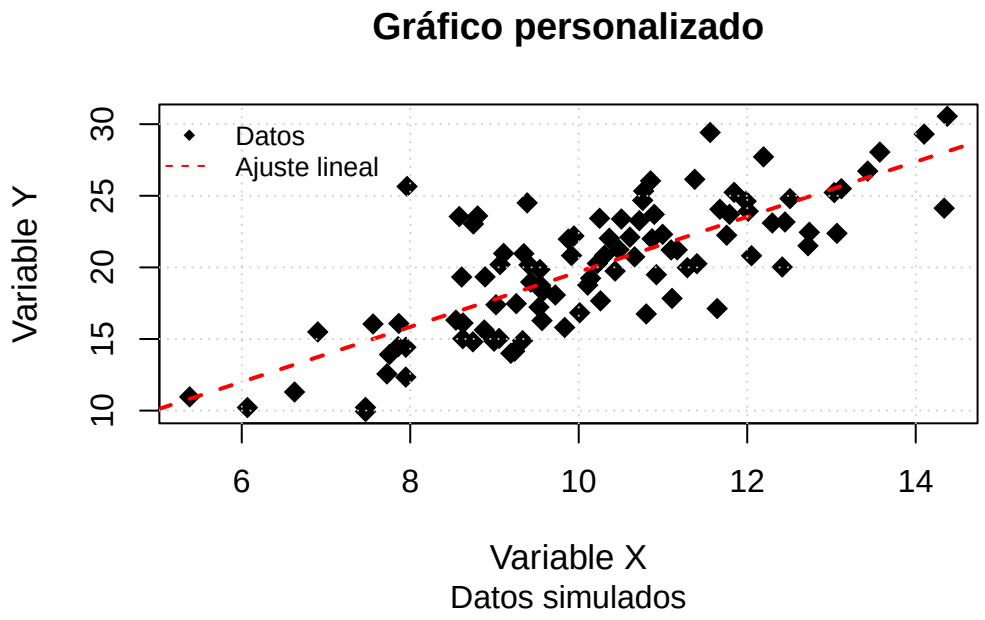
1. **`main, sub, xlab, ylab:`** Permiten definir el título principal, subtítulo y las etiquetas de los ejes X e Y, respectivamente, facilitando la interpretación del gráfico.
2. **`col, border, pch, lty, lwd:`** Controlan el color de los elementos, el color del borde, el tipo de símbolo para los puntos, el tipo de línea y el grosor de las líneas, respectivamente.
3. **`cex, cex.axis, cex.lab, cex.main:`** Ajustan el tamaño relativo de los símbolos, los textos de los ejes, las etiquetas y el título principal.
4. **`legend():`** Añade leyendas explicativas en posiciones específicas del gráfico, mejorando la comprensión de los elementos representados.
5. **`text():`** Permite agregar texto en coordenadas específicas, útil para destacar valores o anotar observaciones relevantes.
6. **`abline():`** Añade líneas horizontales, verticales o de regresión, facilitando la identificación de tendencias o referencias.
7. **`grid():`** Incorpora una cuadrícula de fondo, lo que ayuda a la lectura precisa de las coordenadas y la comparación visual de los datos.

13.5.2 Ejemplo integral

A continuación se presenta un ejemplo completo que ilustra cómo combinar estos argumentos y funciones para lograr una visualización profesional y clara:

```
# Simulación de datos para el ejemplo
set.seed(123)
x <- rnorm(100, mean = 10, sd = 2)
y <- 2 * x + rnorm(100, 0, 3)

# Gráfico de dispersión personalizado
plot(x, y,
     main = "Gráfico personalizado",  # Título principal
     sub = "Datos simulados",         # Subtítulo
     xlab = "Variable X",             # Etiqueta eje X
     ylab = "Variable Y",             # Etiqueta eje Y
     col = "black",                   # Color de los puntos
     pch = 18,                        # Símbolo: rombo sólido
     cex = 1.5,                       # Tamaño de los puntos
     cex.main = 1.2,                  # Tamaño del título
```

```r
     cex.lab = 1.1)                    # Tamaño de las etiquetas

# Añadir línea de regresión lineal
abline(lm(y ~ x), col = "red", lwd = 2, lty = 2)  # Línea de tendencia

# Añadir leyenda explicativa
legend("topleft",
       legend = c("Datos", "Ajuste lineal"),
       pch = c(18, NA),                  # Símbolo para los datos
       lty = c(NA, 2),                   # Línea discontinua para el ajuste
       col = c("black", "red"),
       bty = "n",                        # Sin borde en la leyenda
       cex = 0.8)                        # Tamaño de la leyenda

# Añadir cuadrícula de fondo
## Cuadrícula con líneas punteadas grises
grid(col = "gray80", lty = "dotted")
```



La personalización adecuada de los gráficos no solo mejora la estética, sino que también facilita la interpretación y la comunicación de los resultados, permitiendo resaltar los aspectos más relevantes del análisis (Murrell, 2018; Venables & Ripley, 2002).

14 Visualización de datos con ggplot2

En el contexto del análisis estadístico moderno, la visualización de datos constituye una herramienta esencial para la exploración, interpretación y comunicación de resultados. Si bien el sistema gráfico base de R ofrece una amplia variedad de funciones para la creación de gráficos, la creciente demanda de visualizaciones más sofisticadas, reproducibles y estéticamente profesionales ha impulsado el desarrollo de herramientas especializadas, entre las cuales destaca el paquete **ggplot2** (Wickham, 2016).

ggplot2 es un paquete de R diseñado para la creación de gráficos estadísticos de alta calidad, basado en la "gramática de los gráficos" (Grammar of Graphics) propuesta por Wilkinson (2005). Esta gramática proporciona un marco conceptual que permite construir visualizaciones complejas a partir de componentes independientes y combinables, facilitando la personalización y la integración de múltiples capas de información en un solo gráfico.

El uso de ggplot2 se ha consolidado como un estándar en la comunidad científica y profesional debido a varias razones fundamentales:

1. **Claridad y profesionalismo:** Los gráficos generados con ggplot2 cumplen con los estándares visuales requeridos en publicaciones científicas, reportes técnicos y presentaciones académicas.
2. **Flexibilidad y modularidad:** La estructura de ggplot2 permite añadir, modificar o eliminar elementos gráficos de manera sencilla, adaptando cada visualización a las necesidades específicas del análisis.
3. **Reproducibilidad:** La sintaxis declarativa de ggplot2 facilita la documentación y replicación exacta de los gráficos, aspecto crucial en la investigación científica y la docencia.
4. **Integración con el ecosistema tidyverse:** ggplot2 forma parte del conjunto de paquetes tidyverse, lo que permite una integración fluida con herramientas para manipulación, transformación y modelado de datos (Wickham et al., 2019).

En síntesis, ggplot2 es una herramienta indispensable para quienes buscan comunicar resultados estadísticos de manera clara, precisa y profesional. Su adopción en entornos académicos y profesionales responde a la necesidad de contar con visualizaciones que no solo sean informativas, sino también estéticamente adecuadas para su inclusión en documentos formales y publicaciones científicas.

14.1 Ventajas principales de ggplot2

El paquete **ggplot2** se ha consolidado como una de las herramientas más utilizadas para la visualización de datos en R, tanto en el ámbito académico como profesional. Su

popularidad se debe a una serie de ventajas que lo distinguen frente a otros sistemas gráficos, especialmente en el contexto del análisis estadístico y la elaboración de documentos formales.

1. **Modularidad y gramática de gráficos:** ggplot2 está basado en la "gramática de los gráficos" (Grammar of Graphics), lo que permite construir visualizaciones a partir de componentes independientes: datos, mapeos estéticos, geometrías, escalas, temas y capas adicionales. Esta modularidad facilita la creación de gráficos complejos de manera incremental, permitiendo añadir o modificar elementos sin rehacer el gráfico desde cero (Wilkinson, 2005; Wickham, 2016).
2. **Flexibilidad y personalización:** A diferencia del sistema gráfico base de R, ggplot2 ofrece una amplia gama de opciones para personalizar cada aspecto del gráfico, desde los colores y tipos de símbolos hasta la disposición de leyendas, títulos y escalas. Esta flexibilidad es fundamental para adaptar las visualizaciones a los estándares de publicaciones científicas y a las necesidades específicas de cada análisis (Wickham, 2016).
3. **Resultados visuales profesionales:** Los gráficos generados con ggplot2 presentan una estética cuidada y profesional por defecto, lo que facilita su inclusión directa en artículos científicos, reportes técnicos y presentaciones académicas. Además, la posibilidad de aplicar temas predefinidos o personalizados permite mantener la coherencia visual en todos los productos gráficos de un proyecto (Wickham, 2016).
4. **Reproducibilidad y transparencia:** La sintaxis declarativa de ggplot2 favorece la reproducibilidad de los análisis, ya que cada gráfico puede ser reconstruido exactamente a partir del código utilizado. Esto es especialmente relevante en la investigación científica, donde la transparencia y la replicabilidad son principios fundamentales (Wickham et al., 2019).
5. **Integración con el ecosistema tidyverse:** ggplot2 forma parte del tidyverse, un conjunto de paquetes diseñados para el manejo, transformación y modelado de datos en R. Esta integración permite una transición fluida desde la manipulación de datos hasta la visualización, optimizando el flujo de trabajo y reduciendo la posibilidad de errores (Wickham et al., 2019).
6. **Comunidad activa y abundancia de recursos:** La amplia adopción de ggplot2 ha dado lugar a una comunidad activa de usuarios y desarrolladores, lo que se traduce en una gran cantidad de recursos, tutoriales, ejemplos y extensiones disponibles para resolver dudas y ampliar las capacidades del paquete.

En conjunto, estas ventajas hacen de ggplot2 una herramienta indispensable para quienes buscan comunicar resultados estadísticos de manera clara, precisa y profesional, cumpliendo con los estándares de calidad exigidos en la ciencia y la industria.

14.2 Gramática de los Gráficos en ggplot2

La visualización de datos es una etapa fundamental en el análisis estadístico, ya que permite identificar patrones, tendencias y anomalías que pueden pasar desapercibidos en una inspección numérica (Cleveland, 1993; Tufte, 2001). En este contexto, **ggplot2** se destaca por su enfoque basado en la "gramática de los gráficos" (*Grammar of Graphics*),

un marco conceptual que facilita la construcción de visualizaciones claras, reproducibles y adaptadas a los estándares de la comunicación científica (Wilkinson, 2005; Wickham, 2016).

14.2.1 Principios conceptuales de la gramática de los gráficos

La gramática de los gráficos, propuesta originalmente por Wilkinson (2005), parte de la premisa de que toda visualización estadística puede descomponerse en un conjunto de componentes básicos. Este enfoque modular permite construir gráficos complejos a partir de la combinación sistemática de elementos independientes, lo que resulta especialmente útil para quienes se inician en la programación estadística, ya que reduce la complejidad y favorece la comprensión progresiva del proceso de visualización (Wickham, 2016).

Según Cleveland (1993), la claridad y la precisión en la representación gráfica son esenciales para evitar interpretaciones erróneas y comunicar los resultados de manera efectiva. Por ello, la gramática de los gráficos enfatiza la importancia de definir explícitamente cada elemento visual, asegurando que el gráfico resultante sea informativo y estéticamente adecuado (Tufte, 2001).

14.2.2 Componentes esenciales de un gráfico en ggplot2

A continuación se describen los principales componentes que conforman la gramática de los gráficos en ggplot2, siguiendo la estructura propuesta por Wilkinson (2005) y adaptada por Wickham (2016):

1. **Datos:** Constituyen el insumo fundamental de cualquier gráfico. En R, los datos suelen organizarse en data frames, lo que facilita su manipulación y visualización (Wickham & Grolemund, 2017).
2. **Mapeos estéticos (*aesthetics*):** Son las correspondencias entre las variables de los datos y las propiedades visuales del gráfico, como la posición en los ejes, el color, el tamaño o la forma de los elementos. Definir correctamente los mapeos es crucial para garantizar que la visualización transmita la información deseada (Wickham, 2016).
3. **Geometrías (*geoms*):** Representan los objetos gráficos que visualizan los datos, como puntos (`geom_point()`), líneas (`geom_line()`), barras (`geom_bar()`), cajas (`geom_boxplot()`), entre otros. La elección de la geometría depende del tipo de variable y del objetivo del análisis (Cleveland, 1993).
4. **Transformaciones estadísticas (*stats*):** Permiten aplicar cálculos o resúmenes estadísticos antes de la representación gráfica, como medias, medianas, conteos o ajustes de modelos. Por ejemplo, `geom_smooth()` puede añadir una línea de tendencia basada en un modelo de regresión (Wickham, 2016).
5. **Escalas:** Definen cómo se traducen los valores de las variables a propiedades visuales, por ejemplo, escalas de color, tamaño o forma. Las escalas permiten adaptar el gráfico a diferentes contextos y audiencias (Wilkinson, 2005).

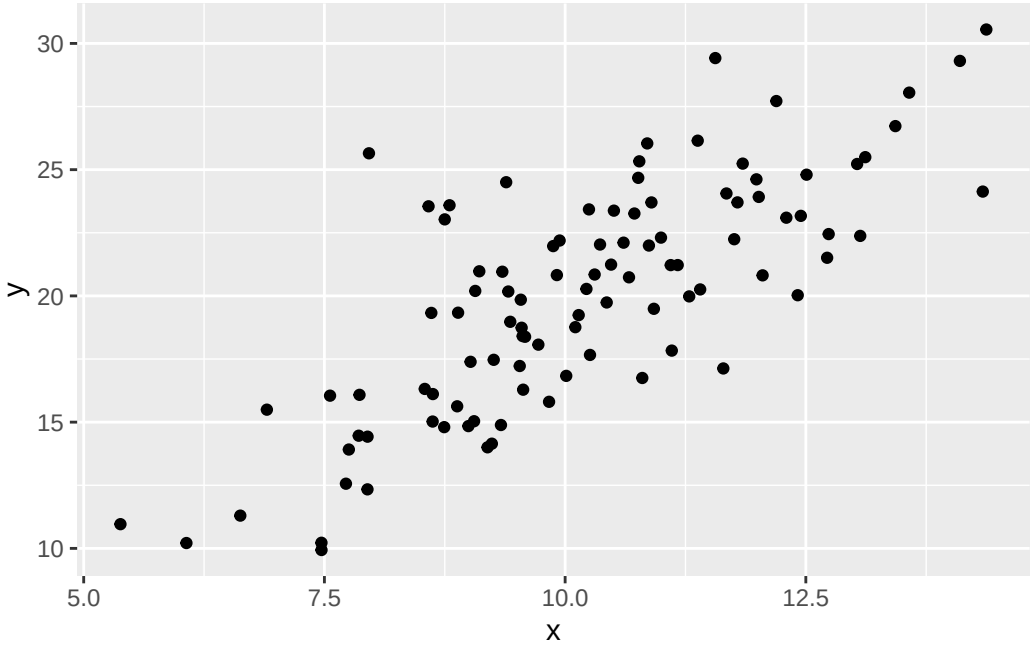
6. **Sistemas de coordenadas:** Determinan el sistema de referencia utilizado para ubicar los elementos gráficos, siendo el cartesiano el más común, aunque también se pueden emplear coordenadas polares u otras transformaciones (Wickham, 2016).
7. **Facetas:** Permiten dividir el gráfico en subgráficos según los valores de una o más variables categóricas, facilitando la comparación visual entre grupos o condiciones experimentales (Wickham, 2016).
8. **Temas:** Controlan la apariencia general del gráfico, incluyendo el tipo y tamaño de fuente, colores de fondo, líneas de cuadrícula y otros elementos estéticos globales. La personalización de temas es fundamental para adaptar los gráficos a los estándares de publicaciones científicas (Tufte, 2001; Wickham, 2016).

14.2.3 Construcción secuencial y sintaxis básica de gráficos en ggplot2

La sintaxis de ggplot2 se basa en la adición secuencial de capas, donde cada componente se incorpora mediante el operador +. Este enfoque modular permite construir gráficos de manera progresiva, añadiendo o modificando elementos según las necesidades del análisis (Wickham, 2016).

Ejemplo con datos simulados:

```
# Cargar el paquete ggplot2
library(ggplot2)

# Simulación de datos
set.seed(123)
x <- rnorm(100, mean = 10, sd = 2)
y <- 2 * x + rnorm(100, 0, 3)
datos <- data.frame(x = x, y = y)

# Construcción secuencial de un gráfico de dispersión
ggplot(datos, aes(x = x, y = y)) +   # Inicialización y mapeo estético
  geom_point()                       # Capa de geometría: puntos
```



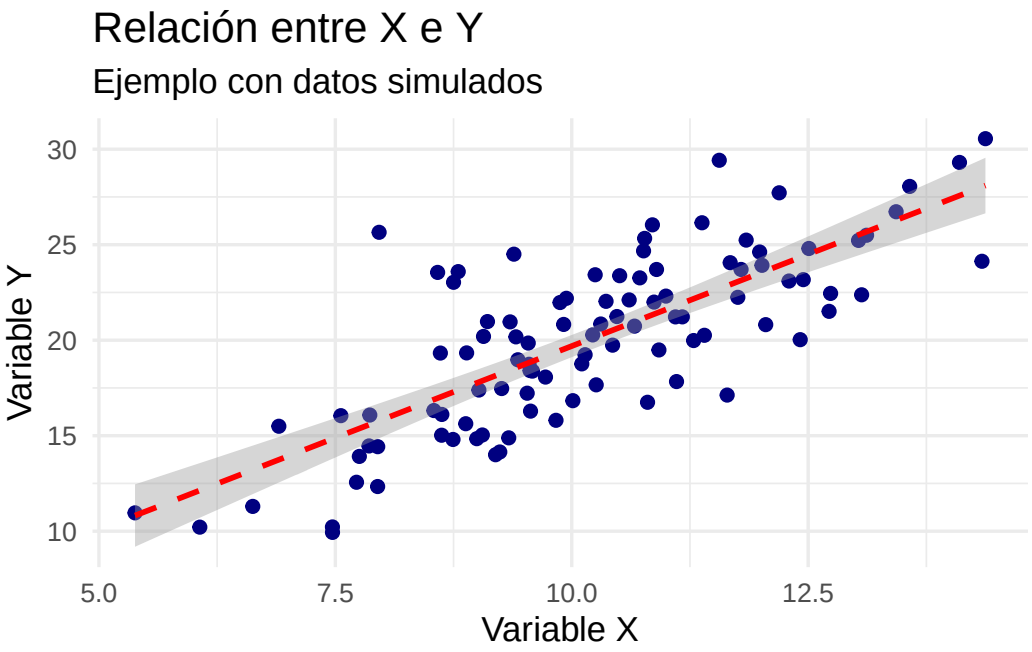
Explicación:

1. `ggplot(datos, aes(x = x, y = y))` inicializa el objeto gráfico y define que la variable `x` se mapea al eje horizontal y `y` al eje vertical.
2. `geom_point()` añade la capa de puntos, representando cada observación como un símbolo en el plano cartesiano.
3. El operador `+` permite añadir más capas o personalizaciones de manera sencilla y ordenada.

14.2.4 Ejemplo avanzado: Incorporación de capas y personalización

La verdadera potencia de ggplot2 se manifiesta al combinar múltiples capas y personalizaciones en un solo gráfico. A continuación se muestra cómo añadir una línea de tendencia y personalizar etiquetas y temas, siguiendo las recomendaciones de claridad y economía visual de Tufte (2001):

```
ggplot(datos, aes(x = x, y = y)) +
  # Puntos personalizados
  geom_point(color = "navy", size = 2) +
  # Línea de regresión
  geom_smooth(method = "lm", color = "red", linetype = "dashed") +
  labs(
    title = "Relación entre X e Y",
    subtitle = "Ejemplo con datos simulados",
    x = "Variable X",
    y = "Variable Y"
  ) +
  # Tema profesional y limpio
  theme_minimal(base_size = 13)
```



Explicación:

1. `geom_smooth(method = "lm", ...)` añade una línea de regresión lineal con estilo personalizado, facilitando la interpretación de la tendencia general de los datos (Cleveland, 1993).
2. `labs()` permite definir títulos y etiquetas descriptivas, mejorando la claridad del gráfico.
3. `theme_minimal()` aplica un tema visual adecuado para presentaciones y publicaciones, siguiendo los principios de economía visual (Tufte, 2001).

14.2.5 Ventajas del enfoque modular y declarativo

El enfoque modular y declarativo de ggplot2 ofrece ventajas significativas para principiantes y usuarios avanzados:

1. Permite construir gráficos complejos de manera incremental y reproducible, facilitando el aprendizaje progresivo (Wickham, 2016).
2. Facilita la modificación y personalización de cada elemento visual, adaptando los gráficos a diferentes audiencias y contextos (Wilkinson, 2005).
3. Favorece la claridad y la transparencia en la comunicación de resultados, aspectos esenciales en la investigación científica y la docencia (Cleveland, 1993; Tufte, 2001).

14.3 Estructura y Flujo de Trabajo para la Construcción de Gráficos en ggplot2

La creación de gráficos profesionales en **ggplot2** sigue un flujo de trabajo sistemático y reproducible, que facilita tanto el aprendizaje para principiantes como la producción de visualizaciones de alta calidad para informes y publicaciones científicas (Wickham, 2016; Wilkinson, 2005). Comprender este workflow es esencial para aprovechar al máximo las capacidades del paquete y garantizar la claridad y la coherencia en la comunicación de resultados.

14.3.1 Preparación y organización de los datos

El primer paso en cualquier proceso de visualización es la preparación de los datos. En R, los datos suelen organizarse en data frames, lo que permite una manipulación eficiente y una integración directa con ggplot2 (Wickham & Grolemund, 2017). Es fundamental asegurarse de que los datos estén limpios, estructurados y listos para ser mapeados a los elementos visuales del gráfico.

Ejemplo:

```
# Simulación de datos para el ejemplo
set.seed(123)
x <- rnorm(100, mean = 10, sd = 2)
y <- 2 * x + rnorm(100, 0, 3)
datos <- data.frame(x = x, y = y)
```

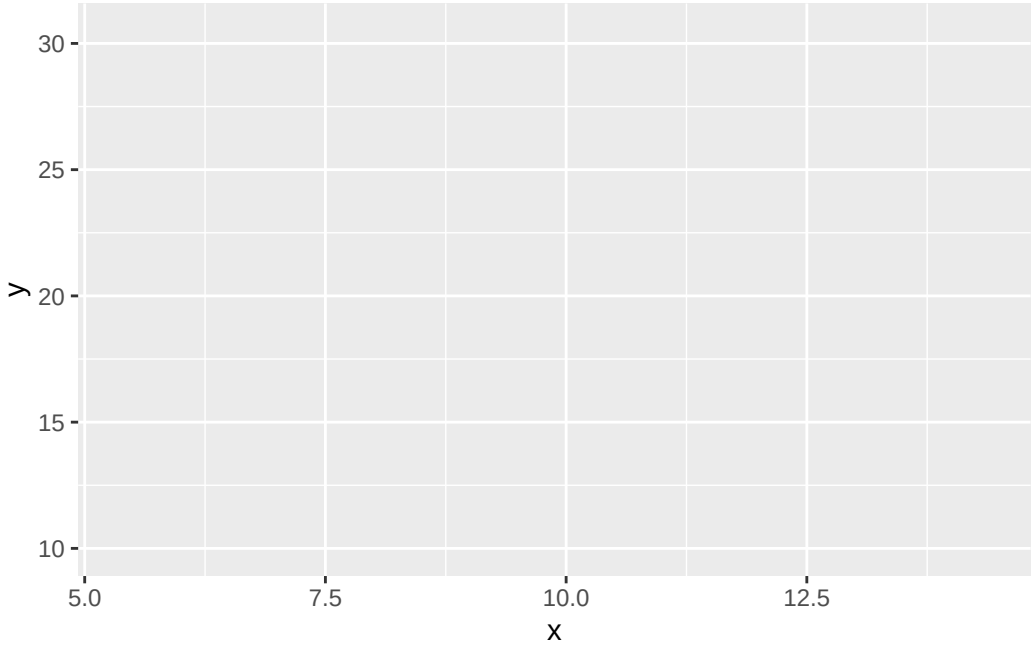
Explicación: Se simulan dos variables numéricas (`x` e `y`) y se almacenan en un data frame llamado `datos`, siguiendo las mejores prácticas de organización de datos para análisis estadístico (Wickham & Grolemund, 2017).

14.3.2 Inicialización del objeto gráfico y definición de mapeos estéticos

El flujo de trabajo (*workflow*) de ggplot2 comienza con la inicialización del objeto gráfico mediante la función `ggplot()`, donde se especifica el data frame y los mapeos estéticos principales a través de la función `aes()`. Los mapeos estéticos determinan cómo se asignan las variables de los datos a las propiedades visuales del gráfico, como los ejes, el color, el tamaño o la forma (Wickham, 2016).

Ejemplo:

```
# Inicialización del objeto gráfico con mapeos estéticos
grafico_base <- ggplot(datos, aes(x = x, y = y))
grafico_base
```



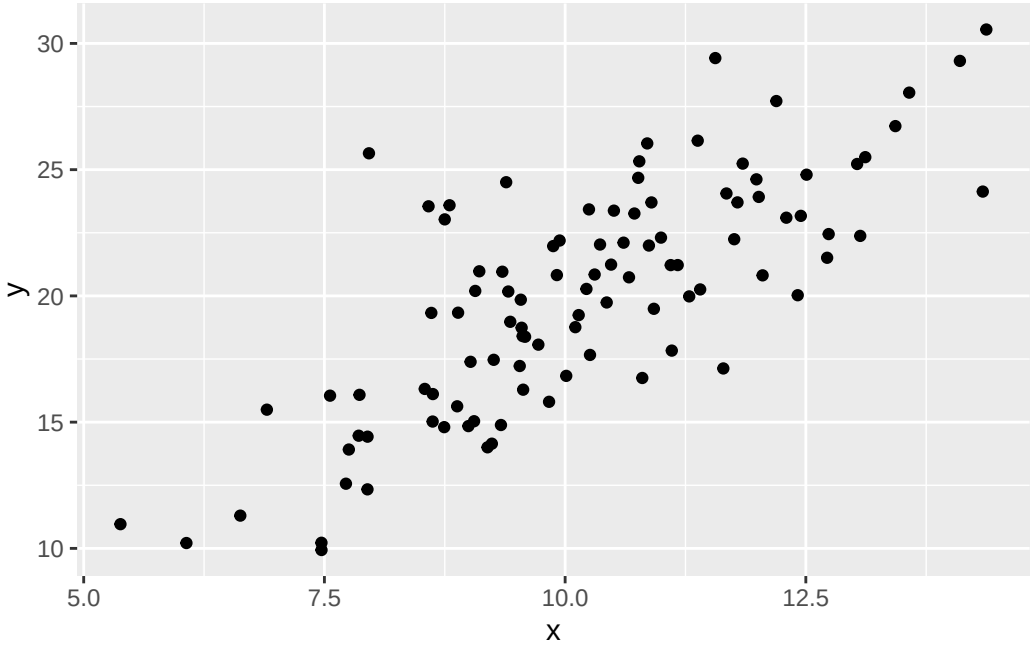
Explicación: Se crea un objeto gráfico vacío, donde se define que la variable `x` se ubicará en el eje horizontal y `y` en el eje vertical. Este objeto sirve como base para añadir capas adicionales.

14.3.3 Adición de geometrías para la representación visual

El siguiente paso consiste en añadir una o más geometrías, que determinan cómo se visualizarán los datos. Las geometrías más comunes incluyen puntos (`geom_point()`), líneas (`geom_line()`), barras (`geom_bar()`) y cajas (`geom_boxplot()`). Cada geometría puede personalizarse mediante argumentos adicionales, como color, tamaño o forma (Cleveland, 1993; Wickham, 2016).

Ejemplo:

```
# Adición de una geometría de puntos
grafico_dispersion <- grafico_base + geom_point()
grafico_dispersion
```



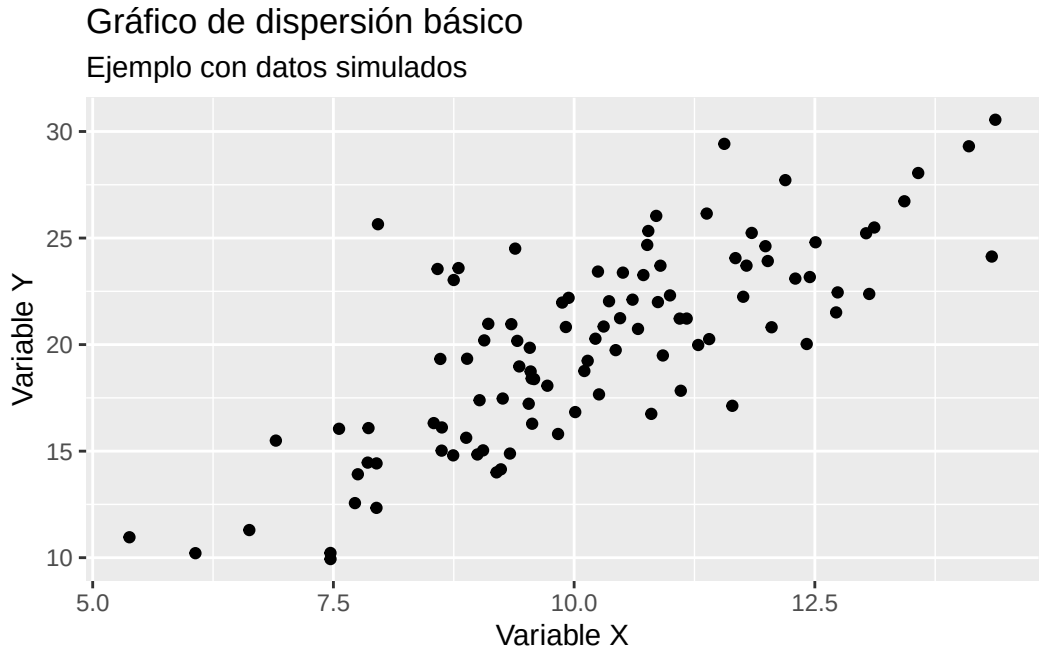
Explicación: `geom_point()` añade una capa de puntos, representando cada observación como un símbolo en el plano cartesiano. El operador `+` permite añadir más capas o personalizaciones de manera ordenada y progresiva.

14.3.4 Personalización de etiquetas, títulos y leyendas

Para mejorar la claridad y la interpretación del gráfico, es recomendable añadir títulos, subtítulos, etiquetas a los ejes y leyendas mediante la función `labs()`. Una correcta rotulación facilita la comunicación de los resultados y evita ambigüedades (Tufte, 2001).

Ejemplo:

```
# Personalización de etiquetas y títulos
grafico_etiquetado <- grafico_dispersion +
  labs(
    title = "Gráfico de dispersión básico",
    subtitle = "Ejemplo con datos simulados",
    x = "Variable X",
    y = "Variable Y"
  )
grafico_etiquetado
```



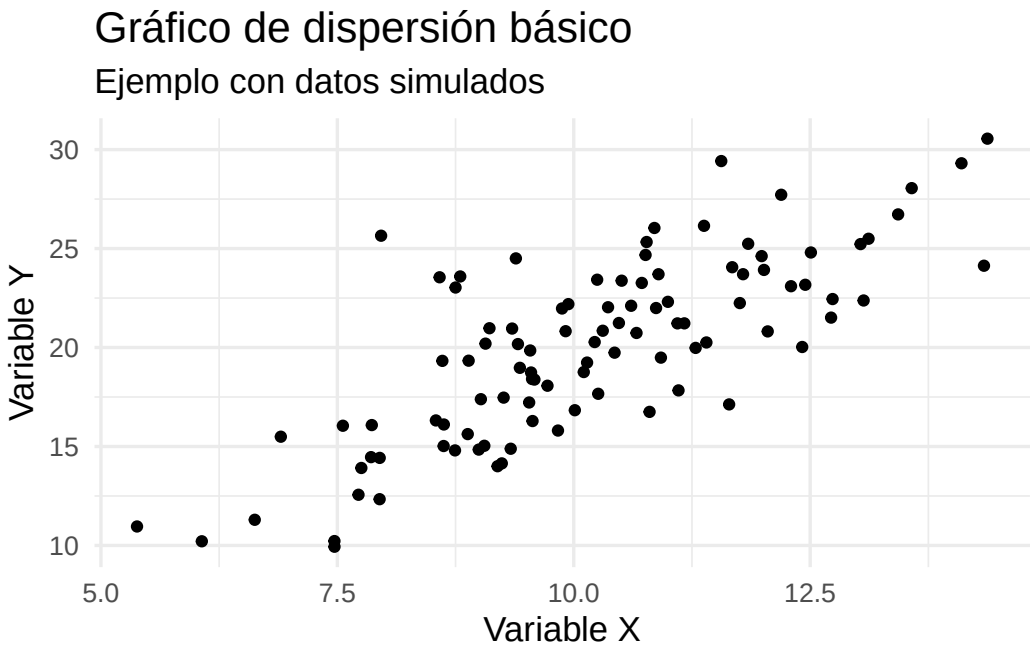
Explicación: `labs()` permite definir el título principal, el subtítulo y las etiquetas de los ejes, mejorando la presentación y la comprensión del gráfico.

14.3.5 Aplicación de temas y ajustes estéticos

Ggplot2 ofrece una variedad de temas predefinidos que modifican la apariencia general del gráfico, adaptándolo a diferentes contextos y estándares de publicación. Los temas controlan aspectos como el fondo, las fuentes, las líneas de cuadrícula y los colores (Wickham, 2016; Tufte, 2001).

Ejemplo:

```
# Aplicación de un tema profesional
grafico_final <- grafico_etiquetado +
  theme_minimal(base_size = 13)
grafico_final
```



Explicación: `theme_minimal()` aplica un estilo visual limpio y profesional, adecuado para presentaciones y publicaciones científicas. El argumento `base_size` ajusta el tamaño base de las fuentes, facilitando la lectura.

14.3.6 Exportación y reutilización del gráfico

Una vez finalizado el gráfico, es posible exportarlo a diferentes formatos (PNG, PDF, SVG) utilizando funciones como `ggsave()`, lo que facilita su inclusión en documentos formales, reportes técnicos y publicaciones científicas (Wickham, 2016).

Ejemplo:

```
# Exportar el gráfico a un archivo PNG
ggsave("grafico_dispersión.png",
       plot = grafico_final,
       width = 6, height = 4, dpi = 300)
```

Explicación: `ggsave()` permite guardar el gráfico en un archivo con la resolución y dimensiones especificadas, asegurando la calidad necesaria para su uso profesional.

14.3.7 Resumen del flujo de trabajo en ggplot2

El flujo de trabajo recomendado para la construcción de gráficos en ggplot2 puede resumirse en los siguientes pasos:

1. **Preparar y organizar los datos** en un formato adecuado (data frame).
2. **Inicializar el objeto gráfico** con los mapeos estéticos principales.
3. **Añadir geometrías** para representar los datos visualmente.

4. **Personalizar etiquetas, títulos y leyendas** para mejorar la claridad.
5. **Aplicar temas y ajustes estéticos** para adaptar el gráfico a los estándares profesionales.
6. **Exportar y reutilizar el gráfico** en diferentes formatos según las necesidades del proyecto.

Este flujo de trabajo modular y progresivo no solo facilita el aprendizaje para principiantes, sino que también garantiza la reproducibilidad, la claridad y la calidad en la comunicación de resultados estadísticos (Wickham, 2016; Wilkinson, 2005; Tufte, 2001).

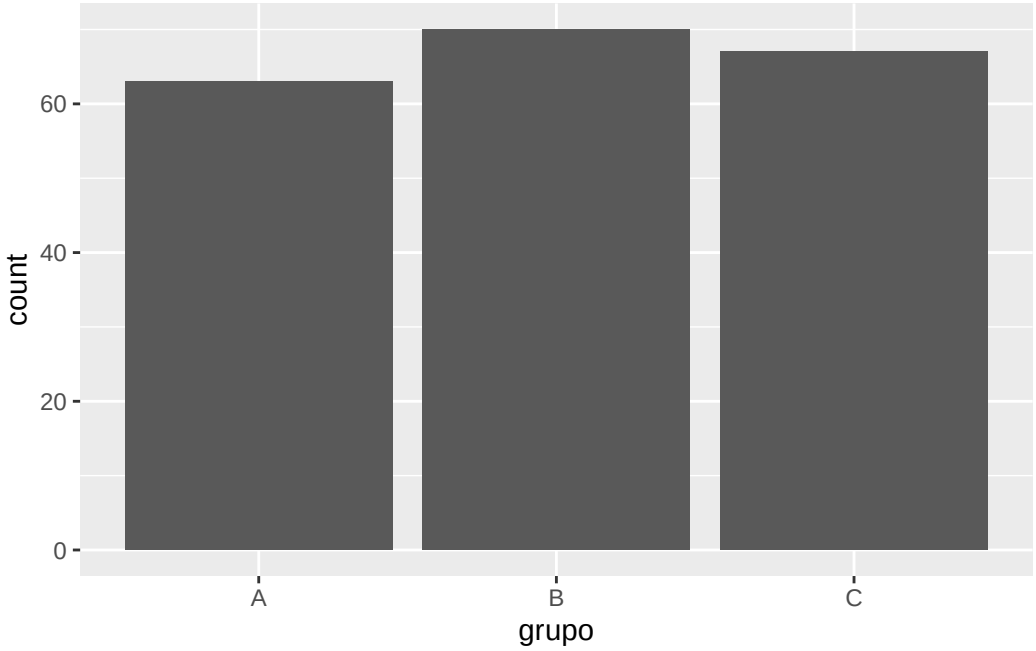
14.4 Creación de gráficos exploratorios y descriptivos en ggplot2

El paquete **ggplot2** proporciona una sintaxis coherente y modular para la creación de diferentes tipos de gráficos estadísticos en R. Cada visualización requiere una estructura específica de datos, generalmente en formato data frame, y utiliza funciones geométricas particulares que determinan cómo se representarán los datos. La construcción de estos gráficos sigue el workflow profesional establecido, donde primero se preparan los datos, luego se inicializa el objeto gráfico con `ggplot()`, se añaden las geometrías correspondientes y finalmente se personalizan los elementos visuales según sea necesario (Wickham, 2016).

14.4.1 Gráficos de barras con geom_bar()

La función `geom_bar()` es la geometría principal para crear gráficos de barras a partir de variables categóricas. Esta función cuenta automáticamente las frecuencias de cada categoría y las representa como barras verticales.

```
# Simulación de datos categóricos
set.seed(123)
grupo <- sample(c("A", "B", "C"), size = 200, replace = TRUE)
datos_cat <- data.frame(grupo = grupo)

# Gráfico de barras
ggplot(datos_cat, aes(x = grupo)) +
  geom_bar()
```



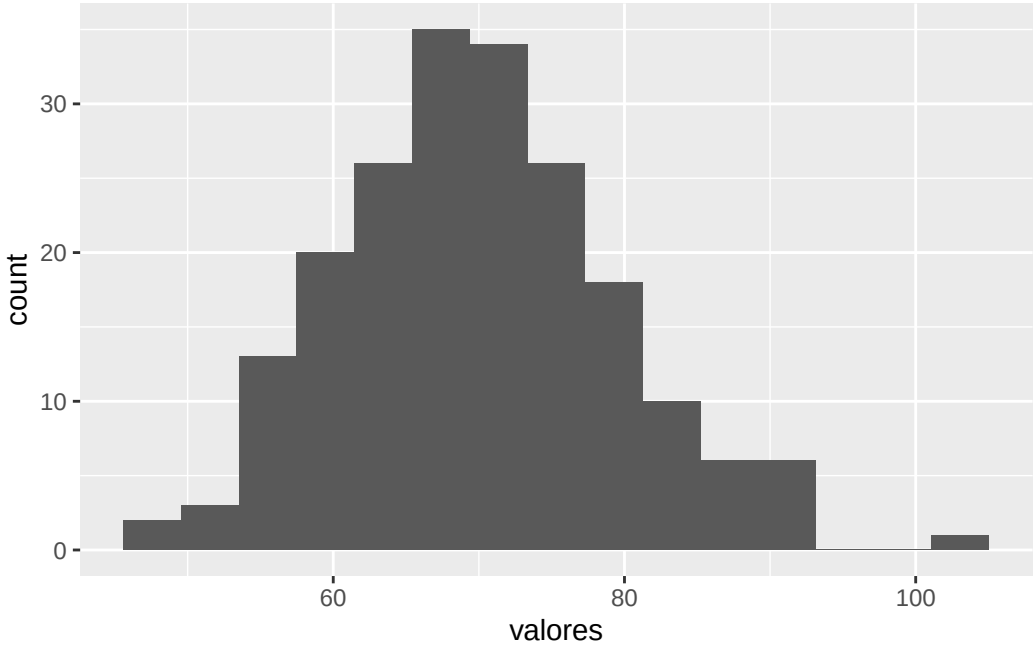
Explicación:

1. Se crea un data frame con una variable categórica (`grupo`).
2. `ggplot(datos_cat, aes(x = grupo))` inicializa el gráfico mapeando la variable al eje X.
3. `geom_bar()` añade las barras, calculando automáticamente las frecuencias.

14.4.2 Histogramas con geom_histogram()

La función `geom_histogram()` genera histogramas para variables numéricas continuas, dividiendo los datos en intervalos (bins) y contando la frecuencia en cada uno.

```
# Simulación de datos numéricos
set.seed(123)
valores <- rnorm(200, mean = 70, sd = 10)
datos_hist <- data.frame(valores = valores)

# Histograma
ggplot(datos_hist, aes(x = valores)) +
  geom_histogram(bins = 15)
```



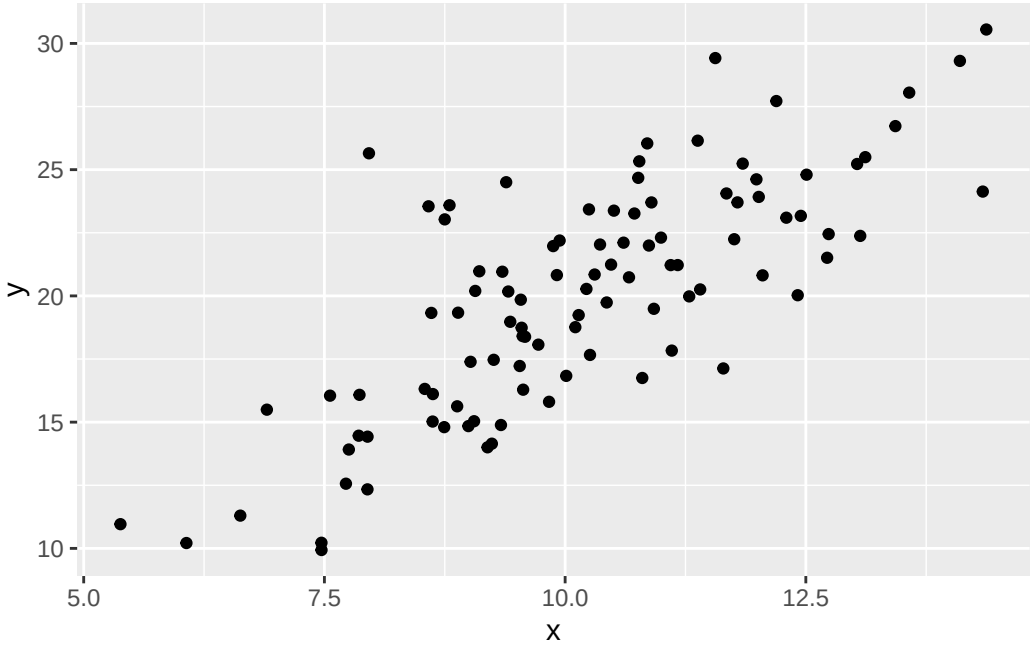
Explicación:

1. Se crea un data frame con una variable numérica (`valores`).
2. `ggplot(datos_hist, aes(x = valores))` mapea la variable al eje X.
3. `geom_histogram()` crea el histograma, especificando el número de bins deseado.

14.4.3 Gráficos de dispersión con geom_point()

La función `geom_point()` crea gráficos de dispersión para analizar la relación entre dos variables numéricas, representando cada observación como un punto en el plano cartesiano.

```
# Simulación de datos correlacionados
set.seed(123)
x <- rnorm(100, mean = 10, sd = 2)
y <- 2 * x + rnorm(100, 0, 3)
datos_disp <- data.frame(x = x, y = y)

# Gráfico de dispersión
ggplot(datos_disp, aes(x = x, y = y)) +
  geom_point()
```



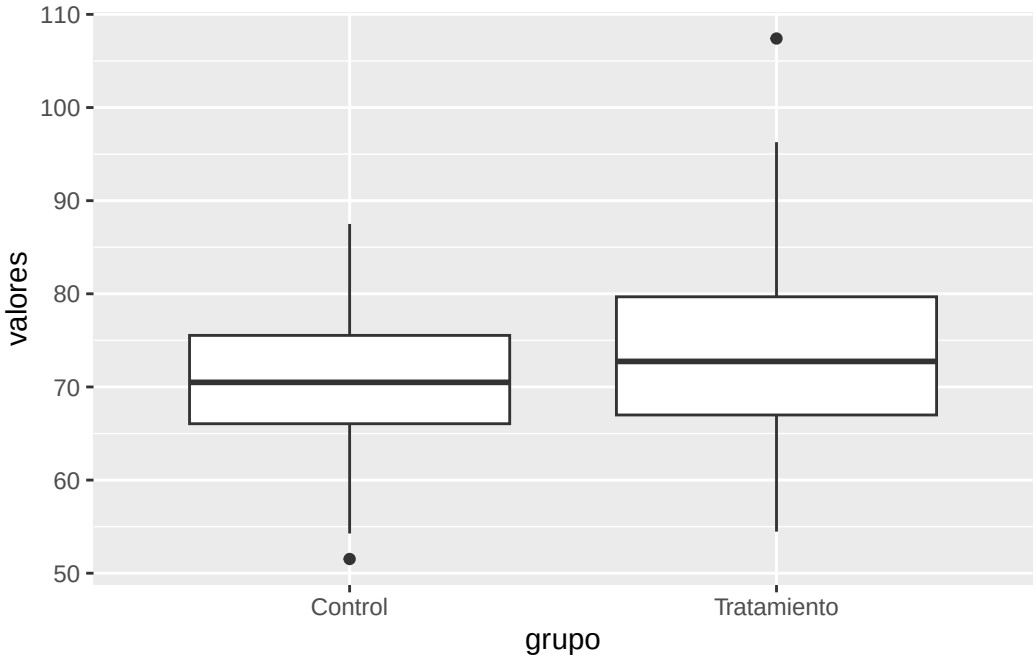
Explicación:

1. Se crea un data frame con dos variables numéricas (`x` e `y`).
2. `ggplot(datos_disp, aes(x = x, y = y))` mapea las variables a los ejes X e Y.
3. `geom_point()` representa cada par de valores como un punto.

14.4.4 Boxplots con geom_boxplot()

La función `geom_boxplot()` construye diagramas de caja para comparar la distribución de una variable numérica entre diferentes grupos o categorías.

```
# Simulación de datos para dos grupos
set.seed(123)
grupo <- factor(rep(c("Control", "Tratamiento"), each = 100))
valores <- c(rnorm(100, 70, 8), rnorm(100, 75, 10))
datos_box <- data.frame(grupo = grupo, valores = valores)

# Boxplot
ggplot(datos_box, aes(x = grupo, y = valores)) +
  geom_boxplot()
```



Explicación:

1. Se crea un data frame con una variable categórica (`grupo`) y una numérica (`valores`).
2. `ggplot(datos_box, aes(x = grupo, y = valores))` mapea el grupo al eje X y los valores al eje Y.
3. `geom_boxplot()` genera los diagramas de caja para cada grupo.

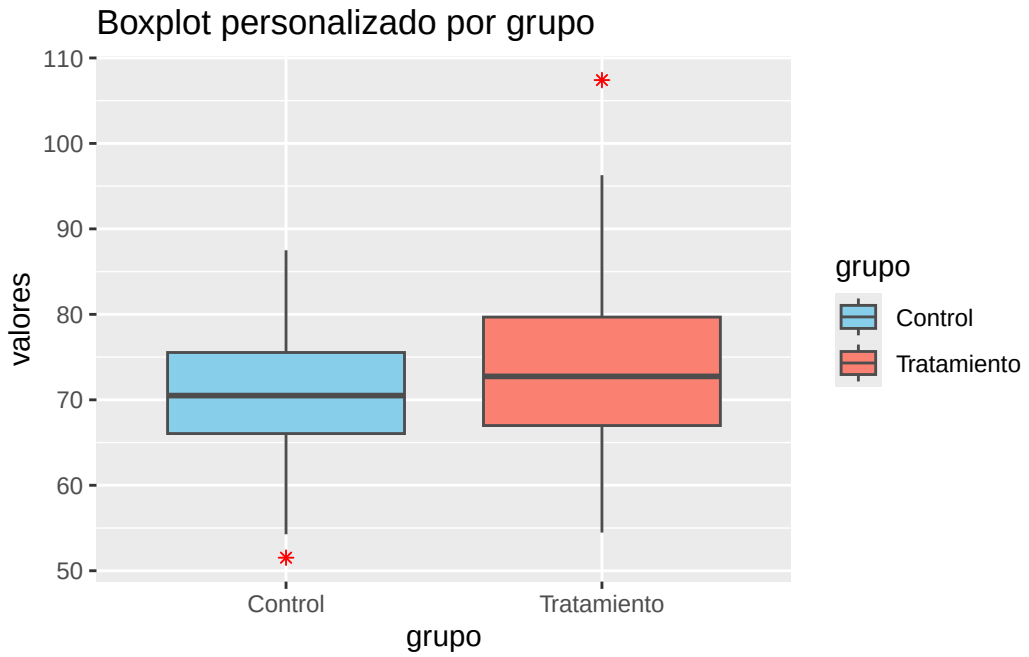
14.5 Personalización de gráficos en ggplot2

La personalización es una de las fortalezas principales de **ggplot2**, permitiendo adaptar cada gráfico a los estándares de comunicación científica, a las necesidades de la audiencia y a los requisitos de publicaciones profesionales. El flujo de trabajo de personalización en ggplot2 es progresivo y modular: cada aspecto visual puede ajustarse mediante capas adicionales o argumentos específicos, lo que facilita la reproducibilidad y la claridad en la presentación de resultados (Wickham, 2016).

14.5.1 Modificación de colores y escalas

El control de los colores es fundamental para mejorar la interpretación, la accesibilidad y la estética de los gráficos. ggplot2 permite modificar los colores de los elementos gráficos tanto de forma automática como manual, utilizando funciones de escala específicas. Para variables categóricas, se emplea `scale_fill_manual()` (relleno) o `scale_color_manual()` (bordes, líneas y puntos). Para variables continuas, existen escalas como `scale_fill_gradient()` o `scale_color_gradient()`, que permiten definir paletas de colores personalizadas o predefinidas.

```r
# Simulación de los datos
set.seed(123)
grupo <- factor(rep(c("Control", "Tratamiento"), each = 100))
valores <- c(rnorm(100, 70, 8), rnorm(100, 75, 10))
datos_box <- data.frame(grupo = grupo, valores = valores)

# Ejemplo de personalización de colores en un boxplot
ggplot(datos_box, aes(x = grupo, y = valores, fill = grupo)) +
  geom_boxplot(color = "gray30",
               outlier.colour = "red",
               outlier.shape = 8) +
  scale_fill_manual(values = c("skyblue", "salmon")) +
  scale_color_manual(values = c("gray30", "gray30")) +
  labs(title = "Boxplot personalizado por grupo")
```

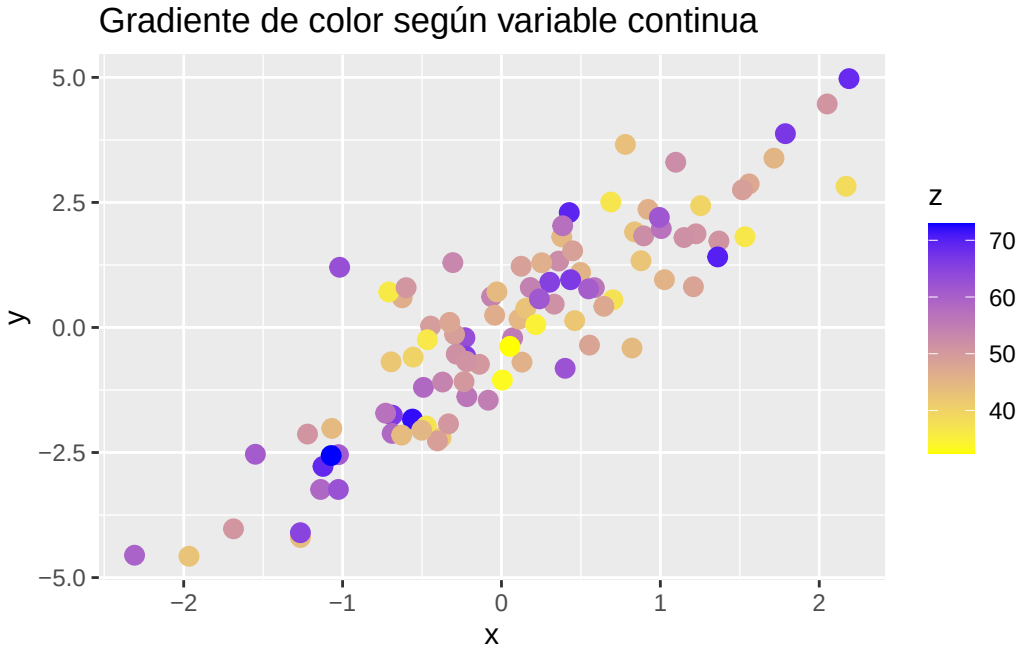


Explicación:

1. Se crea un data frame con variables categórica y numérica.
2. `aes(fill = grupo)` mapea el color de relleno a la variable de grupo.
3. `geom_boxplot()` permite personalizar el color del borde y el estilo de los valores atípicos.
4. `scale_fill_manual()` asigna colores específicos a cada grupo.
5. `scale_color_manual()` ajusta el color del borde de las cajas.

Para variables continuas, se puede utilizar una escala de gradiente:

```r
# Simulación de los datos
set.seed(123)
x <- rnorm(100)
y <- 2 * x + rnorm(100)
z <- rnorm(100, mean = 50, sd = 10)
datos_disp <- data.frame(x = x, y = y, z = z)
# Ejemplo de gradiente de color en un gráfico de dispersión
ggplot(datos_disp, aes(x = x, y = y, color = z)) +
  geom_point(size = 3) +
  scale_color_gradient(low = "yellow", high = "blue") +
  labs(title = "Gradiente de color según variable continua")
```



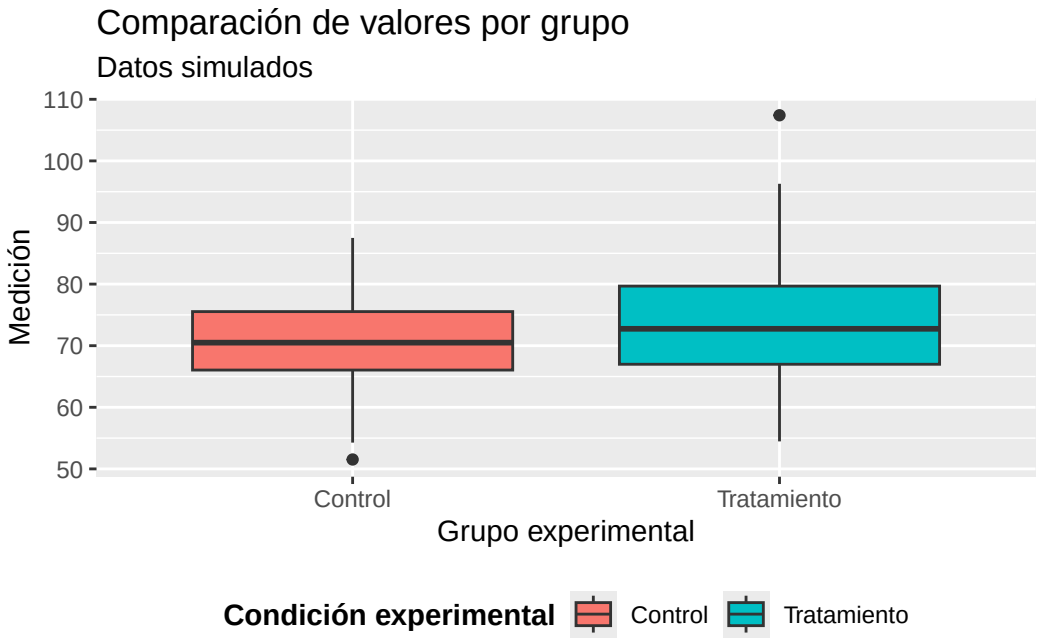
Explicación:

1. Se crea un data frame con dos variables numéricas y una variable adicional para el color.
2. `aes(color = z)` mapea la variable continua al color de los puntos.
3. `scale_color_gradient()` define los colores mínimo y máximo del gradiente.

14.5.2 Etiquetas, títulos y leyendas

La función `labs()` es la herramienta principal para añadir y personalizar títulos, subtítulos, etiquetas de ejes y leyendas. Una correcta rotulación es esencial para la interpretación y la comunicación efectiva de los resultados. Además, la posición y el formato de la leyenda pueden ajustarse mediante argumentos en la función `theme()`.

```r
# Ejemplo de personalización de etiquetas y leyendas
ggplot(datos_box, aes(x = grupo, y = valores, fill = grupo)) +
  geom_boxplot() +
  labs(
    title = "Comparación de valores por grupo",
    subtitle = "Datos simulados",
    x = "Grupo experimental",
    y = "Medición",
    fill = "Condición experimental"
  ) +
  theme(legend.position = "bottom",
        legend.title = element_text(face = "bold"))
```



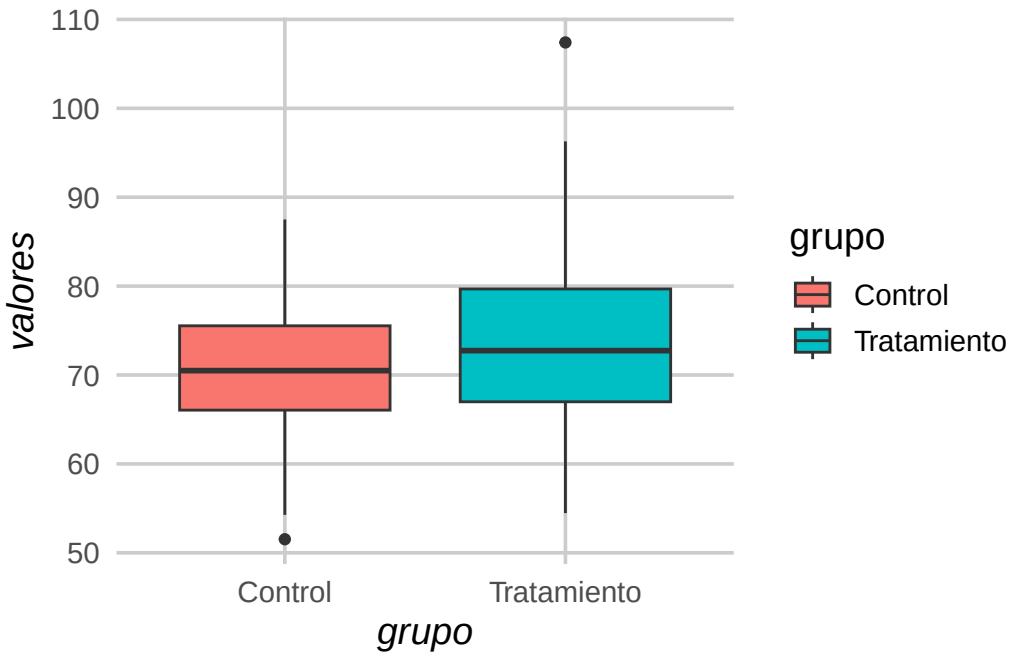
Explicación:

1. `labs()` define el título, subtítulo, etiquetas de ejes y el texto de la leyenda.
2. `theme(legend.position = "bottom")` coloca la leyenda debajo del gráfico.
3. `legend.title = element_text(face = "bold")` resalta el título de la leyenda.

14.5.3 Aplicación y personalización de temas

Los temas en ggplot2 controlan la apariencia global del gráfico, incluyendo el fondo, las fuentes, las líneas de cuadrícula y otros elementos estéticos. Existen temas predefinidos como `theme_minimal()`, `theme_classic()`, `theme_bw()`, y `theme_light()`, que pueden ser utilizados directamente o modificados mediante la función `theme()` para ajustar detalles específicos. La personalización de temas es clave para adaptar los gráficos a los estándares de publicaciones científicas y presentaciones profesionales (Tufte, 2001; Wickham, 2016).

```r
# Ejemplo de aplicación y ajuste de un tema
ggplot(datos_box, aes(x = grupo, y = valores, fill = grupo)) +
  geom_boxplot() +
  theme_minimal(base_size = 14) +
  theme(
    plot.title = element_text(face = "bold", color = "navy", size = 18),
    plot.subtitle = element_text(face = "italic", color = "gray40"),
    axis.title = element_text(face = "italic", size = 14),
    axis.text = element_text(color = "gray30"),
    panel.grid.major = element_line(color = "gray80"),
    panel.grid.minor = element_blank(),
    legend.position = "right"
  )
```



Explicación:

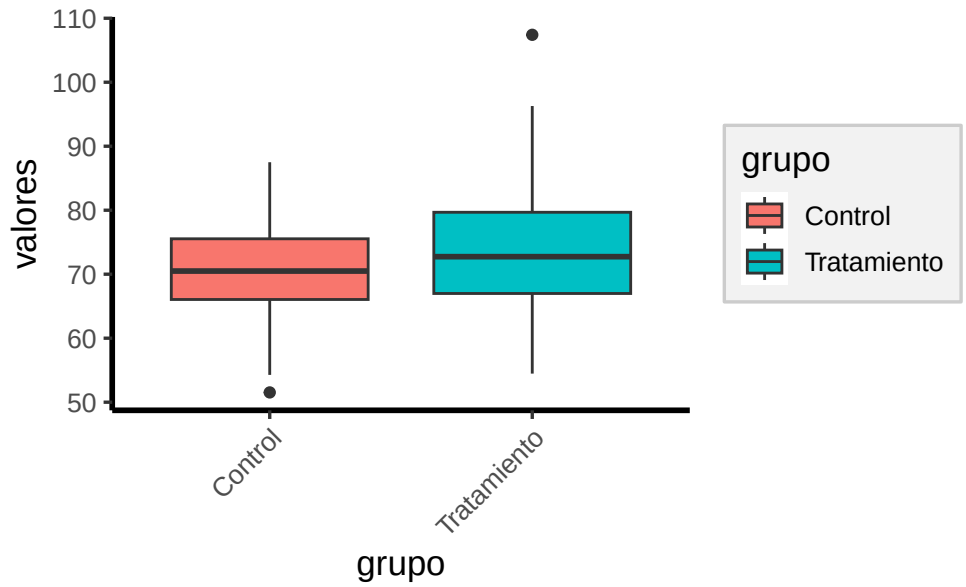
1. Se aplica el tema `theme_minimal()` para un estilo limpio y profesional.
2. `plot.title` y `plot.subtitle` ajustan el estilo y color de los títulos.
3. `axis.title` y `axis.text` modifican el estilo y tamaño de los textos de ejes.
4. `panel.grid.major` y `panel.grid.minor` controlan la visibilidad y color de las líneas de cuadrícula.
5. `legend.position` define la ubicación de la leyenda.

14.5.4 Personalización avanzada: fuentes, márgenes y elementos gráficos

ggplot2 permite un control detallado sobre elementos como el tipo y tamaño de fuente, los márgenes del gráfico, la orientación de las etiquetas y la visibilidad de los ejes. Estas

opciones avanzadas se gestionan principalmente a través de la función `theme()`, lo que permite adaptar el gráfico a los requisitos específicos de cada publicación o presentación.

```
# Ejemplo de personalización avanzada
ggplot(datos_box, aes(x = grupo, y = valores, fill = grupo)) +
  geom_boxplot() +
  theme_classic(base_size = 13) +
  theme(
    plot.margin = margin(20, 20, 20, 20),
    axis.text.x = element_text(angle = 45, hjust = 1),
    axis.line = element_line(color = "black", linewidth = 1),
    legend.background = element_rect(fill = "gray95", color = "gray80")
  )
```



Explicación:

1. `plot.margin` ajusta los márgenes del gráfico.
2. `axis.text.x` rota las etiquetas del eje X para mejorar la legibilidad.
3. `axis.line` resalta los ejes con líneas más gruesas y visibles.
4. `legend.background` modifica el fondo de la leyenda.

La personalización progresiva y modular de los gráficos en ggplot2 permite adaptar cada visualización a los estándares de comunicación científica y a las necesidades específicas de cada proyecto, garantizando resultados reproducibles y de alta calidad (Wickham, 2016).

14.6 Uso de facetas para comparación de grupos en ggplot2

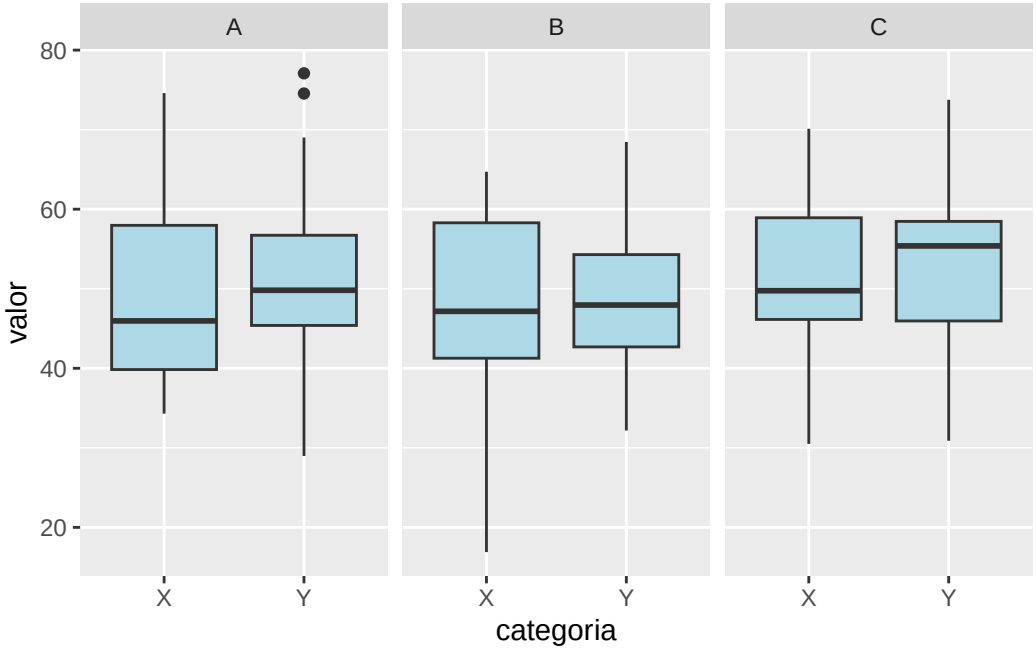
Las facetas en **ggplot2** permiten dividir un gráfico en múltiples paneles según los valores de una o más variables categóricas, facilitando la comparación visual entre subgrupos o condiciones experimentales. Esta funcionalidad es especialmente útil en el análisis exploratorio de datos, ya que permite identificar patrones, diferencias y tendencias específicas en cada grupo sin perder la coherencia visual y la escala común entre paneles (Wickham, 2016).

El flujo de trabajo para el uso de facetas en ggplot2 consiste en preparar los datos con las variables categóricas de interés, construir el gráfico base y añadir la capa de facetas mediante las funciones `facet_wrap()` o `facet_grid()`. Estas funciones ofrecen flexibilidad para organizar los paneles en filas, columnas o matrices, y permiten personalizar etiquetas, escalas y disposición de los subgráficos.

14.6.1 Facetado simple con facet_wrap()

La función `facet_wrap()` divide el gráfico en paneles independientes según los valores de una sola variable categórica. Los paneles se organizan automáticamente en una cuadrícula, lo que resulta útil para comparar múltiples niveles de una variable.

```
# Simulación de datos con una variable de facetas
set.seed(123)
grupo <- sample(c("A", "B", "C"), size = 150, replace = TRUE)
categoria <- sample(c("X", "Y"), size = 150, replace = TRUE)
valor <- rnorm(150, mean = 50, sd = 10)
datos_facet <- data.frame(grupo = grupo,
                          categoria = categoria,
                          valor = valor)

# Gráfico de dispersión facetado por grupo
ggplot(datos_facet, aes(x = categoria, y = valor)) +
  geom_boxplot(fill = "lightblue") +
  facet_wrap(~ grupo)
```



Explicación

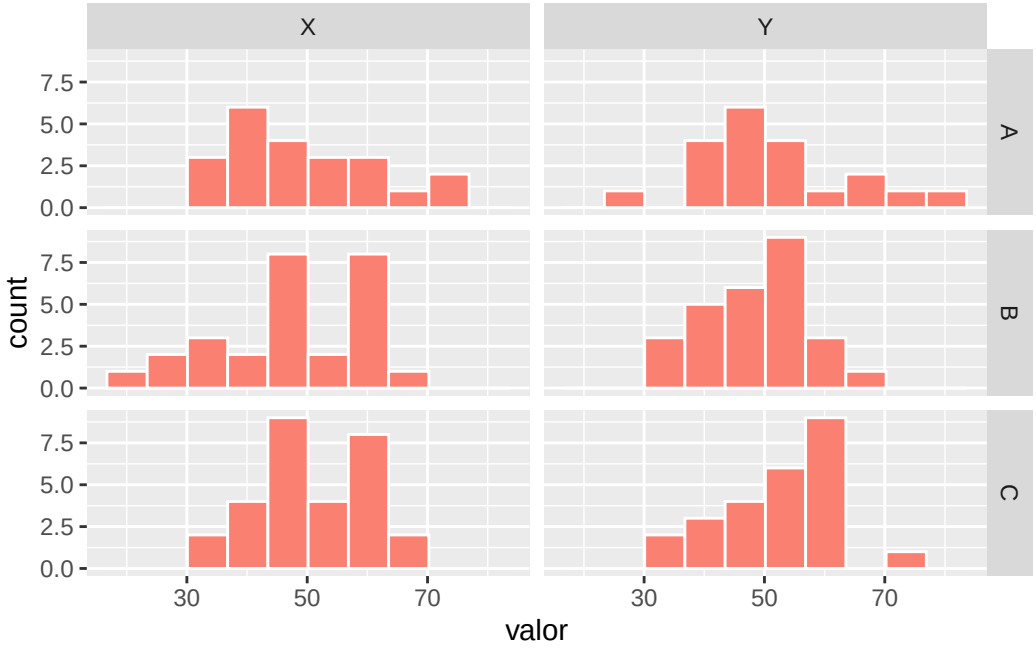
1. Se crea un data frame con una variable categórica para las facetas (`grupo`), una variable categórica para el eje X (`categoria`) y una variable numérica (`valor`).
2. `ggplot(datos_facet, aes(x = categoria, y = valor))` inicializa el gráfico.
3. `geom_boxplot()` representa los datos como diagramas de caja.
4. `facet_wrap(~ grupo)` divide el gráfico en paneles independientes para cada nivel de la variable `grupo`.

El argumento `nrow` o `ncol` en `facet_wrap()` permite controlar el número de filas o columnas de la cuadrícula de paneles.

14.6.2 Facetado múltiple con facet_grid()

La función `facet_grid()` permite crear una matriz de paneles utilizando dos variables categóricas, una para las filas y otra para las columnas. Esta organización es ideal para comparar simultáneamente los efectos de dos factores sobre la variable de interés.

```
# Gráfico de dispersión facetado por grupo y categoría
ggplot(datos_facet, aes(x = valor)) +
  geom_histogram(bins = 10, fill = "salmon", color = "white") +
  facet_grid(grupo ~ categoria)
```



Explicación:

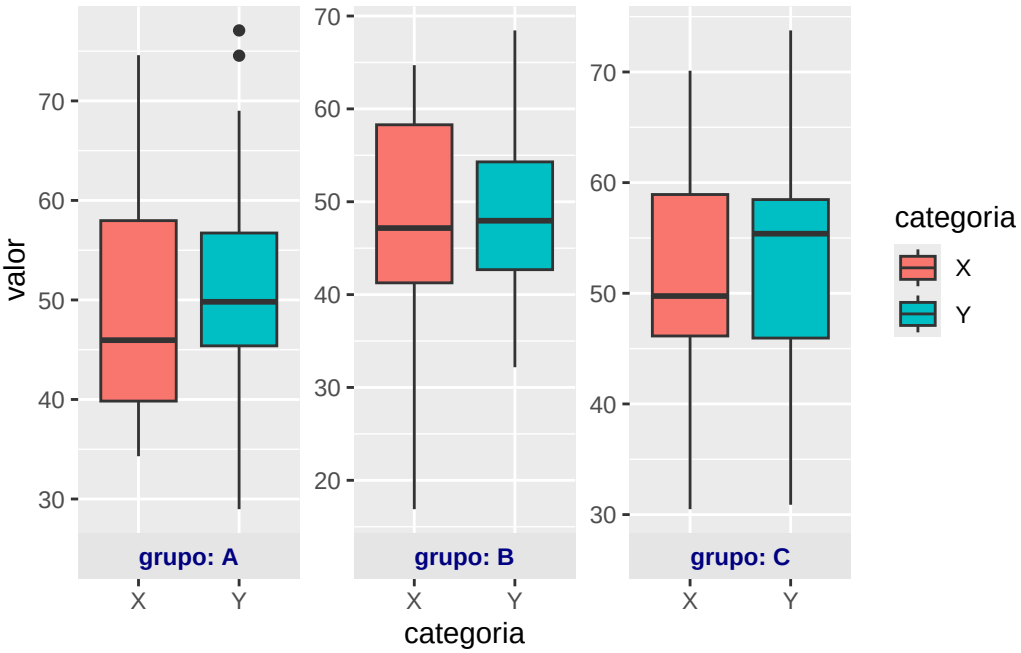
1. Se utiliza el mismo data frame con dos variables categóricas (`grupo` y `categoria`).
2. `geom_histogram()` representa la distribución de la variable numérica.
3. `facet_grid(grupo ~ categoria)` crea una matriz de paneles, donde las filas corresponden a los niveles de `grupo` y las columnas a los niveles de `categoria`.

El uso de `facet_grid()` es especialmente útil cuando se desea analizar la interacción entre dos factores y observar cómo varía la distribución de la variable numérica en cada combinación de niveles.

14.6.3 Personalización de facetas

ggplot2 permite personalizar las etiquetas de los paneles, la disposición de los subgráficos y la independencia de las escalas entre paneles. Los argumentos `labeller`, `scales` y `strip.position` en las funciones de facetas ofrecen un control detallado sobre la presentación final.

```
# Personalización avanzada de facetas
ggplot(datos_facet, aes(x = categoria, y = valor, fill = categoria)) +
  geom_boxplot() +
  facet_wrap(~ grupo, nrow = 1, labeller = label_both,
             strip.position = "bottom", scales = "free_y") +
  theme(strip.background = element_rect(fill = "gray90"),
        strip.text = element_text(face = "bold", color = "navy"))
```



Explicación:

1. `nrow = 1` organiza los paneles en una sola fila.
2. `labeller = label_both` muestra el nombre de la variable y el valor en la etiqueta del panel.
3. `strip.position = "bottom"` coloca las etiquetas de los paneles en la parte inferior.
4. `scales = "free_y"` permite que cada panel tenga su propia escala en el eje Y.
5. `theme()` personaliza el fondo y el texto de las etiquetas de los paneles.

La personalización de facetas es clave para mejorar la legibilidad y la interpretación de los gráficos comparativos, especialmente cuando se trabaja con conjuntos de datos complejos o con múltiples niveles de agrupamiento.

El uso de facetas en ggplot2, mediante `facet_wrap()` y `facet_grid()`, es una herramienta poderosa para la exploración visual y la comunicación de resultados en análisis estadístico, permitiendo comparar grupos de manera clara, eficiente y reproducible (Wickham, 2016).

14.7 Comparación entre ggplot2 y el sistema gráfico base de R

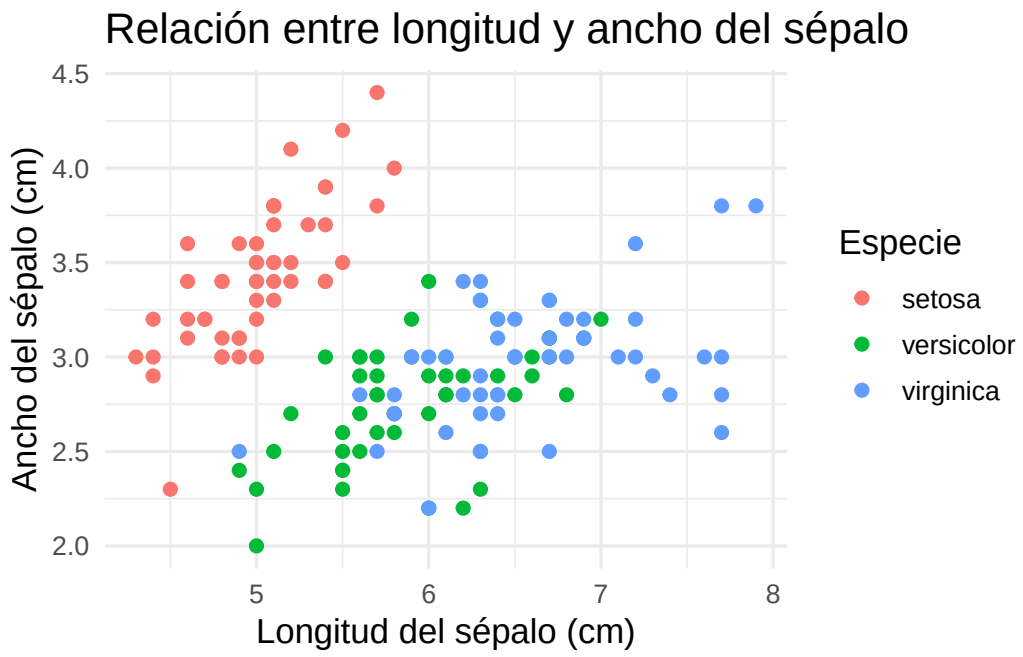
La comparación entre **ggplot2** y el sistema gráfico base de R es fundamental para quienes inician en la visualización de datos, ya que permite comprender las ventajas y limitaciones de cada enfoque. Para garantizar una comparación justa, se utilizará el conjunto de datos `iris`, uno de los más clásicos y ampliamente utilizados en la literatura estadística y en la enseñanza de R. Este conjunto contiene mediciones de longitud y ancho de sépalos y pétalos de tres especies de flores, y es ideal para ilustrar gráficos de dispersión con agrupamiento por especie (Anderson, 1935).

14.7.1 Sintaxis y filosofía

1. **ggplot2** se basa en la gramática de los gráficos, permitiendo construir visualizaciones complejas mediante la combinación modular de componentes. Su sintaxis es declarativa, lo que facilita la especificación de *qué* se desea visualizar.
2. **El sistema gráfico base** utiliza una sintaxis imperativa, donde cada paso debe ser definido explícitamente. Es más directo para gráficos simples, pero menos flexible para personalizaciones avanzadas.

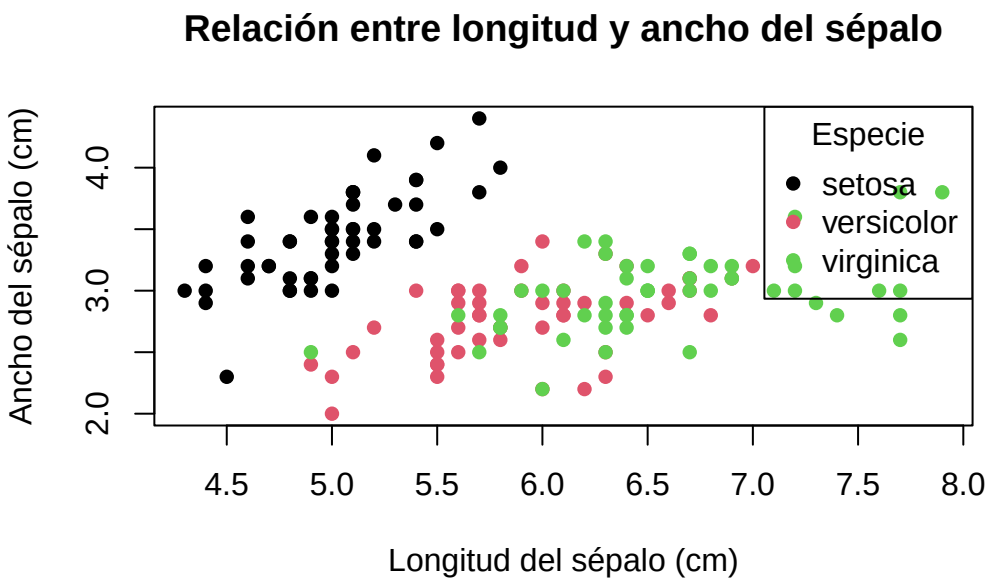
Ejemplo comparativo:

```
# 1. ggplot2
library(ggplot2)
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point(size = 2) +
  labs(
    title = "Relación entre longitud y ancho del sépalo",
    x = "Longitud del sépalo (cm)",
    y = "Ancho del sépalo (cm)",
    color = "Especie"
  ) +
  theme_minimal(base_size = 13)
```




```
# 2. Sistema gráfico base
plot(iris$Sepal.Length, iris$Sepal.Width,
     col = as.numeric(iris$Species),
     pch = 16,
     main = "Relación entre longitud y ancho del sépalo",
     xlab = "Longitud del sépalo (cm)",
```

```
     ylab = "Ancho del sépalo (cm)")
legend("topright",
       legend = levels(iris$Species),
       col = 1:3,
       pch = 16,
       title = "Especie")
```



Ambos gráficos utilizan el mismo conjunto de datos y presentan título, etiquetas de ejes y leyenda, lo que permite una comparación equitativa de la sintaxis y el resultado visual.

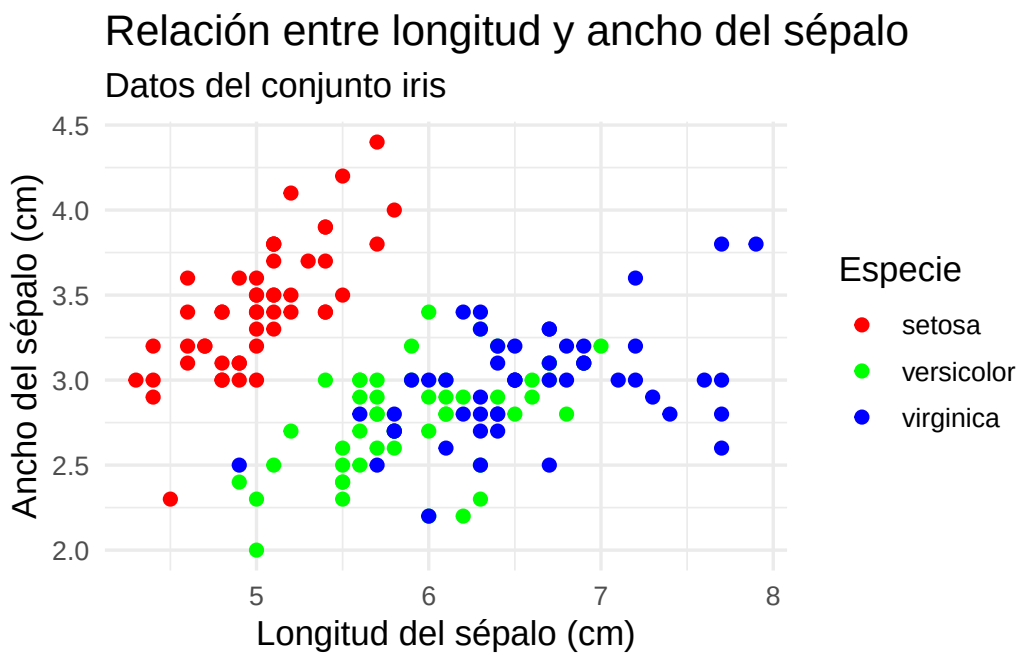
14.7.2 Flexibilidad y personalización

1. **ggplot2** permite personalizar cada elemento del gráfico de manera eficiente, utilizando capas y funciones específicas para colores, escalas, temas y leyendas.
2. **El sistema gráfico base** requiere argumentos adicionales y, en ocasiones, funciones externas para lograr el mismo nivel de personalización, lo que puede dificultar la reproducibilidad y la claridad del código.

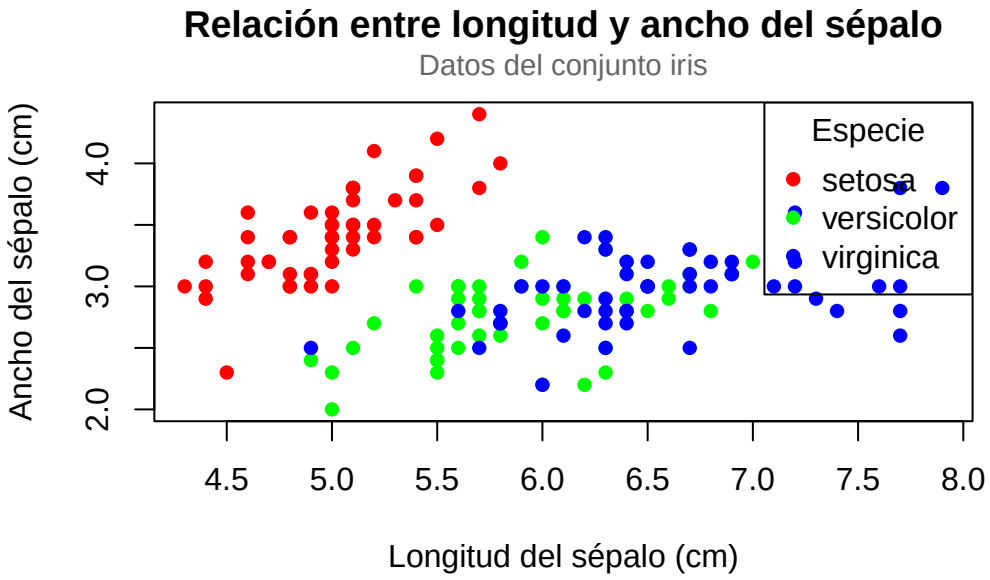
Ejemplo:

```
# 1. ggplot2
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  geom_point(size = 2) +
  scale_color_manual(values = c("red", "green", "blue")) +
  labs(
    title = "Relación entre longitud y ancho del sépalo",
    subtitle = "Datos del conjunto iris",
    x = "Longitud del sépalo (cm)",
```

```
    y = "Ancho del sépalo (cm)",
    color = "Especie"
  ) +
  theme_minimal(base_size = 13)
```




```
# 2. Sistema gráfico base
plot(iris$Sepal.Length, iris$Sepal.Width,
     col = c("red", "green", "blue")[as.numeric(iris$Species)],
     pch = 16,
     main = "Relación entre longitud y ancho del sépalo",
     xlab = "Longitud del sépalo (cm)",
     ylab = "Ancho del sépalo (cm)")
legend("topright",
       legend = levels(iris$Species),
       col = c("red", "green", "blue"),
       pch = 16,
       title = "Especie")
mtext("Datos del conjunto iris",
      side = 3, line = 0.5,
      cex = 0.9, col = "gray40")
```



Ambos gráficos incluyen personalización de colores, subtítulo, título, etiquetas de ejes y leyenda, asegurando una comparación justa.

14.7.3 Resumen comparativo

A continuación se presenta un cuadro comparativo que integra los aspectos más relevantes discutidos, facilitando la consulta rápida y la toma de decisiones informada:

Característica	ggplot2	Sistema gráfico base
Sintaxis	Declarativa, modular, basada en la gramática de los gráficos	Imperativa, secuencial
Flexibilidad	Alta, personalización eficiente y detallada	Limitada, requiere mayor esfuerzo
Estética	Moderna y profesional por defecto, fácil de mantener entre gráficos	Tradicional, requiere ajustes manuales
Flujo de trabajo	Modular, reproducible y documentado; facilita colaboración y revisión	Menos modular, menos reproducible
Reutilización	Alta, gracias a la estructura por capas y la posibilidad de guardar objetos gráficos	Baja, requiere repetir comandos y ajustes
Curva de aprendizaje	Inicialmente más pronunciada, pero ventajosa en proyectos complejos	Más sencilla para gráficos básicos
Integración	Excelente con el ecosistema tidyverse y flujos de trabajo modernos	Integración limitada con otros paquetes

Característica	ggplot2	Sistema gráfico base
Comunidad y recursos	Amplia documentación, comunidad activa y abundantes ejemplos	Documentación tradicional, menos recursos

En conclusión, **ggplot2** es la opción preferente para la elaboración de gráficos complejos, personalizados y de alta calidad, especialmente en contextos académicos y científicos donde la reproducibilidad, la estética y la flexibilidad son prioritarias. El sistema gráfico base de R, por su parte, sigue siendo útil para la creación rápida de gráficos exploratorios y para usuarios que requieren soluciones sencillas y directas, o que priorizan la inmediatez sobre la personalización (Murrell, 2018; Wickham, 2016).

14.8 Material de apoyo y paquetes adicionales recomendados

El aprendizaje y perfeccionamiento en la visualización de datos con `ggplot2` puede potenciarse mediante el acceso a recursos especializados y la integración de paquetes adicionales que amplían las capacidades del entorno gráfico en R. A continuación, se presentan recomendaciones de materiales de apoyo y herramientas complementarias, útiles tanto para quienes inician como para usuarios avanzados.

14.8.1 Recursos para profundizar en ggplot2

El ecosistema de R y `ggplot2` cuenta con una amplia variedad de materiales didácticos, manuales y cursos en línea que facilitan la adquisición de competencias en visualización de datos:

1. **Documentación oficial de `ggplot2`**: El sitio oficial del paquete ofrece una referencia completa de funciones, ejemplos y guías de uso, actualizadas conforme a las nuevas versiones (Wickham, 2016). La documentación es un recurso esencial para comprender a fondo las capacidades del paquete. https://ggplot2.tidyverse.org/
2. **Libros especializados**: ***ggplot2: Elegant Graphics for Data Analysis*** (Wickham, 2016) es la obra de referencia para comprender la filosofía y el uso avanzado del paquete. Este libro proporciona una base sólida para la creación de gráficos complejos y personalizados.
3. **Cursos y tutoriales en línea**: Plataformas como DataCamp, Coursera y edX ofrecen cursos interactivos sobre visualización de datos con R y `ggplot2`, que incluyen ejercicios prácticos y proyectos aplicados (Healy, 2018).

14.8.2 Paquetes adicionales recomendados

El uso de paquetes complementarios puede optimizar tareas específicas y ampliar las posibilidades de personalización y análisis gráfico:

1. **DataExplorer**: Facilita la exploración inicial de datos mediante la generación automática de reportes gráficos y estadísticos, permitiendo identificar patrones, valores atípicos y distribuciones de manera eficiente (Cui, 2020). Este paquete es especialmente útil para el análisis exploratorio de datos.
2. **ggthemes**: Proporciona una colección de temas predefinidos inspirados en estilos de publicaciones y medios reconocidos, lo que permite adaptar la estética de los gráficos a diferentes contextos profesionales. Los temas predefinidos pueden ahorrar tiempo y mejorar la apariencia de los gráficos.
3. **plotly**: Permite transformar gráficos estáticos de `ggplot2` en visualizaciones interactivas, ideales para presentaciones y análisis exploratorio en entornos web. La interactividad puede mejorar la comunicación de los resultados.
4. **ggrepel**: Mejora la legibilidad de los gráficos al evitar el solapamiento de etiquetas, especialmente útil en gráficos de dispersión con muchos puntos o etiquetas extensas. Evitar el solapamiento de etiquetas es crucial para la claridad visual.
5. **viridis**: Ofrece paletas de colores perceptualmente uniformes y accesibles para personas con daltonismo, recomendadas para mapas de calor y representaciones de datos continuos. El uso de paletas de colores accesibles es importante para la inclusión.

14.8.3 Consideraciones finales

La visualización de datos es un campo en constante evolución. Se recomienda explorar de manera continua nuevas herramientas, técnicas y buenas prácticas, así como participar en comunidades y foros especializados para resolver dudas y compartir experiencias. La integración de `ggplot2` con otros paquetes del ecosistema tidyverse y herramientas adicionales permite abordar desafíos complejos de visualización de manera eficiente, reproducible y profesional (Wickham et al., 2019).

Capítulo V

Gestión y Exportación de resultados

15 Introducción a la Gestión de Proyectos en R

La gestión de proyectos en R es fundamental para mantener el orden, la claridad y la eficiencia en el análisis estadístico de datos, incluso en proyectos de pequeña escala. La adopción de buenas prácticas desde el inicio previene errores, facilita la revisión del trabajo y mejora la comunicación de los resultados, tanto para el usuario principal como para otros colaboradores o revisores (Wickham & Grolemund, 2017).

15.1 Organización Básica de Proyectos Simples en R

En proyectos introductorios, donde el análisis se realiza a partir de una única base de datos y el flujo de trabajo es lineal, se recomienda centralizar todos los elementos del proyecto en una sola carpeta. Esta carpeta debe contener:

1. El archivo de datos (por ejemplo, un archivo CSV o Excel).
2. El archivo del proyecto de RStudio (extensión .Rproj).
3. El script de análisis en R (por ejemplo, analisis.R).
4. Los resultados exportados, como gráficos (PNG, PDF) y tablas (CSV, Excel).

Para evitar confusiones y facilitar la trazabilidad, se recomienda utilizar nombres de archivos descriptivos, en minúsculas, sin espacios ni símbolos especiales. Por ejemplo: `datos_clientes.csv`, `analisis_regresion.R`, `resultados_graficos.pdf`.

15.2 Organización Avanzada: Estructura de Directorios en Proyectos Complejos

En proyectos de mayor envergadura, que involucran múltiples fuentes de datos, análisis diversos y colaboración entre varios usuarios, es recomendable implementar una estructura de directorios jerárquica. Esta organización permite separar claramente los datos crudos, los datos procesados, los scripts, los resultados y la documentación, facilitando la escalabilidad y el mantenimiento del proyecto (Wilson et al., 2017).

Ejemplo de estructura recomendada para proyectos grandes:

```
proyecto/
  datos/
     raw/          # Datos originales
     processed/    # Datos procesados
  scripts/         # Scripts de análisis
  resultados/      # Salidas y gráficos
  docs/            # Documentación y reportes
  README.md        # Descripción general del proyecto
```

Esta estructura está ampliamente recomendada en la literatura sobre gestión de proyectos en ciencia de datos, como se detalla en el manual de Wilson et al. (2017), que enfatiza la importancia de la organización para la reproducibilidad y la colaboración efectiva.

15.3 Uso de RStudio Projects para la Gestión Eficiente

RStudio Projects es una herramienta integrada en RStudio que facilita la gestión de proyectos, incluso en análisis simples. Al crear un proyecto, se genera un archivo `.Rproj` que define el directorio de trabajo y centraliza todos los archivos relacionados. Esto asegura que el entorno de trabajo sea siempre el correcto y evita errores al cargar o guardar archivos. Para crear un proyecto, seleccione "File > New Project", elija "New Directory" y asigne un nombre y ubicación a la carpeta. Todos los archivos del análisis deben guardarse en esa carpeta para mantener la organización y la reproducibilidad (Wickham & Grolemund, 2017).

15.4 Principios de Reproducibilidad y Documentación

La reproducibilidad es un principio esencial en el análisis estadístico. Consiste en la capacidad de repetir un análisis y obtener los mismos resultados utilizando los mismos datos y scripts. Para lograrlo, es fundamental mantener todos los archivos del proyecto juntos y documentar cada paso del proceso. Se recomienda:

1. Utilizar scripts bien comentados, explicando cada parte del análisis.
2. Incluir los datos originales en la carpeta del proyecto.
3. Exportar los resultados en formatos accesibles y guardarlos en la misma carpeta.
4. Utilizar el archivo `.Rproj` para centralizar el entorno de trabajo.
5. Agregar comentarios en el script que expliquen el propósito de cada sección y las decisiones tomadas.

Esta documentación facilita la revisión, el aprendizaje y la colaboración, incluso en proyectos individuales (Wickham & Grolemund, 2017).

16 Exportación de Resultados de Análisis en R

La exportación de resultados constituye una etapa fundamental en el análisis estadístico de datos, ya que permite almacenar y compartir los productos del análisis, como gráficos y tablas, para su posterior utilización en informes, presentaciones o análisis adicionales. La correcta elección del formato de exportación garantiza la accesibilidad, reutilización y compatibilidad de los resultados con otras herramientas y plataformas (R Core Team, 2023; Wickham, 2016). Además, una exportación adecuada contribuye a la reproducibilidad y trazabilidad de los análisis, aspectos cruciales en la ciencia de datos moderna (Gentleman & Temple Lang, 2007; National Academies of Sciences, Engineering, and Medicine, 2019).

En el contexto del análisis estadístico clásico, la exportación de resultados facilita la comunicación de hallazgos y la integración de los mismos en documentos científicos, reportes técnicos o presentaciones. R ofrece funciones específicas para exportar tanto gráficos como tablas de datos en los formatos más utilizados en la práctica profesional y académica, asegurando la calidad y la fidelidad de la información exportada (R Core Team, 2023).

16.1 Exportación de gráficos: formatos PNG y PDF

La exportación de gráficos es fundamental para documentar visualmente los resultados del análisis. En R, la función `ggsave()` del paquete `ggplot2` permite guardar gráficos en diversos formatos, siendo PNG y PDF los más empleados en la estadística clásica. Según Wickham (2016), esta función ofrece flexibilidad y control sobre la calidad y el formato de los gráficos exportados.

16.1.1 Sintaxis general de ggsave()

La función `ggsave()` del paquete `ggplot2` permite guardar gráficos en diferentes formatos. Su sintaxis básica es:

```
ggsave(
  filename,
  plot = last_plot(),
  device = NULL,
  path = NULL,
  scale = 1,
  width = NA,
  height = NA,
  units = c("in", "cm", "mm"),
```

```
  dpi = 300,
  limitsize = TRUE
)
```

A continuación, se describen los argumentos principales de la función:

1. `filename` (obligatorio): Es el nombre del archivo de salida, incluyendo la extensión (por ejemplo, `"grafico.png"` o `"grafico.pdf"`). La extensión determina el formato del archivo. Este argumento es obligatorio ya que define el nombre y tipo del archivo a exportar.
2. `plot` (opcional): Permite especificar el objeto gráfico que se desea guardar. Si se omite, la función utiliza `last_plot()`, que guarda el último gráfico creado en la sesión de R. Este argumento puede omitirse si se desea guardar el último gráfico generado.
3. `device` (opcional): Indica el tipo de formato del archivo, como `"png"` o `"pdf"`. Si no se especifica, el formato se deduce automáticamente a partir de la extensión del archivo en `filename`. Por ejemplo, si `filename` es `"grafico.png"`, `device` se establecerá automáticamente como `"png"`.
4. `path` (opcional): Define el directorio donde se guardará el archivo. Si no se proporciona, el archivo se guarda en el directorio de trabajo actual. Es recomendable verificar el directorio de trabajo con `getwd()` antes de exportar.
5. `scale` (opcional): Ajusta el tamaño del gráfico multiplicando las dimensiones especificadas en `width` y `height` por el valor de `scale`. El valor predeterminado es 1 (tamaño original), lo que significa que el gráfico se guarda con las dimensiones especificadas en `width` y `height` sin modificar.
6. `width` y `height` (opcionales): Determinan el ancho y la altura del gráfico en las unidades especificadas por `units`. Si no se definen, se usan las dimensiones predeterminadas, que varían según el dispositivo de salida.
7. `units` (opcional): Especifica las unidades de medida para `width` y `height`. Puede ser `"in"` (pulgadas), `"cm"` (centímetros) o `"mm"` (milímetros). Si no se especifica, el valor predeterminado es `"in"` (pulgadas).
8. `dpi` (opcional): Define la resolución del gráfico en puntos por pulgada, relevante para formatos rasterizados como PNG. El valor predeterminado es 300, adecuado para impresión. Este argumento no es relevante para formatos vectoriales como PDF, ya que estos formatos no tienen una resolución fija.
9. `limitsize` (opcional): Controla si se permite guardar gráficos con dimensiones muy grandes (mayores a 50 pulgadas). Si está en `TRUE`, se genera un error al intentar guardar gráficos excesivamente grandes. El valor predeterminado es `TRUE`.

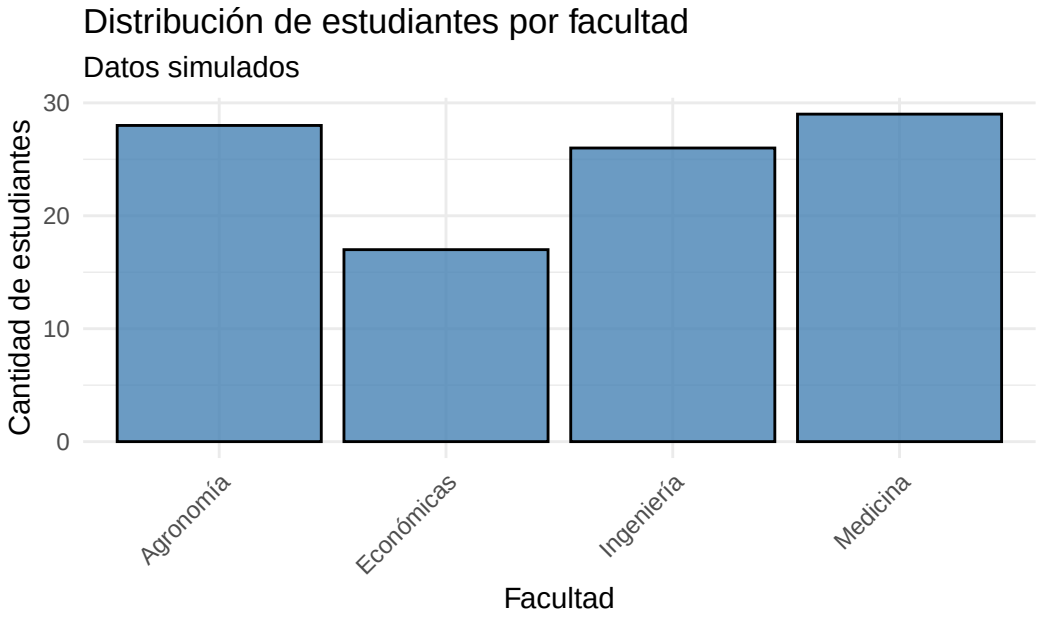
En resumen, los argumentos `plot`, `device`, `path`, `scale`, `width`, `height`, `units`, `dpi` y `limitsize` pueden omitirse si se desea utilizar sus valores por defecto. Sin embargo, es fundamental especificar `filename` para definir el nombre y el formato del archivo de salida.

Esta explicación permite comprender tanto la estructura general de la función como el propósito de cada argumento, facilitando su uso correcto en la exportación de gráficos en R (Wickham, 2016).

16.1.2 Ejemplo práctico: creación y exportación de un gráfico

Supóngase que se desea crear y exportar un gráfico de barras que represente la distribución de estudiantes por facultad. En este ejemplo, se simularán los datos para ilustrar el proceso.

```
# Cargar el paquete tidyverse, que incluye ggplot2
## Permite el acceso a funciones de manipulación y visualización de datos
if (!require("tidyverse")) install.packages("tidyverse")

# Simular datos de ejemplo
set.seed(123) # Para reproducibilidad
facultades <- c("Agronomía", "Ingeniería", "Medicina", "Económicas")
datos <- data.frame(
  ## Simula 100 estudiantes asignados a facultades
  FACULTAD = sample(facultades, 100, replace = TRUE)
)

# Crear un gráfico de barras
## Especifica los datos y la variable a graficar
mi_grafico <- ggplot(data = datos, aes(x = FACULTAD)) +
  ## Genera barras con color y transparencia
  geom_bar(fill = "steelblue", color = "black", alpha = 0.8) +
  labs(
    title = "Distribución de estudiantes por facultad",
    subtitle = "Datos simulados",
    x = "Facultad",
    y = "Cantidad de estudiantes",
    caption = "Fuente: Simulación"
  ) + # Añade etiquetas y título al gráfico
  theme_minimal() + # Aplica un tema visual sencillo
  theme(
    # Rota las etiquetas del eje x para mejor legibilidad
    axis.text.x = element_text(angle = 45, hjust = 1)
  )

mi_grafico # Muestra el gráfico
```



Fuente: Simulación

Guardar el gráfico en formato PNG

```
# Guardar el gráfico en formato PNG con dimensiones de 8x6 pulgadas
ggsave(
  filename = "grafico_simulado.png", # Nombre del archivo de salida
  plot = mi_grafico, # Objeto gráfico a guardar
  width = 8, # Ancho en pulgadas
  height = 6, # Alto en pulgadas
  dpi = 300 # Resolución adecuada para impresión
)
```

En este ejemplo, el archivo `"grafico_simulado.png"` se guardará en el directorio de trabajo actual, con alta calidad para impresión o presentaciones digitales.

Guardar el gráfico en formato PDF

```
# Guardar el gráfico en formato PDF con dimensiones de 8x6 pulgadas
ggsave(
  filename = "grafico_simulado.pdf", # Nombre del archivo de salida
  plot = mi_grafico, # Objeto gráfico a guardar
  width = 8, # Ancho en pulgadas
  height = 6 # Alto en pulgadas
  # No es necesario especificar dpi, ya que PDF es un formato vectorial
)
```

El formato PDF es ideal para informes y publicaciones científicas, ya que permite escalar el gráfico sin pérdida de calidad (Wickham, 2016).

16.2 Exportación de tablas de datos: formatos CSV y Excel

La exportación de tablas de datos es una etapa esencial en el flujo de trabajo estadístico, ya que permite compartir información, documentar resultados y facilitar análisis adicionales en otras herramientas. Los formatos CSV y Excel son los más empleados en la práctica profesional y académica debido a su amplia compatibilidad y facilidad de uso (R Core Team, 2023).

16.2.1 Exportar a CSV con write.csv()

El formato CSV (Comma Separated Values) es ampliamente utilizado por su compatibilidad con programas de hojas de cálculo y software estadístico. En R, la función `write.csv()` permite exportar un data frame o matriz a un archivo de texto plano en este formato (R Core Team, 2023).

Sintaxis general de `write.csv()`:

```
write.csv(
  x,
  file,
  row.names = TRUE,
  na = "NA",
  fileEncoding = "",
)
```

Explicación de los argumentos principales:

1. **`x` (obligatorio):** Es el objeto de datos que se desea exportar, generalmente un data frame o una matriz. Este argumento es imprescindible, ya que define el contenido del archivo a exportar (R Core Team, 2023).
2. **`file` (obligatorio):** Especifica el nombre del archivo de salida, incluyendo la extensión `.csv`. El archivo se guardará en el directorio de trabajo actual, a menos que se indique una ruta diferente. Es fundamental definir este argumento para que la función se ejecute correctamente (R Core Team, 2023).
3. **`row.names` (opcional):** Indica si se deben incluir los nombres de las filas como una columna adicional en el archivo exportado. El valor predeterminado es `TRUE`, pero es común establecerlo en `FALSE` para evitar agregar una columna innecesaria. Si no se especifica, se utilizará el valor por defecto (R Core Team, 2023).
4. **`na` (opcional):** Define la cadena de texto que se utilizará para representar los valores faltantes (`NA`) en el archivo exportado. El valor predeterminado es `"NA"`, lo que garantiza la identificación de datos ausentes en otros programas (R Core Team, 2023).
5. **`fileEncoding` (opcional):** Permite especificar la codificación del archivo de salida, lo cual es útil para asegurar la compatibilidad con otros sistemas operativos o programas. El valor predeterminado es una cadena vacía (`""`), lo que significa que se utiliza la codificación por defecto del sistema (R Core Team, 2023).

En síntesis, los argumentos `x` y `file` son obligatorios, mientras que `row.names`, `na` y `fileEncoding` pueden omitirse si se desea utilizar sus valores por defecto, lo que simplifica la sintaxis para exportaciones estándar.

Ejemplo de exportación a CSV con datos simulados:

```
# Crear un data frame de ejemplo
mi_tabla <- data.frame(
  Nombre = c("Ana", "Luis", "María"),
  Edad = c(25, 30, 22),
  Ciudad = c("Madrid", "Barcelona", "Valencia")
)

# Exportar el data frame a un archivo CSV
write.csv(
  x = mi_tabla,          # Objeto de datos a exportar
  file = "resultados.csv", # Nombre del archivo de salida
  row.names = FALSE      # No incluir los nombres de las filas
)
```

El archivo "resultados.csv" se guardará en el directorio de trabajo actual y podrá ser abierto en cualquier editor de texto o programa de hojas de cálculo (R Core Team, 2023).

16.2.2 Exportar a Excel con write_xlsx() del paquete writexl

El formato Excel (`.xlsx`) es ideal para compartir datos estructurados y aprovechar las funcionalidades avanzadas de hojas de cálculo. En R, la función `write_xlsx()` del paquete `writexl` permite exportar un data frame o una lista de data frames a un archivo Excel, facilitando la interoperabilidad con otros usuarios y sistemas (R Core Team, 2023).

Sintaxis general de `write_xlsx()`:

```
write_xlsx(
  x,           # Objeto de datos a exportar
  path,        # Nombre del archivo de salida
  col_names = TRUE,
  format_headers = TRUE
)
```

Explicación de los argumentos principales:

1. **`x` (obligatorio):** Es el objeto de datos a exportar, que puede ser un data frame o una lista de data frames. Si se proporciona una lista, cada data frame se guardará en una hoja diferente del archivo Excel (R Core Team, 2023).
2. **`path` (obligatorio):** Especifica el nombre del archivo de salida, incluyendo la extensión `.xlsx`. El archivo se guardará en el directorio de trabajo actual, a menos que se indique una ruta diferente. Este argumento es esencial para definir el destino del archivo (R Core Team, 2023).

3. **`col_names` (opcional):** Indica si se deben incluir los nombres de las columnas en la primera fila del archivo. El valor predeterminado es `TRUE`, lo que facilita la interpretación de los datos exportados (R Core Team, 2023).

4. **`format_headers` (opcional):** Determina si los encabezados de las columnas deben tener un formato especial (por ejemplo, negrita). El valor predeterminado es `TRUE`, lo que mejora la presentación visual del archivo (R Core Team, 2023).

Así, los argumentos `col_names` y `format_headers` pueden omitirse si se desea utilizar sus valores por defecto, mientras que `x` y `path` son obligatorios.

Ejemplo de exportación a Excel con datos simulados:

```
# Instalar y cargar el paquete writexl si no está disponible
if (!require("writexl")) install.packages("writexl")


# Crear un data frame de ejemplo
mi_tabla <- data.frame(
  Nombre = c("Ana", "Luis", "María"),
  Edad = c(25, 30, 22),
  Ciudad = c("Madrid", "Barcelona", "Valencia")
)

# Exportar el data frame a un archivo Excel
write_xlsx(
  x = mi_tabla,            # Objeto de datos a exportar
  path = "resultados.xlsx" # Nombre del archivo de salida
  # col_names y format_headers se mantienen en TRUE por defecto
)
```

El archivo "resultados.xlsx" se podrá abrir en Microsoft Excel o software compatible, permitiendo aprovechar las funcionalidades avanzadas de hojas de cálculo (R Core Team, 2023).

17 Introducción al control de versiones

El control de versiones y la colaboración en línea son prácticas cada vez más importantes en el análisis estadístico y la ciencia de datos. Git y GitHub permiten gestionar de manera eficiente los cambios en los archivos de un proyecto, compartir el trabajo con otros y mantener un historial completo de todas las modificaciones realizadas. Aunque estas herramientas pueden parecer complejas al principio, su integración con RStudio y su utilidad en proyectos de cualquier tamaño justifican su aprendizaje y uso desde etapas tempranas (Bryan, 2018).

17.1 ¿Qué es Git y por qué es importante?

Git es un sistema de control de versiones distribuido ampliamente adoptado en la comunidad científica y de desarrollo de software. Su principal fortaleza radica en la capacidad de gestionar de manera eficiente el historial de cambios, facilitar la colaboración entre múltiples usuarios y permitir la experimentación segura mediante la creación de ramas (branches) (Bryan, 2018; The Turing Way Community, 2023).

La integración de Git con entornos de desarrollo como RStudio simplifica su uso en proyectos de cualquier tamaño. Esta integración permite restaurar versiones anteriores de los archivos, identificar el origen y la motivación de las modificaciones, y experimentar con nuevas ideas sin comprometer el trabajo previo. Además, Git ayuda a reducir el riesgo de pérdida de información, ya que los archivos pueden ser restaurados a cualquier estado anterior (Bryan, 2018).

17.2 GitHub: Plataforma para la colaboración y la ciencia abierta

GitHub es una plataforma en línea que permite alojar repositorios de Git, compartir proyectos y colaborar con otros usuarios. Ofrece herramientas para la gestión de proyectos, seguimiento de incidencias (issues), revisión de código y documentación, lo que la convierte en un recurso esencial para la ciencia de datos reproducible y la investigación colaborativa (Grolemund & Wickham, 2017; The Turing Way Community, 2023).

En el contexto de proyectos de R, Git y GitHub permiten mantener un registro ordenado de los scripts, datos y resultados, facilitando la colaboración y la reproducibilidad del análisis. Estas herramientas son recomendadas desde las etapas iniciales de cualquier proyecto, ya que su adopción temprana contribuye a la integridad y transparencia del trabajo científico (Bryan, 2018).

17.3 Publicación y sincronización de proyectos de R en GitHub

La publicación de un proyecto de R en GitHub implica la creación de un repositorio en la plataforma y su sincronización con la carpeta local del proyecto. Este proceso puede realizarse tanto desde la interfaz gráfica de RStudio como mediante la línea de comandos, y es recomendable implementarlo desde el inicio del proyecto para garantizar la trazabilidad y la colaboración efectiva (Bryan, 2018).

17.3.1 Pasos para publicar un proyecto de R en GitHub

Antes de comenzar a utilizar Git y GitHub, es fundamental tener Git instalado en el ordenador. Git es el sistema de control de versiones que permite gestionar los cambios en los archivos del proyecto y sincronizarlos con el repositorio remoto en GitHub.

Para descargar e instalar Git, se puede acceder a la página oficial de Git (https://git-scm.com/downloads) y seguir las instrucciones correspondientes al sistema operativo utilizado

(Windows, macOS o Linux). Una vez instalado, Git estará disponible para ser utilizado desde la línea de comandos o a través de la interfaz de RStudio (Bryan, 2018).

1. **Creación de una cuenta y repositorio en GitHub:** El primer paso consiste en crear una cuenta personal en GitHub. Una vez registrado, se debe crear un nuevo repositorio, asignándole un nombre descriptivo y, opcionalmente, una breve descripción. Es posible elegir entre un repositorio público o privado, según las necesidades del proyecto.

2. **Configurar el nombre de usuario en Git:** En la terminal, se debe establecer el nombre de usuario global con el siguiente comando:

   ```
   git config --global user.name "nombre"
   ```

   Esto permite que Git asocie los cambios realizados con el usuario correspondiente (Bryan, 2018).

3. **Configurar el correo electrónico en Git:** También en la terminal, se debe definir el correo electrónico global con el comando:

   ```
   git config --global user.email "correo"
   ```

   Este correo debe coincidir con el registrado en GitHub para asegurar la correcta vinculación de los commits (Bryan, 2018).

4. **Inicialización de Git en la carpeta local del proyecto:** En la computadora local, se debe ubicar la carpeta del proyecto de R (que contiene el archivo `.Rproj`, los datos, los scripts y los resultados exportados). En RStudio, se activa el control de versiones desde el menú "Tools > Project Options > Git/SVN", seleccionando Git. Alternativamente, en la terminal, se ejecuta el comando:

   ```
   git init
   ```

   Esto crea una carpeta oculta llamada `.git` que permitirá a Git rastrear los cambios en los archivos del proyecto.

5. **Conexión del repositorio local con el repositorio remoto en GitHub:** Para vincular la carpeta local con el repositorio creado en GitHub, se debe copiar la URL del repositorio (por ejemplo, `https://github.com/usuario/repositorio.git`) y ejecutar el siguiente comando en la terminal:

   ```
   git remote add origin https://github.com/usuario/repositorio.git
   ```

6. **Agregado y confirmación de archivos:** Se agregan los archivos del proyecto al control de versiones con el comando:

   ```
   git add .
   ```

   El punto (`.`) indica que se agregarán todos los archivos de la carpeta.

7. **Comentar los cambios realizados al código:** Se realiza el primer registro de cambios (commit) con un mensaje descriptivo:

   ```
   git commit -m "Primer commit: subida inicial del proyecto"
   ```

8. **Crear la rama principal "main":** Para asegurarse de que la rama principal se llame "main", se ejecuta:

```
git branch -M main
```

   Esto es importante para mantener la compatibilidad con la configuración estándar de GitHub (Grolemund & Wickham, 2017).

9. **Subida de archivos al repositorio remoto:** Finalmente, los archivos se suben al repositorio remoto con el comando:

```
git push -u origin main
```

   Una vez completados estos pasos, el proyecto estará disponible en GitHub, permitiendo su consulta, descarga y colaboración (Bryan, 2018; Grolemund & Wickham, 2017). Cabe resaltar que los **pasos 2** y **3** únicamente se realizan la primera vez que se configura git en una computadora o si se desea cambiar de usuario en un dispositivo.

17.4 Modificación, seguimiento y colaboración en proyectos de GitHub

Una vez que el proyecto está alojado en GitHub, es posible continuar su desarrollo y mantener un registro detallado de todas las modificaciones. El flujo de trabajo básico consiste en realizar cambios en los archivos del proyecto, registrar estos cambios mediante commits con mensajes claros y específicos, y sincronizarlos con el repositorio remoto utilizando el comando `git push` (Bryan, 2018). A continuación se datalla este proceso:

17.4.1 Modificación, seguimiento y colaboración en proyectos de GitHub

1. **Sincronización con el repositorio remoto:** Antes de realizar cualquier cambio en el proyecto, se recomienda actualizar la copia local del repositorio para asegurarse de trabajar con la versión más reciente. Esto se logra ejecutando el siguiente comando en la terminal:

```
git pull
```

   Este comando descarga y fusiona los cambios realizados por otros colaboradores en el repositorio remoto, evitando conflictos y asegurando la coherencia del trabajo (Bryan, 2018).

2. **Realización de cambios en los archivos del proyecto:** Una vez sincronizado el repositorio local, se pueden modificar, agregar o eliminar archivos según las necesidades del proyecto. Estas modificaciones pueden incluir la edición de scripts, la incorporación de nuevos datos o la actualización de documentación.

3. **Preparación de los archivos modificados para el seguimiento:** Tras realizar los cambios, es necesario agregar los archivos modificados al área de preparación (staging area) mediante el comando:

```
git add .
```

El punto (.) indica que se incluirán todos los archivos modificados, nuevos o eliminados en el próximo commit.

4. **Registro de los cambios realizados:** Para documentar las modificaciones, se debe crear un commit con un mensaje claro y descriptivo que explique la naturaleza de los cambios. Esto se realiza con el comando:

```
git commit -m "Mensaje descriptivo de los cambios realizados"
```

La claridad y especificidad en los mensajes de commit facilitan la revisión y el seguimiento del historial del proyecto (Bryan, 2018).

5. **Sincronización de los cambios con el repositorio remoto:** Finalmente, para compartir los cambios con otros colaboradores y mantener actualizado el repositorio en línea, se utiliza el comando:

```
git push
```

Este comando sube los commits locales al repositorio remoto en GitHub, permitiendo la colaboración y el respaldo continuo del proyecto.

El uso sistemático de este flujo de trabajo garantiza que el proyecto esté siempre respaldado, documentado y disponible para la colaboración, promoviendo la transparencia y la reproducibilidad en el desarrollo científico y profesional (Gentleman & Temple Lang, 2007; National Academies of Sciences, Engineering, and Medicine, 2019; The Turing Way Community, 2023).

17.5 Importación y reutilización de repositorios de GitHub

La importación de repositorios de GitHub, conocida como "clonación", permite descargar una copia completa de un proyecto para trabajar localmente, modificarlo o adaptarlo a nuevas necesidades. Este procedimiento es útil tanto para uso personal como para la colaboración en proyectos de otros usuarios (Bryan, 2018).

17.5.1 Pasos para clonar un repositorio

1. **Obtener la URL del repositorio:** Se debe acceder a la página del proyecto en GitHub y copiar la URL del repositorio, esta se encuentra disponible mediante el botón "Code".

2. **Ejecutar el comando de clonación:** En la terminal de RStudio, se debe utilizar el siguiente comando, reemplazando la URL por la correspondiente al repositorio de interés:

```
git clone https://github.com/usuario/analisis_estadistico.git
```

Este comando crea una carpeta local que contiene todos los archivos del proyecto, así como su historial completo de versiones.

17.5.2 Trabajo local y sincronización de cambios

Una vez clonado el repositorio, es posible modificar los archivos, ejecutar scripts, agregar nuevos datos o exportar resultados. Si se dispone de los permisos necesarios (por ejemplo, en proyectos propios o en los que se tiene acceso de escritura), los cambios realizados pueden sincronizarse con el repositorio remoto utilizando los comandos `git add`, `git commit` y `git push`. En caso de no contar con permisos de escritura, los cambios pueden mantenerse localmente o bien proponer modificaciones a través de un "pull request".

17.5.3 Beneficios de la reutilización de repositorios

La clonación de repositorios facilita la reutilización de análisis y metodologías existentes, fomenta el aprendizaje a partir de proyectos desarrollados por otros usuarios y promueve la colaboración en equipo. Además, este proceso asegura la trazabilidad y la integridad del trabajo, ya que todo el historial de cambios queda registrado y disponible para su consulta y auditoría (Bryan, 2018; The Turing Way Community, 2023).

17.6 Recursos adicionales para el aprendizaje continuo

Para quienes deseen profundizar en el uso de Git y GitHub en el contexto de R y la ciencia de datos, se recomienda consultar los siguientes recursos:

1. **Libro recomendado: Happy Git and GitHub for the useR"** de Jenny Bryan (2018) es una guía completa y accesible, disponible de forma gratuita en línea, que cubre desde los conceptos básicos del control de versiones hasta técnicas avanzadas de colaboración y gestión de proyectos en GitHub. El libro está orientado específicamente a usuarios de R y ofrece ejemplos prácticos y actualizados para el análisis estadístico y la ciencia de datos (Bryan, 2018).
2. **Manual colaborativo: The Turing Way** (The Turing Way Community, 2023) es un manual colaborativo que aborda la reproducibilidad, la ética y la colaboración en la investigación científica. Este recurso proporciona información detallada sobre el uso de Git y GitHub en proyectos de ciencia abierta, con énfasis en las buenas prácticas y la gestión de datos.

Ambos recursos constituyen una base sólida para el aprendizaje continuo y la aplicación efectiva de Git y GitHub en proyectos de R, contribuyendo a la mejora de la reproducibilidad, la transparencia y la colaboración en la investigación científica (Gentleman & Temple Lang, 2007; National Academies of Sciences, Engineering, and Medicine, 2019).

Capítulo VI

Referencias

18 Referencias

Anderson, E. (1935). The irises of the Gaspe Peninsula. *Bulletin of the American Iris Society*, 59, 2–5.

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604), 452–454. https://doi.org/10.1038/533452a

Bryan, J. (2018). *Happy Git and GitHub for the useR*. https://happygitwithr.com/

Burnham, K. P., & Anderson, D. R. (2004). Multimodel inference: understanding AIC and BIC in model selection. *Sociological Methods & Research*, 33(2), 261-304.

Chang, W. (2018). *R graphics cookbook* (2nd ed.). O'Reilly Media.

Chambers, J. (2008). *Software for data analysis: Programming with R* (1st ed.). Springer. https://doi.org/10.1007/978-0-387-75936-4

Cleveland, W. S. (1993). *Visualizing Data*. Hobart Press.

Cui, B. (2023). *DataExplorer: Automate data exploration for complete preliminary analysis* (versión 0.8.3) [Paquete R]. CRAN. https://CRAN.R-project.org/package=DataExplorer

Field, A. (2013). *Discovering statistics using IBM SPSS statistics: and sex and drugs and rock'n'roll* (4th ed.). Sage.

Field, A. (2018). *Discovering statistics using R*. Sage.

Fisher, R. (1936). Iris [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76

Friendly, M. (2008). A brief history of data visualization. In *Handbook of Data Visualization* (pp. 15–56). Springer. https://doi.org/10.1007/978-3-540-33037-0_2

Gentleman, R., & Temple Lang, D. (2007). Statistical analyses and reproducible research. *Journal of Computational and Graphical Statistics*, 16(1), 1–23. https://doi.org/10.1198/106186007X178663

Grolemund, G., & Wickham, H. (2017). *R for data science*. O'Reilly Media. https://r4ds.had.co.nz/

Healy, K. (2018). *Data visualization: A practical introduction*. Princeton University Press.

Hernández, F., Usuga, O., & Mazo, M. (12 de agosto de 2024). *Modelos de regresión con R*. Github.io. https://fhernanb.github.io/libro_regresion/

Ihaka, R., & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3), 299–314. https://doi.org/10.1080/10618600.1996.10474713

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning: with applications in R*. Springer.

Kutner, M. H., Nachtsheim, C. J., Neter, J., & Li, W. (2005). *Applied linear statistical models* (5th ed.). McGraw-Hill, Irwin.

Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to linear regression analysis* (Vol. 821). John Wiley & Sons.

Moore, D. S., Notz, W. I., & Flinger, M. A. (2017). *The basic practice of statistics* (8th ed.). W. H. Freeman.

Murrell, P. (2018). *R graphics* (3rd ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9780429422768

National Academies of Sciences, Engineering, and Medicine. (2019). *Reproducibility and replicability in science*. National Academies Press. https://doi.org/10.17226/25303

Navarro, D. J. (2019). *Learning statistics with R: A tutorial for psychology students and other beginners* (versión 0.6). https://learningstatisticswithr.com

R Core Team. (2023). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.r-project.org/

Revelle, W. (2023). *psych: Procedures for psychological, psychometric, and personality research* (versión 2.3.6) [Paquete R]. CRAN. https://CRAN.R-project.org/package=psych

Rosales Castillo, J. M. (2005). Micropropagación de Calahuala Phlebodium psedoaureum (Cav.) Lellinger con tres tipos de explantes en diferentes medios de cultivo in vitro. Tesis Ing. Agr. Guatemala, Universidad de San Carlos de Guatemala, Facultad de Agronomía.

The Turing Way Community. (2023). *The Turing Way: A handbook for reproducible, ethical and collaborative research*. https://the-turing-way.netlify.app

Trujillo Sierra, E. (2022). Modelo de Regresión Lineal Múltiple - Salinidad. RStudio Pubs. Recuperado de: https://rstudio-pubs-static.s3.amazonaws.com/940966_d007915418ef41c7874f7316aa972543.html

Tufte, E. (2001). *The visual display of quantitative information* (2nd ed.). Graphics Press.

Tukey, J. W. (1977). *Exploratory data analysis*. Addison-Wesley.

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with S* (4th ed.). Springer. https://doi.org/10.1007/978-0-387-21706-2

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer. https://ggplot2.tidyverse.org

Wickham, H., Averick, M., Bryan, J., Chang, W., D'Agostino McGowan, L., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686. https://doi.org/10.21105/joss.01686

Wickham, H., François, R., Henry, L., & Müller, K. (2019). *dplyr: A grammar of data manipulation* (versión 1.1.2) [Paquete R]. CRAN. https://CRAN.R-project.org/package=dplyr

Wickham, H., & Grolemund, G. (2017). *R for data science: Import, tidy, transform, visualize, and model data*. O'Reilly Media. https://r4ds.had.co.nz

Wilkinson, L. (2005). *The grammar of graphics* (2nd ed.). Springer. https://doi.org/10.1007/0-387-28695-0

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., ... Mons, B. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3, 160018. https://doi.org/10.1038/sdata.2016.18

Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, A. J., & Teal, T. K. (2017). Good enough practices in scientific computing. *PLOS Computational Biology*, 13(6), e1005510. https://doi.org/10.1371/journal.pcbi.1005510

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). *R Markdown: The definitive guide* (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781138359444

Capítulo VII

Ejemplos de análisis estadístico

19 Estimación de parámetros de estadística descriptiva en R

La estadística descriptiva es una rama fundamental de la estadística que se dedica a resumir y presentar datos de manera informativa. A través de medidas como la media, la mediana y la moda, es posible obtener una comprensión inicial de las características principales de un conjunto de datos (Navarro, 2019). En esta sección, se ilustrará cómo calcular y presentar estos estadísticos descriptivos utilizando el lenguaje de programación R y el paquete `tidyverse`, que facilita la manipulación y visualización de datos (Wickham, 2019).

19.1 Base de datos

El conjunto de datos IRIS es uno de los conjuntos de datos más utilizados en la literatura de estadística y aprendizaje automático. Fue introducido por Ronald Fisher en 1936 y contiene mediciones de cuatro características morfológicas de flores de tres especies distintas de iris: *Iris setosa*, *Iris versicolor* e *Iris virginica*. Este dataset es ampliamente empleado para ilustrar técnicas de análisis estadístico y clasificación supervisada (Fisher, 1936).

Referencia del dataset: Fisher, R. (1936). Iris [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76

Acceso a recursos: El script completo con el ejemplo desarrollado y la base de datos IRIS pueden descargarse en el siguiente repositorio: https://github.com/Ludwing-MJ/Est_Desc_EJ.git

19.2 Configuración del Entorno de Trabajo

Antes de comenzar cualquier análisis, es fundamental configurar adecuadamente el entorno de trabajo. Esto implica instalar y cargar los paquetes necesarios, así como explorar y comprender la estructura del conjunto de datos que se utilizará. En esta sección, se detallarán los pasos para configurar el entorno de trabajo y realizar una exploración inicial del conjunto de datos `iris`.

19.2.1 Instalación y carga de paquetes necesarios

R cuenta con una amplia variedad de paquetes que facilitan la realización de análisis estadísticos y visualizaciones de datos. Para este manual, se utilizarán los siguientes paquetes:

1. **tidyverse:** Proporciona un conjunto de paquetes para la manipulación, transformación y visualización de datos, incluyendo `dplyr`, `ggplot2`, `tidyr`, entre otros (Wickham et al., 2019).
2. **psych:** Ofrece funciones para el análisis psicométrico y estadístico, incluyendo descripciones detalladas de los datos (Revelle, 2023).
3. **ggplot2:** Facilita la creación de gráficos de alta calidad (Wickham, 2016).
4. **DataExplorer:** Proporciona herramientas para la exploración automática de datos (Cui, 2023).

El siguiente código instala y carga estos paquetes, verificando primero si ya están instalados para evitar reinstalaciones innecesarias:

```
# Instalación y carga de paquetes necesarios
## Para manipulación de datos
if (!require(tidyverse)) install.packages("tidyverse")
## Para estadísticas descriptivas
if (!require(psych)) install.packages("psych")
## Para visualización de correlaciones
if (!require(corrplot)) install.packages("corrplot")
## Para gráficos avanzados
if (!require(ggplot2)) install.packages("ggplot2")
## Para exploración automática
if (!require(DataExplorer)) install.packages("DataExplorer")
```

19.2.2 Carga y exploración inicial del dataset

Una vez configurado el entorno de trabajo, es fundamental cargar y explorar el conjunto de datos que se utilizará. En este caso, se utilizará el conjunto de datos `iris`, que está incluido por defecto en R. Este conjunto de datos contiene información sobre las dimensiones de los sépalos y pétalos de tres especies de flores de iris (setosa, versicolor y virginica).

El siguiente código carga el conjunto de datos `iris` y muestra las primeras filas y la estructura de los datos:

```
# Cargar el dataset IRIS
data(iris)

# Visualizar las primeras filas
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

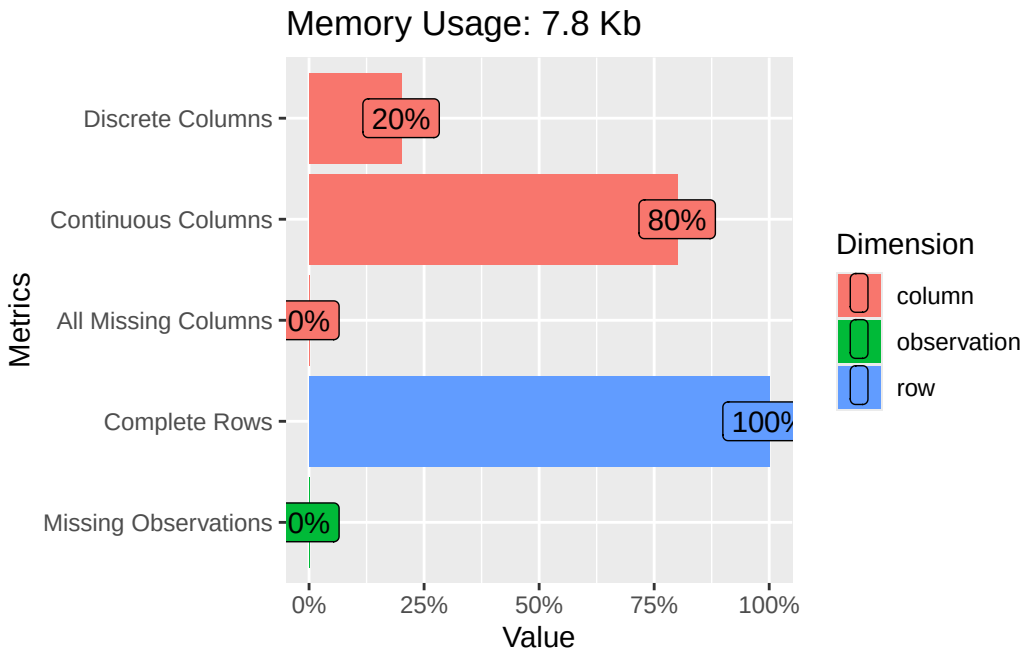
```r
# Explorar la estructura de los datos
str(iris)
```

```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

1. `head(iris)` proporciona una vista rápida de las primeras filas del conjunto de datos, lo que ayuda a detectar valores atípicos evidentes y familiarizarse con las variables.
2. `str(iris)` muestra la estructura del conjunto de datos, incluyendo el tipo de cada variable (numérico o factor) y los primeros valores de cada variable. Esto facilita la planificación del análisis y la identificación de posibles problemas con los datos.

Además de `head()` y `str()`, el paquete `DataExplorer` ofrece funciones para la exploración automática de datos, como `plot_intro()`, que genera un informe completo sobre las características del conjunto de datos.

```r
# Exploración automática con DataExplorer
DataExplorer::plot_intro(iris)
```



La función `plot_intro()` genera un informe que incluye información sobre el número de variables, el número de observaciones, el porcentaje de valores faltantes y el tipo de cada variable. Esto permite obtener una visión general del conjunto de datos de manera rápida y eficiente.

En resumen, la configuración del entorno de trabajo y la exploración inicial del conjunto de datos son pasos fundamentales para garantizar la calidad y validez del análisis. La

instalación y carga de los paquetes necesarios, así como la exploración de la estructura y características del conjunto de datos, permiten identificar posibles problemas y planificar el análisis de manera efectiva (Wickham & Grolemund, 2017).

19.3 Medidas de tendencia central

Las medidas de tendencia central son estadísticos que resumen el centro de un conjunto de datos. Las más comunes son la media, la mediana y la moda. Estas medidas proporcionan información valiosa sobre los valores típicos en una distribución y son fundamentales para comprender las características principales de los datos (Moore et al., 2017).

19.3.1 Media y mediana

La media es el promedio aritmético de los valores, mientras que la mediana es el valor que se encuentra en el centro de la distribución cuando los datos están ordenados. En R, se pueden calcular fácilmente con las funciones base `mean()` y `median()`.

```
# Media aritmética de la longitud del sépalo
mean(iris$Sepal.Length)
```

```
[1] 5.843333
```

```
# Mediana de la longitud del sépalo
median(iris$Sepal.Length)
```

```
[1] 5.8
```

La media de la longitud del sépalo es 5.84 cm y la mediana es 5.80 cm. La cercanía entre la media y la mediana sugiere que la distribución de la variable `Sepal.Length` es relativamente simétrica, es decir, no presenta una asimetría significativa. En distribuciones simétricas, la media y la mediana tienden a ser iguales, mientras que en distribuciones asimétricas, la media se desplaza hacia la cola más larga (James et al., 2013).

19.3.2 Cálculo de la moda

La moda es el valor que aparece con mayor frecuencia en un conjunto de datos. A diferencia de la media y la mediana, R no tiene una función base para calcular la moda. Por lo tanto, se define una función personalizada para calcular la moda, que maneja adecuadamente los valores faltantes (NA) y permite identificar múltiples modas en caso de empate.

```r
# Función para calcular la moda
moda <- function(x) {
  # Eliminar valores NA
  x <- na.omit(x)

  # Verificar si el vector está vacío
  if (length(x) == 0) return(NA_character_)

  # Calcular la frecuencia de cada valor
  tabla <- table(x)

  # Identificar el/los valores con mayor frecuencia
  max_frecuencia <- max(tabla)
  modas <- names(tabla[tabla == max_frecuencia])

  # Verificar si todos los valores son únicos (sin moda)
  if (max_frecuencia == 1) return(NA_character_)

  # Retornar la moda como un string separado por comas
  return(paste(modas, collapse = ", "))
}
```

Explicación de la función:

1. `x` es el vector de datos para el cual se calculará la moda.
2. `na.omit(x)` elimina los valores faltantes del vector.
3. `table(x)` calcula la frecuencia de cada valor en el vector.
4. `max(tabla)` identifica la frecuencia máxima.
5. `names(tabla[tabla == max_frecuencia])` extrae los valores que tienen la frecuencia máxima.
6. `paste(modas, collapse = ", ")` retorna la moda como un string separado por comas en caso de múltiples modas.

Una vez definida la función `moda()`, se puede calcular la moda de la variable `Sepal.Length`:

```r
# Calculo de la moda de la longitud del sépalo
moda (iris$Sepal.Length)
```

```
[1] "5"
```

La moda de la longitud del sépalo es “5”, lo que indica que este valor es el más frecuente en el conjunto de datos. La diferencia entre la moda (5.00) y la media (5.84) sugiere que la distribución de la variable `Sepal.Length` puede tener una ligera asimetría o presentar múltiples picos.

19.4 Medidas de dispersión (globales)

Las medidas de dispersión cuantifican la variabilidad de los datos, es decir, qué tan dispersos están los valores alrededor de la media. Las medidas de dispersión más comunes son la varianza, la desviación estándar, el rango y el rango intercuartílico (IQR).

```
# Varianza y desviación estándar
var(iris$Sepal.Length)
```

```
[1] 0.6856935
```

```
sd(iris$Sepal.Length)
```

```
[1] 0.8280661
```

```
# Rango y rango intercuartílico
range(iris$Sepal.Length)
```

```
[1] 4.3 7.9
```

```
IQR(iris$Sepal.Length)
```

```
[1] 1.3
```

Interpretación:

1. `var()` mide la dispersión cuadrática media respecto de la media. Un valor alto indica mayor variabilidad.
2. `sd()` es la raíz cuadrada de la varianza y mantiene las unidades originales. Es una medida de dispersión más interpretable que la varianza.
3. `range()` devuelve los valores mínimo y máximo del conjunto de datos. La diferencia entre el valor máximo y el valor mínimo indica la amplitud total de los datos.
4. `IQR()` es el rango intercuartílico, que se calcula como la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1). El IQR es una medida de dispersión robusta frente a valores atípicos, ya que no se ve afectado por los valores extremos.

En resumen, las medidas de tendencia central y dispersión proporcionan información valiosa sobre las características principales de un conjunto de datos. La media, la mediana y la moda resumen el centro de la distribución, mientras que la varianza, la desviación estándar, el rango y el IQR cuantifican la variabilidad de los datos. La combinación de estas medidas permite obtener una visión integral de los datos y comprender su distribución y dispersión (Moore et al., 2017).

19.5 Medidas de tendencia central por grupos

El análisis de medidas de tendencia central por grupos permite comparar las características de diferentes subconjuntos de datos. A continuación, se explorarán dos enfoques para calcular la media, la mediana y la moda por especie en el conjunto de datos `iris`: el enfoque base con la función `aggregate()` y el enfoque moderno con el paquete `dplyr`.

19.5.1 Enfoque base con aggregate()

La función `aggregate()` es una herramienta versátil en R para realizar cálculos por grupos. Permite dividir un marco de datos por una o más variables categóricas y aplicar una función a cada subconjunto. En este caso, se utilizará `aggregate()` para calcular la media y la mediana de las variables numéricas por especie.

```
# Media y mediana por especie usando aggregate()
aggregate(. ~ Species,
          data = iris,
          FUN  = function(v) c(media = mean(v),
                               mediana = median(v)))
```

```
     Species Sepal.Length.media Sepal.Length.mediana Sepal.Width.media
1     setosa              5.006                 5.000             3.428
2 versicolor              5.936                 5.900             2.770
3  virginica              6.588                 6.500             2.974
  Sepal.Width.mediana Petal.Length.media Petal.Length.mediana Petal.Width.media
1               3.400              1.462                 1.500             0.246
2               2.800              4.260                 4.350             1.326
3               3.000              5.552                 5.550             2.026
  Petal.Width.mediana
1               0.200
2               1.300
3               2.000
```

La función `aggregate()` divide el marco de datos `iris` por la variable categórica `Species` y aplica la función especificada a cada subconjunto (Venables & Ripley, 2002). En este caso, la función calcula la media y la mediana de cada variable numérica para cada especie. La salida muestra los valores de la media y la mediana para cada variable y especie.

19.5.2 Enfoque moderno con dplyr (media, mediana y moda)

Para facilitar la comparación de estadísticos descriptivos entre diferentes especies y variables, se propone generar una tabla resumen que incluya la media, la mediana y la moda para cada combinación de especie y variable. A continuación, se muestra el código para generar esta tabla utilizando el paquete `dplyr`.

```r
# Crear tabla resumen
tabla_resumen <- iris %>%
  # Convertir a formato largo para facilitar los cálculos
  pivot_longer(
    cols = -Species,
    names_to = "Variable",
    values_to = "Valor"
  ) %>%
  group_by(Species, Variable) %>%
  summarise(
    Media = round(mean(Valor, na.rm = TRUE), 2),
    Mediana = round(median(Valor, na.rm = TRUE), 2),
    Moda = moda(Valor),
  ) %>%
  arrange(Species, Variable)

# Visualizar la tabla resumen
tabla_resumen
```

```
# A tibble: 12 x 5
# Groups:   Species [3]
   Species    Variable     Media Mediana Moda
   <fct>      <chr>        <dbl>   <dbl> <chr>
 1 setosa     Petal.Length  1.46    1.5  1.4, 1.5
 2 setosa     Petal.Width   0.25    0.2  0.2
 3 setosa     Sepal.Length  5.01    5    5, 5.1
 4 setosa     Sepal.Width   3.43    3.4  3.4
 5 versicolor Petal.Length  4.26    4.35 4.5
 6 versicolor Petal.Width   1.33    1.3  1.3
 7 versicolor Sepal.Length  5.94    5.9  5.5, 5.6, 5.7
 8 versicolor Sepal.Width   2.77    2.8  3
 9 virginica  Petal.Length  5.55    5.55 5.1
10 virginica  Petal.Width   2.03    2    1.8
11 virginica  Sepal.Length  6.59    6.5  6.3
12 virginica  Sepal.Width   2.97    3    3
```

En este código:

1. La función `moda` calcula la moda de un vector numérico, manejando adecuadamente valores faltantes y posibles empates.
2. `pivot_longer()` transforma el conjunto de datos de formato ancho a largo, facilitando el cálculo de estadísticos por variable.
3. `group_by(Species, Variable)` agrupa los datos por especie y por cada característica morfométrica.
4. `summarise()` calcula la media (`mean`), la mediana (`median`) y la moda (`moda`) para cada grupo, redondeando los valores numéricos a dos decimales para mejorar la presentación.

5. `.groups = "drop"` elimina la estructura de agrupamiento tras el resumen, dejando la tabla lista para su visualización o exportación.
6. `arrange(Species, Variable)` ordena la tabla para facilitar la comparación entre especies y variables.

La tabla resumen permite comparar de manera clara y directa los valores centrales de cada variable morfométrica entre las especies de iris. Por ejemplo, se observa que *Iris virginica* presenta, en promedio, sépalos más largos que las otras especies, mientras que *Iris setosa* tiene los sépalos más anchos. La cercanía entre la media y la mediana en la mayoría de los casos indica que las distribuciones de las variables son aproximadamente simétricas, es decir, no presentan una asimetría significativa. Cuando la moda coincide con la mediana y la media, se refuerza la idea de simetría y ausencia de sesgo. Por el contrario, diferencias notables entre estos estadísticos pueden sugerir la presencia de valores atípicos, discretización de los datos o una ligera asimetría en la distribución (Navarro, 2019).

En resumen, ambos enfoques permiten calcular medidas de tendencia central por grupos, pero el enfoque moderno con `dplyr` ofrece mayor flexibilidad y claridad en la presentación de los resultados. La tabla resumen generada con `dplyr` facilita la comparación entre especies y variables, y promueve la reproducibilidad y claridad en el análisis (Wickham et al., 2023).

19.6 Resumen estadístico completo

R proporciona diversas funciones para obtener resúmenes estadísticos de manera rápida y eficiente. Además de las funciones base, el paquete `psych` ofrece descripciones más detalladas y completas de los datos. A continuación, se explorarán ambas opciones para obtener una visión integral de las características del conjunto de datos `iris`.

19.6.1 Resumen estadístico con funciones base

La función `summary()` es una herramienta fundamental en R para obtener un panorama general de los datos. Proporciona información sobre los valores mínimos, máximos, cuartiles y la media de cada variable numérica, así como la frecuencia de cada categoría en las variables factor (o categóricas).

```
# Resumen estadístico con funciones base
summary(iris)
```

```
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
```

```
 versicolor:50
 virginica :50
```

Interpretación:

1. **Variables numéricas:** Para cada variable numérica (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width), `summary()` muestra el valor mínimo (Min.), el primer cuartil (1st Qu.), la mediana (Median), la media (Mean), el tercer cuartil (3rd Qu.) y el valor máximo (Max.). Estos estadísticos permiten evaluar la distribución y dispersión de los datos.
2. **Variable categórica:** Para la variable Species, `summary()` muestra la frecuencia de cada especie (setosa, versicolor, virginica). Esto permite verificar si las clases están balanceadas o no.

Resumen estadístico detallado con el paquete psych

El paquete `psych` proporciona funciones para obtener descripciones más detalladas de los datos, incluyendo medidas de tendencia central, dispersión, forma de la distribución y error estándar. La función `describe()` es especialmente útil para obtener un resumen completo de las variables numéricas.

```r
# Instalar y cargar el paquete psych si es necesario
if (!require(psych)) install.packages("psych")


# Resumen detallado con psych
describe(iris[, 1:4])
```

```
             vars   n mean   sd median trimmed  mad min max range  skew
Sepal.Length    1 150 5.84 0.83   5.80    5.81 1.04 4.3 7.9   3.6  0.31
Sepal.Width     2 150 3.06 0.44   3.00    3.04 0.44 2.0 4.4   2.4  0.31
Petal.Length    3 150 3.76 1.77   4.35    3.76 1.85 1.0 6.9   5.9 -0.27
Petal.Width     4 150 1.20 0.76   1.30    1.18 1.04 0.1 2.5   2.4 -0.10
             kurtosis   se
Sepal.Length    -0.61 0.07
Sepal.Width      0.14 0.04
Petal.Length    -1.42 0.14
Petal.Width     -1.36 0.06
```

Interpretación:

1. **vars:** Número de variable.
2. **n:** Número de observaciones.
3. **mean:** Media.
4. **sd:** Desviación estándar.

5. **median:** Mediana.
6. **trimmed:** Media truncada (5% por defecto).
7. **mad:** Desviación absoluta mediana.
8. **min:** Valor mínimo.
9. **max:** Valor máximo.
10. **range:** Rango (max - min).
11. **skew:** Asimetría.
12. **kurtosis:** Curtosis.
13. **se:** Error estándar de la media.

La función `describe()` proporciona información adicional sobre la forma de la distribución de los datos. La asimetría (skew) mide la falta de simetría de la distribución, mientras que la curtosis (kurtosis) mide la concentración de los datos alrededor de la media. Estos estadísticos son útiles para identificar posibles valores atípicos y evaluar la normalidad de los datos.

En resumen, `summary()` ofrece un panorama rápido de los estadísticos básicos, mientras que `describe()` (del paquete `psych`) añade información sobre la asimetría, curtosis y error estándar, profundizando el diagnóstico de los datos (Revelle, 2023). La combinación de ambas funciones permite obtener una visión completa y detallada de las características del conjunto de datos.

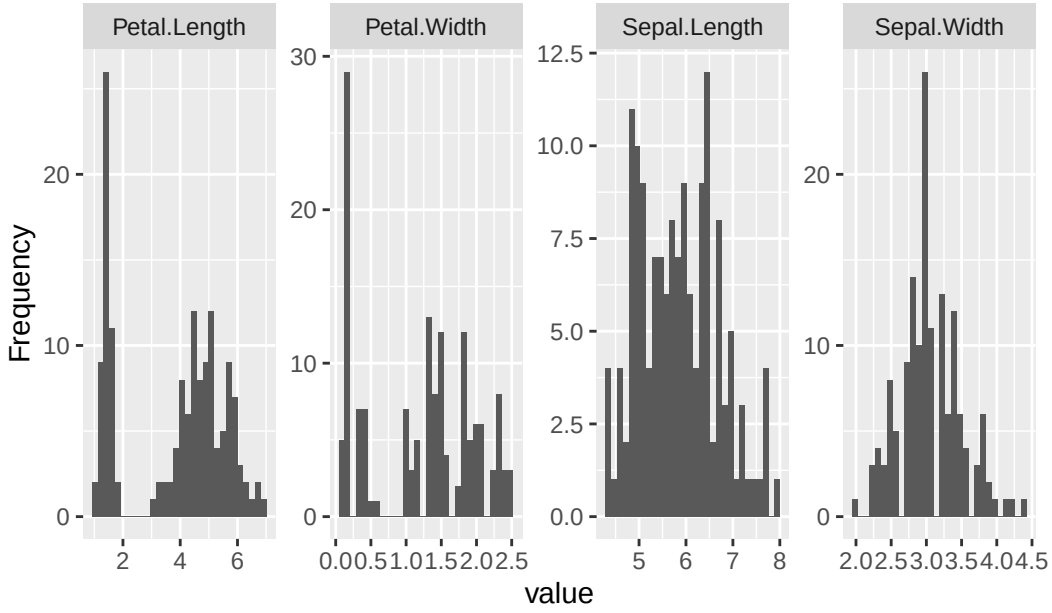
19.7 Visualizaciones básicas con DataExplorer y ggplot2

El paquete `DataExplorer` permite generar visualizaciones exploratorias de manera eficiente y automática, facilitando la interpretación de los datos. Este paquete proporciona funciones para obtener una visión general de las variables, sus distribuciones y las relaciones entre ellas, optimizando el análisis exploratorio inicial (Cui, 2023).

19.7.1 Histogramas de variables numéricas

La función `plot_histogram()` genera histogramas para cada variable numérica del conjunto de datos, lo que permite visualizar la distribución de los datos y detectar posibles asimetrías o valores atípicos.

```
# Histogramas de variables numéricas
plot_histogram(iris)
```

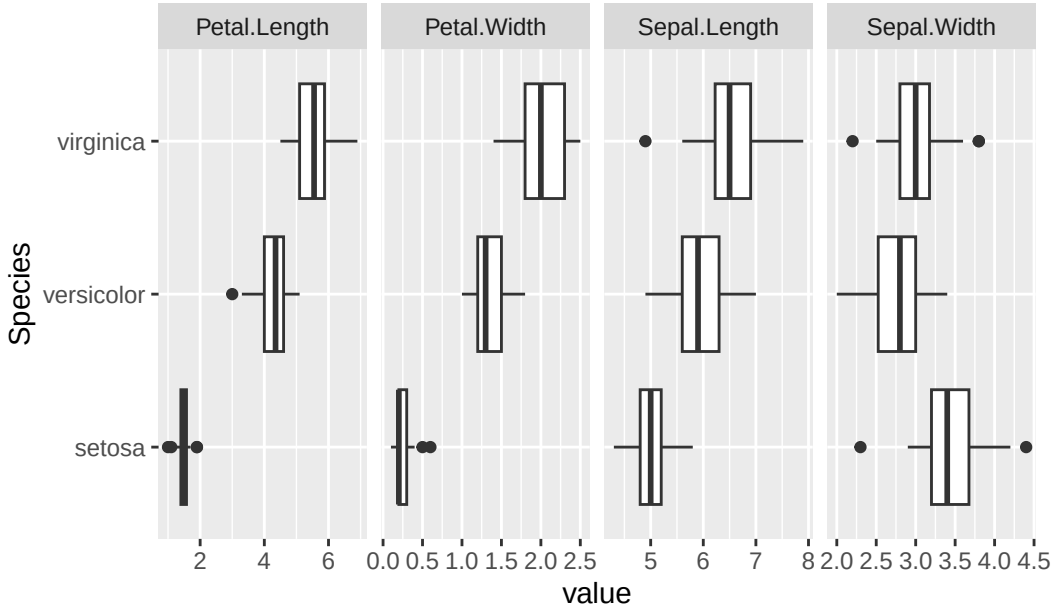


Interpretación: Los histogramas permiten visualizar la distribución de cada variable numérica y detectar posibles asimetrías o valores atípicos. Por ejemplo, se puede observar si la distribución es simétrica, asimétrica a la derecha (positiva) o asimétrica a la izquierda (negativa).

19.7.2 Diagramas de caja por especie

La función `plot_boxplot()` genera diagramas de caja para cada variable numérica, agrupados por la variable categórica `Species`. Esto permite comparar la distribución de cada variable entre las diferentes especies.

```
# Diagramas de caja por especie
plot_boxplot(iris, by = "Species")
```

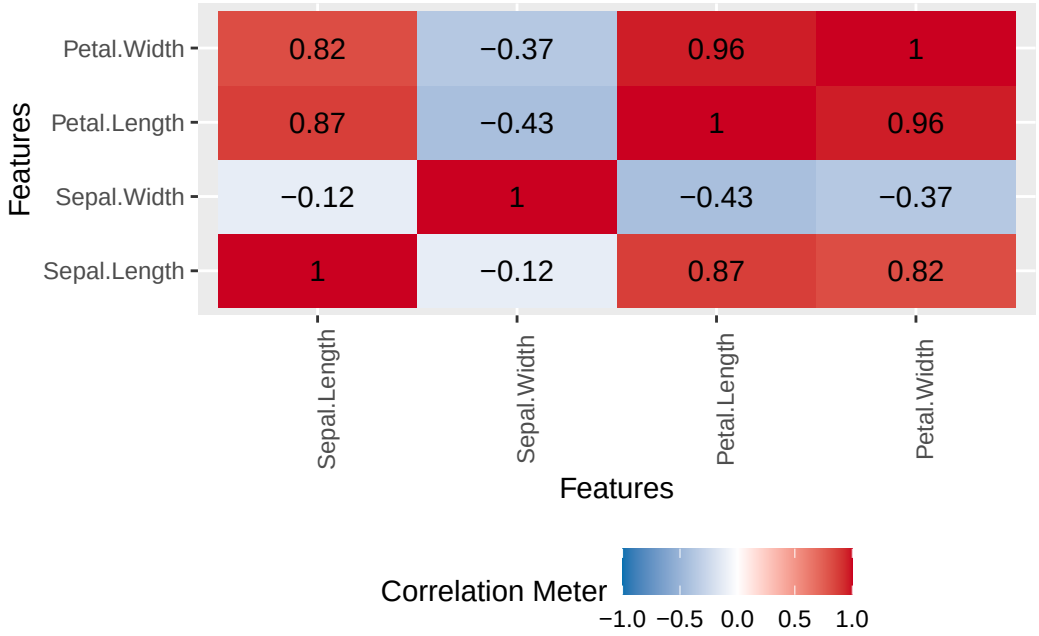


Interpretación: Los diagramas de caja permiten comparar la distribución de cada variable entre las diferentes especies. Se puede observar la mediana, los cuartiles, los valores atípicos y la dispersión de los datos para cada especie.

19.7.3 Mapa de calor de correlaciones

La función `plot_correlation()` genera un mapa de calor de las correlaciones entre las variables numéricas del conjunto de datos. Esto permite visualizar las correlaciones de manera gráfica y detectar las relaciones más fuertes entre las variables.

```
# Mapa de calor de correlaciones
plot_correlation(iris[, 1:4])
```



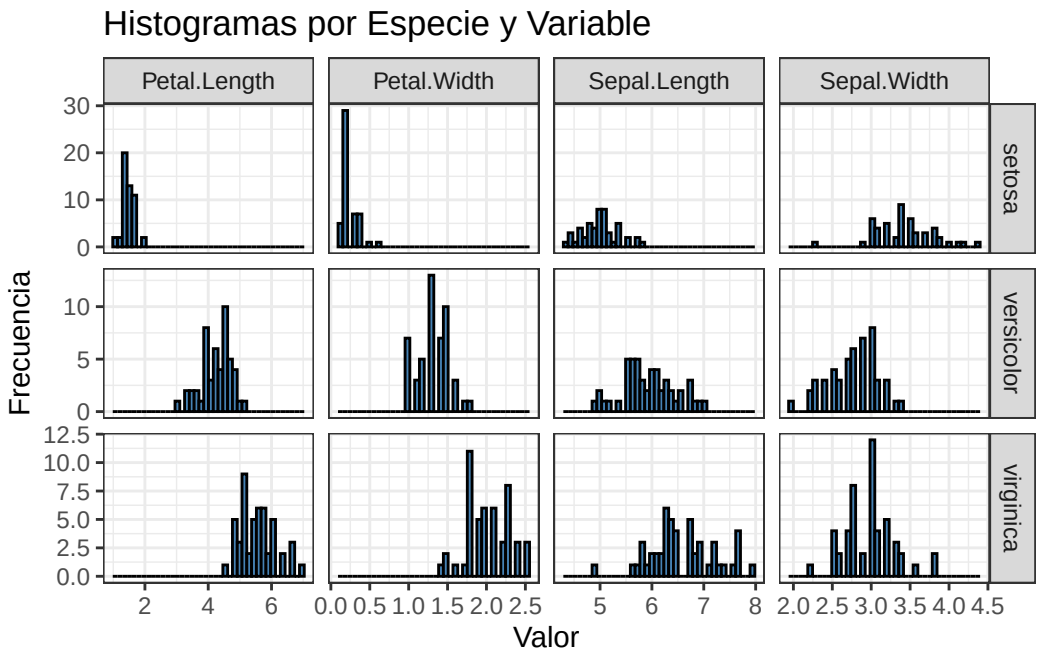
Interpretación: El mapa de calor de correlaciones permite visualizar las correlaciones entre las variables numéricas de manera gráfica. Los colores más intensos indican correlaciones más fuertes, mientras que los colores más claros indican correlaciones más débiles.

19.7.4 Visualización de histogramas por especie y variable

Para visualizar la distribución de cada variable dentro de cada especie, se propone generar histogramas individuales para cada combinación de especie y variable. A continuación, se muestra el código para generar estos histogramas utilizando el paquete `ggplot2`:

```
library(ggplot2)

# Convertir los datos a formato largo
iris_long <- iris %>%
  tidyr::pivot_longer(
    cols = starts_with("Sepal") | starts_with("Petal"),
    names_to = "Variable",
    values_to = "Value"
  )

# Generar histogramas para cada especie y variable
ggplot(iris_long, aes(x = Value)) +
  geom_histogram(bins = 40, fill = "steelblue", color = "black") +
  facet_grid(Species ~ Variable, scales = "free") +
  labs(
    title = "Histogramas por Especie y Variable",
    x = "Valor",
    y = "Frecuencia"
```

```
) +
theme_bw()
```



En este código:

1. `library(ggplot2)` carga el paquete `ggplot2`, que permite crear gráficos de alta calidad.
2. `iris %>% tidyr::pivot_longer(...)` transforma los datos de formato ancho a formato largo, facilitando la generación de gráficos.
3. `ggplot(iris_long, aes(x = Value))` crea un objeto gráfico base, especificando que se utilizará la variable "Value" en el eje x.
4. `geom_histogram(bins = 40, fill = "steelblue", color = "black")` agrega un histograma al gráfico, especificando el número de bins, el color de relleno y el color del borde.
5. `facet_grid(Species ~ Variable, scales = "free")` genera un panel de gráficos, mostrando un histograma para cada combinación de especie y variable. El argumento `scales = "free"` permite que cada histograma tenga su propia escala en el eje x.
6. `labs(...)` agrega etiquetas al gráfico, incluyendo el título, la etiqueta del eje x y la etiqueta del eje y.
7. `theme_bw()` aplica un tema visual en blanco y negro al gráfico.

Interpretación: Los histogramas permiten visualizar la distribución de cada variable dentro de cada especie. Se puede observar la forma de la distribución, la presencia de valores atípicos y la dispersión de los datos para cada combinación de especie y variable. El argumento `scales = "free"` permite que cada histograma tenga su propia escala en

el eje x, lo que facilita la comparación entre las diferentes combinaciones de especie y variable.

La combinación de las funciones del paquete `DataExplorer` y la visualización de histogramas con `ggplot2` permite obtener una visión completa y detallada de las características del conjunto de datos, facilitando la interpretación de los datos y la identificación de posibles patrones o relaciones (Wickham, 2016).

20 Regresión lineal usando R

La regresión lineal constituye una de las técnicas estadísticas fundamentales para el análisis de la relación entre variables cuantitativas. Su objetivo principal es modelar la relación existente entre una variable dependiente (también denominada respuesta) y una o más variables independientes (o predictoras), permitiendo así predecir valores de la variable dependiente a partir de valores conocidos de las independientes (Kutner et al., 2005; Montgomery et al., 2012).

20.1 Definición y Objetivos

El análisis de regresión lineal busca cuantificar y describir la relación lineal entre variables, proporcionando una ecuación matemática que representa dicha relación. En el caso más simple, la regresión lineal simple, se estudia la relación entre dos variables: una dependiente `Y` y una independiente `X`. El modelo se expresa generalmente como:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

En esta ecuación, ***Y*** representa la variable dependiente, ***X*** es la variable independiente, ***ß0*** es la intersección (el valor de ***Y***) cuando ***X*** es cero) y ***ß1*** es la pendiente que indica el cambio en ***Y*** por cada unidad de cambio en ***X***. El término epilson ( ) representa el error del modelo, que captura la variabilidad en ***Y*** que no se explica por ***X*** (Kutner, Nachtsheim, Neter, & Li, 2005).

La regresión lineal es ampliamente utilizada en diversas disciplinas, como economía, biología, ingeniería y ciencias sociales, debido a su capacidad para identificar tendencias, realizar pronósticos y evaluar la fuerza y dirección de las relaciones entre variables (Kutner et al., 2005; Hernández et al., 2024).

20.1.1 Tipos de Regresión Lineal

Existen dos tipos principales de regresión lineal: la regresión lineal simple y la regresión lineal múltiple. La regresión lineal simple involucra una sola variable independiente, mientras que la regresión lineal múltiple considera dos o más variables independientes para explicar la variabilidad de la variable dependiente (Kutner et al., 2005). La extensión a modelos múltiples permite capturar relaciones más complejas y controlar el efecto de variables adicionales.

20.1.2 Supuestos del Modelo de Regresión Lineal

Para que los resultados del modelo de regresión lineal sean válidos y confiables, es necesario que se cumplan ciertos supuestos fundamentales (Montgomery et al., 2012):

1. **Linealidad:** Se asume que la relación entre la variable dependiente y las independientes es lineal.
2. **Independencia:** Los errores (residuos) del modelo son independientes entre sí.
3. **Homoscedasticidad:** La varianza de los errores es constante a lo largo de los valores de las variables independientes.
4. **Normalidad:** Los errores del modelo se distribuyen normalmente.

La verificación de estos supuestos es esencial, ya que su incumplimiento puede afectar la validez de las inferencias y predicciones realizadas a partir del modelo (Kutner et al., 2005; Montgomery et al., 2012).

20.2 Base de datos

El presente ejemplo práctico expone el procedimiento para realizar un análisis de regresión lineal simple utilizando un conjunto de datos experimentales sobre esporofitos. Los datos fueron recolectados en el laboratorio de cultivo de tejidos de la Facultad de Agronomía de la Universidad de San Carlos de Guatemala, en el marco de un estudio enfocado en la reproducción in vitro del helecho conocido como calahuala (*Phlebodium pseudoaureum* (Cav.) Lellinger).

En el experimento, se midió la altura de cada esporofito y se registró la cantidad de esporofitos germinados en 30 frascos, todos ellos cultivados en medio Murashige y Skoog. La información analizada corresponde a los resultados obtenidos bajo estas condiciones controladas. Este análisis toma como referencia la investigación de Rosales Castillo (2005), quien desarrolló un protocolo de micropropagación de calahuala empleando tres tipos de explantes y diferentes medios de cultivo in vitro.

El objetivo principal de este análisis es evaluar la relación entre la cantidad de esporofitos germinados (variable independiente) y la altura de los esporofitos (variable dependiente), empleando la regresión lineal simple como herramienta estadística. Además de ajustar un modelo que describa esta relación, se busca verificar rigurosamente los supuestos estadísticos que garantizan la validez de las inferencias obtenidas.

Acceso a recursos: El script completo con los ejemplos desarrollados y la base de datos sobre esporofitos están disponibles para su consulta y descarga en el siguiente repositorio: https://github.com/Ludwing-MJ/Reg_Lineal_EJ.git

20.3 Preparación del Entorno en R

20.3.1 Instalación y Carga de Paquetes

Para realizar el análisis de regresión lineal con datos de esporofitos de calahuala (Phlebodium pseudoaureum), es necesario utilizar varios paquetes de R que facilitan la importación, manipulación y visualización de datos. A continuación, se presenta el código para la instalación y carga de los paquetes necesarios (Grolemund & Wickham, 2017):

```
# Instalación y carga de paquetes necesarios
# Para leer archivos Excel
if(!require(readxl)) install.packages("readxl")
# Para visualización de datos
if(!require(ggplot2)) install.packages("ggplot2")
# Para manipulación de datos
if(!require(dplyr)) install.packages("dplyr")
# Para diagnósticos de regresión
if(!require(car)) install.packages("car")
# Para pruebas de supuestos
if(!require(lmtest)) install.packages("lmtest")
# Para pruebas de normalidad
if(!require(nortest)) install.packages("nortest")
# Para estadística descriptiva
if(!require(psych)) install.packages("psych")
```

20.3.2 Importación y Exploración de Datos

Los datos sobre los esporofitos de calahuala se encuentran almacenados en un archivo Excel (esporofitos.xlsx). A continuación, se presenta el código para importar y realizar una exploración inicial de los datos:

```
# Importar datos desde el archivo Excel
datos_esporofitos <- read_excel("esporofitos.xlsx")

# Visualizar las primeras filas del conjunto de datos
head(datos_esporofitos)
```

```
# A tibble: 6 x 3
  frasco cantidad_de_esporofitos altura_mm
   <dbl>                   <dbl>     <dbl>
1      1                      40      21.4
2      2                      45      21
3      3                      60      20.5
4      4                      55      20
5      5                      58      21
6      6                      40      21.7
```

```r
# Estructura del conjunto de datos
str(datos_esporofitos)
```

```
tibble [30 x 3] (S3: tbl_df/tbl/data.frame)
 $ frasco                 : num [1:30] 1 2 3 4 5 6 7 8 9 10 ...
 $ cantidad_de_esporofitos: num [1:30] 40 45 60 55 58 40 52 65 45 40 ...
 $ altura_mm              : num [1:30] 21.4 21 20.5 20 21 21.7 21.1 19.5 21.1 21.3 ...
```

```r
# Resumen estadístico básico
summary(datos_esporofitos)
```

```
     frasco      cantidad_de_esporofitos   altura_mm
 Min.   : 1.00   Min.   : 40.0           Min.   :12.10
 1st Qu.: 8.25   1st Qu.: 58.5           1st Qu.:14.03
 Median :15.50   Median :150.0           Median :16.95
 Mean   :15.50   Mean   :144.3           Mean   :17.10
 3rd Qu.:22.75   3rd Qu.:215.2           3rd Qu.:20.38
 Max.   :30.00   Max.   :267.0           Max.   :21.70
```

```r
# Verificar valores faltantes
colSums(is.na(datos_esporofitos))
```

```
                frasco cantidad_de_esporofitos                altura_mm
                     0                        0                        0
```

20.3.3 Preparación y Limpieza de Datos

Es importante realizar una limpieza inicial de los datos para asegurar su calidad antes del análisis (Wickham, 2016):

```r
# Eliminar filas con valores faltantes (si existen)
datos_esporofitos <- na.omit(datos_esporofitos)

# Renombrar columnas para facilitar el análisis (si es necesario)
names(datos_esporofitos) <- c("frasco", "cantidad", "altura")

# Verificar la estructura final de los datos
str(datos_esporofitos)
```

```
tibble [30 x 3] (S3: tbl_df/tbl/data.frame)
 $ frasco  : num [1:30] 1 2 3 4 5 6 7 8 9 10 ...
 $ cantidad: num [1:30] 40 45 60 55 58 40 52 65 45 40 ...
 $ altura  : num [1:30] 21.4 21 20.5 20 21 21.7 21.1 19.5 21.1 21.3 ...
```

Este conjunto de códigos prepara el entorno para realizar el análisis de regresión lineal con los datos de esporofitos de calahuala, siguiendo las mejores prácticas en análisis de datos con R (Grolemund & Wickham, 2017). La estructura organizada facilita la reproducibilidad del análisis y permite un manejo eficiente de los datos provenientes del estudio de micropropagación realizado por Rosales Castillo (2005).

20.4 Análisis Descriptivo de los Datos

El análisis descriptivo de los datos es un paso crucial antes de realizar un análisis de regresión lineal. Permite comprender las características principales de las variables, identificar posibles problemas en los datos (como valores atípicos o distribuciones no normales) y evaluar la pertinencia de aplicar un modelo de regresión lineal (Tukey, 1977). A continuación, se presenta el análisis descriptivo utilizando el paquete `psych` en R.

20.4.1 Estadísticas Descriptivas con el Paquete `psych`

El paquete `psych` proporciona funciones convenientes para calcular y presentar estadísticas descriptivas de manera eficiente (Revelle, 2023). Se utiliza la función `describe()` para obtener un resumen de las principales estadísticas de las variables de interés: altura de los esporofitos y cantidad de esporofitos germinados.

```
# Calcular estadísticas descriptivas
descripcion <- describe(datos_esporofitos)

# Visualizar las estadísticas descriptivas
print(descripcion)
```

```
         vars  n  mean    sd median trimmed    mad  min   max range  skew
frasco      1 30  15.5  8.80  15.50   15.50  11.12  1.0  30.0  29.0  0.00
cantidad    2 30 144.3 78.95 150.00  143.00 124.54 40.0 267.0 227.0 -0.01
altura      3 30  17.1  3.19  16.95   17.14   4.60 12.1  21.7   9.6  0.00
         kurtosis    se
frasco      -1.32  1.61
cantidad    -1.53 14.41
altura      -1.48  0.58
```

La función `describe()` proporciona las siguientes estadísticas para cada variable:

1. `vars`: Número de variable.
2. `n`: Número de observaciones.
3. `mean`: Media.
4. `sd`: Desviación estándar.
5. `median`: Mediana.
6. `trimmed`: Media recortada al 10%.

7. `mad`: Desviación absoluta mediana.
8. `min`: Valor mínimo.
9. `max`: Valor máximo.
10. `range`: Rango (máximo - mínimo).
11. `skew`: Asimetría.
12. `kurtosis`: Curtosis.
13. `se`: Error estándar de la media.

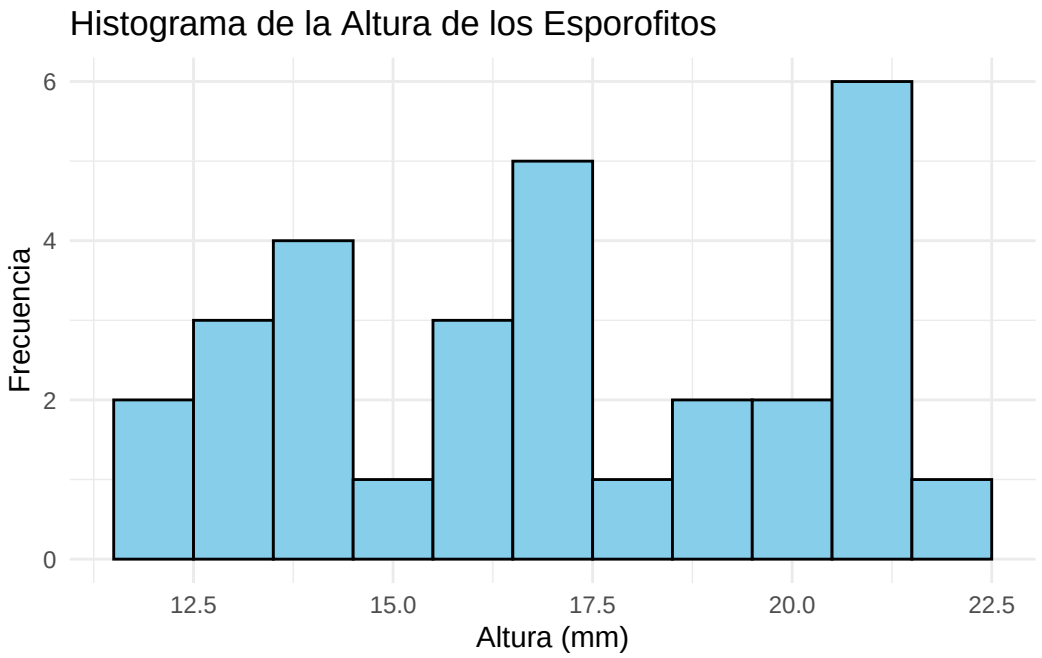
20.4.2 Visualización de Datos

La visualización de datos es fundamental para complementar el análisis descriptivo y obtener una comprensión más profunda de la distribución y relación entre las variables (Cleveland, 1993; Tufte, 2001). Se utilizan histogramas y diagramas de dispersión para visualizar la distribución de cada variable y la relación entre ellas.

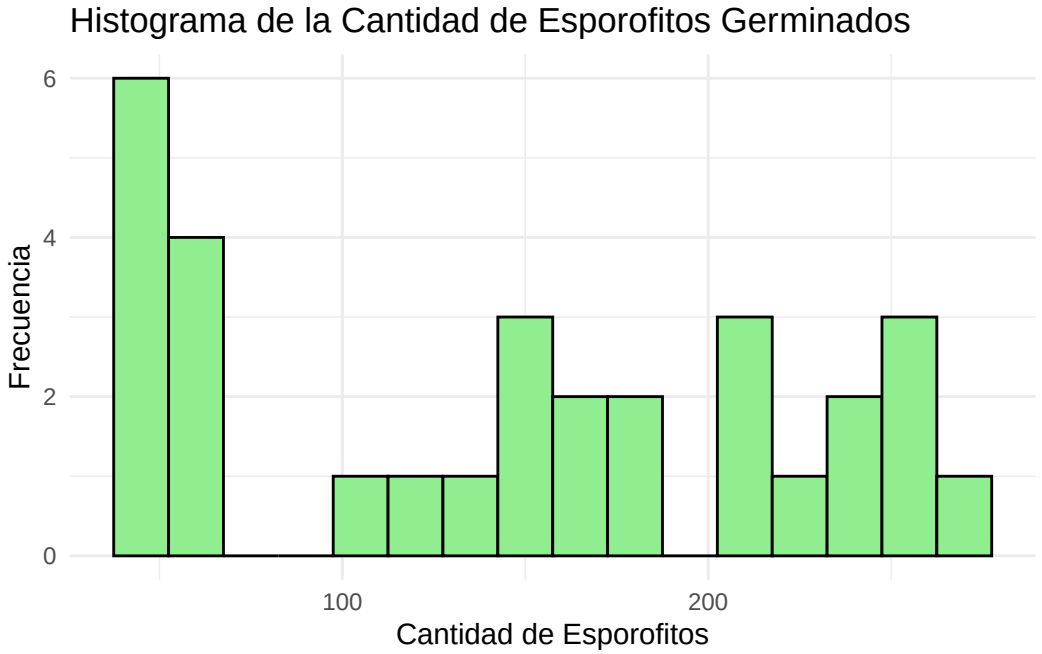
20.4.2.1 Histogramas

Los histogramas permiten visualizar la distribución de cada variable y evaluar su forma, simetría y presencia de valores atípicos.

```
# Histograma de la altura de los esporofitos
ggplot(datos_esporofitos, aes(x = altura)) +
  geom_histogram(binwidth = 1, fill = "skyblue", color = "black") +
  labs(title = "Histograma de la Altura de los Esporofitos",
       x = "Altura (mm)",
       y = "Frecuencia") +
  theme_minimal()
```




```
# Histograma de la cantidad de esporofitos germinados
ggplot(datos_esporofitos, aes(x = cantidad)) +
  geom_histogram(binwidth = 15, fill = "lightgreen", color = "black") +
  labs(title = "Histograma de la Cantidad de Esporofitos Germinados",
       x = "Cantidad de Esporofitos",
       y = "Frecuencia") +
  theme_minimal()
```

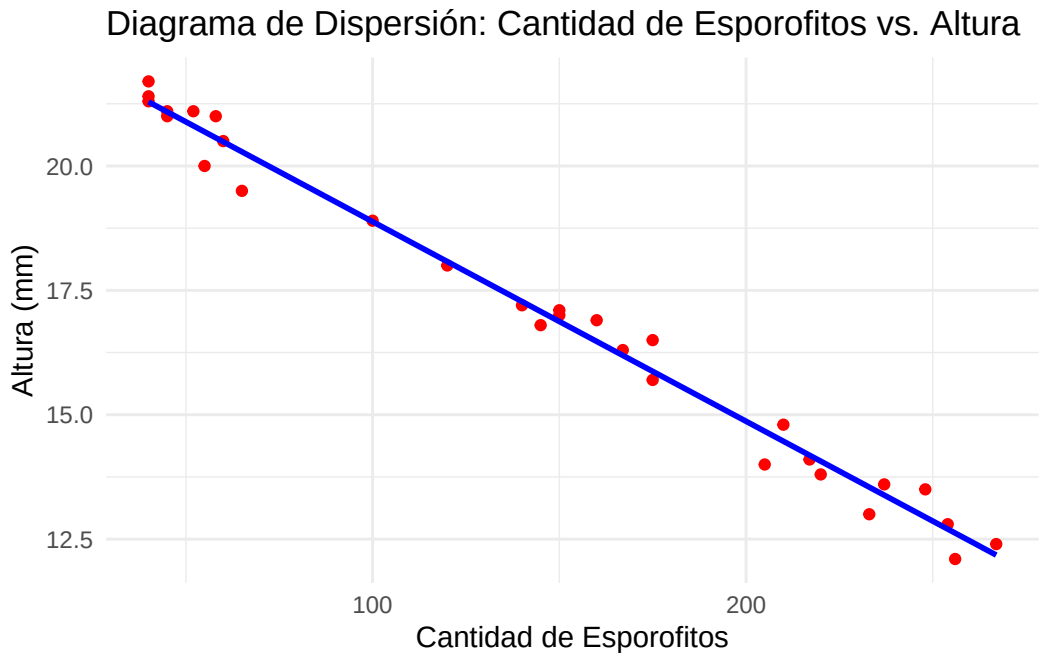


20.4.2.2 Diagrama de Dispersión

El diagrama de dispersión permite visualizar la relación entre la altura de los esporofitos y la cantidad de esporofitos germinados.

```
# Diagrama de dispersión
ggplot(datos_esporofitos, aes(x = cantidad, y = altura)) +
  geom_point(color = "red") +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  labs(title = "Diagrama de Dispersión: Cantidad de Esporofitos vs. Altura",
       x = "Cantidad de Esporofitos",
       y = "Altura (mm)") +
  theme_minimal()
```

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



20.4.3 Interpretación del Análisis Descriptivo

El análisis descriptivo proporciona información valiosa sobre las características de los datos. Por ejemplo, la media y la mediana indican el valor central de cada variable, mientras que la desviación estándar y el rango miden su dispersión. La asimetría y la curtosis informan sobre la forma de la distribución.

Los histogramas permiten identificar si las variables tienen una distribución aproximadamente normal o si presentan asimetrías o valores atípicos. El diagrama de dispersión permite evaluar visualmente si existe una relación lineal entre las variables y si hay patrones inusuales en los datos.

En el contexto del análisis de regresión lineal, el análisis descriptivo ayuda a determinar si los datos cumplen con los supuestos del modelo y si es necesario realizar transformaciones en las variables para mejorar el ajuste del modelo (Kutner et al., 2005).

20.5 Ajuste del Modelo de Regresión Lineal

Una vez realizado el análisis descriptivo de los datos, el siguiente paso es ajustar el modelo de regresión lineal. Este proceso implica la estimación de los parámetros del modelo que mejor describen la relación entre la variable dependiente (altura de los esporofitos) y la variable independiente (cantidad de esporofitos germinados).

20.5.1 Creación del Modelo

Se utiliza la función `lm()` para ajustar el modelo de regresión lineal. La sintaxis general es `lm(variable_dependiente ~ variable_independiente, data = nombre_del_data_frame)`. En este caso, se busca modelar la altura de los esporofitos en función de la cantidad de esporofitos germinados (Montgomery et al., 2012).

```
# Ajustar el modelo de regresión lineal
modelo <- lm(altura ~ cantidad, data = datos_esporofitos)
```

20.5.2 Resumen del Modelo

Para obtener información detallada sobre el modelo ajustado, se utiliza la función `summary()`. Esta función proporciona los coeficientes estimados, el error estándar, el valor t, el valor p y el coeficiente de determinación (`R^2`) (Kutner et al., 2005).

```
# Resumen del modelo
resumen_modelo <- summary(modelo)

# Visualizar el resumen del modelo
print(resumen_modelo)
```

```

Call:
lm(formula = altura ~ cantidad, data = datos_esporofitos)

Residuals:
     Min       1Q   Median       3Q      Max
-0.78523 -0.15056  0.01664  0.22403  0.62850

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 22.8933399  0.1446248  158.29   <2e-16 ***
cantidad    -0.0401248  0.0008826  -45.46   <2e-16 ***
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3753 on 28 degrees of freedom
Multiple R-squared:  0.9866,    Adjusted R-squared:  0.9862
F-statistic:  2067 on 1 and 28 DF,  p-value: < 2.2e-16
```

El resumen del modelo incluye la siguiente información clave:

1. **Coefficients:**
    a. `Estimate`: Estimación de los coeficientes del modelo (intercepto y pendiente).
    b. `Std. Error`: Error estándar de los coeficientes.
    c. `t value`: Valor t para la prueba de hipótesis de que el coeficiente es igual a cero.
    d. `Pr(>|t|)`: Valor p asociado al valor t.
2. **Residual standard error:** Estimación de la desviación estándar de los residuos.
3. **Multiple R-squared:** Coeficiente de determinación (`R^2`), que indica la proporción de la varianza de la variable dependiente explicada por el modelo.
4. **Adjusted R-squared:** Coeficiente de determinación ajustado, que tiene en cuenta el número de variables independientes en el modelo.
5. **F-statistic:** Estadístico F para la prueba de hipótesis de que todos los coeficientes del modelo son iguales a cero.
6. **p-value:** Valor p asociado al estadístico F.

20.5.3 Interpretación del Modelo de Regresión Lineal

El modelo ajustado es: `altura=22.89-0.04×cantidad`

Donde:

1. **Intercepto (0=22.89):** El intercepto representa la altura estimada de los esporofitos cuando la cantidad de esporofitos germinados es cero. Matemáticamente, si no hubiera esporofitos germinados en un frasco, la altura estimada sería de 22.89 mm. Sin embargo, en el contexto biológico, este valor puede carecer de sentido práctico, ya que no es realista tener altura sin esporofitos germinados, pero es necesario para la ecuación del modelo (Kutner et al., 2005).
2. **Pendiente (1=−0.04):** La pendiente indica que, por cada esporofito germinado adicional en el frasco, la altura promedio de los esporofitos disminuye en 0.04 mm. Este valor negativo sugiere una relación inversa entre la cantidad de esporofitos germinados y la altura de los esporofitos: a mayor cantidad de esporofitos germinados, menor es la altura promedio de los mismos.

3. **Significancia estadística de los coeficientes:** Ambos coeficientes (intercepto y pendiente) presentan valores p menores a 2e-16, lo que indica que son altamente significativos desde el punto de vista estadístico. Esto significa que existe evidencia suficiente para afirmar que la cantidad de esporofitos germinados es un predictor relevante de la altura de los esporofitos en este experimento (Montgomery et al., 2012).
4. **Coeficiente de determinación (R2=0.9866):** El valor de R2 es 0.9866, lo que indica que el 98.66% de la variabilidad observada en la altura de los esporofitos es explicada por la cantidad de esporofitos germinados. Este valor extremadamente alto sugiere que el modelo ajustado tiene un excelente poder explicativo para estos datos.
5. **Error estándar de los residuos:** El error estándar de los residuos es 0.3753, lo que indica que, en promedio, las predicciones del modelo difieren de los valores observados en aproximadamente 0.3753 mm.
6. **Estadístico F y valor p global:** El estadístico F es 2067 con un valor p menor a 2.2e-16, lo que confirma que el modelo en su conjunto es significativo y que la variable independiente (cantidad) contribuye de manera significativa a explicar la variabilidad en la altura.

20.6 Diagnóstico del Modelo de Regresión Lineal

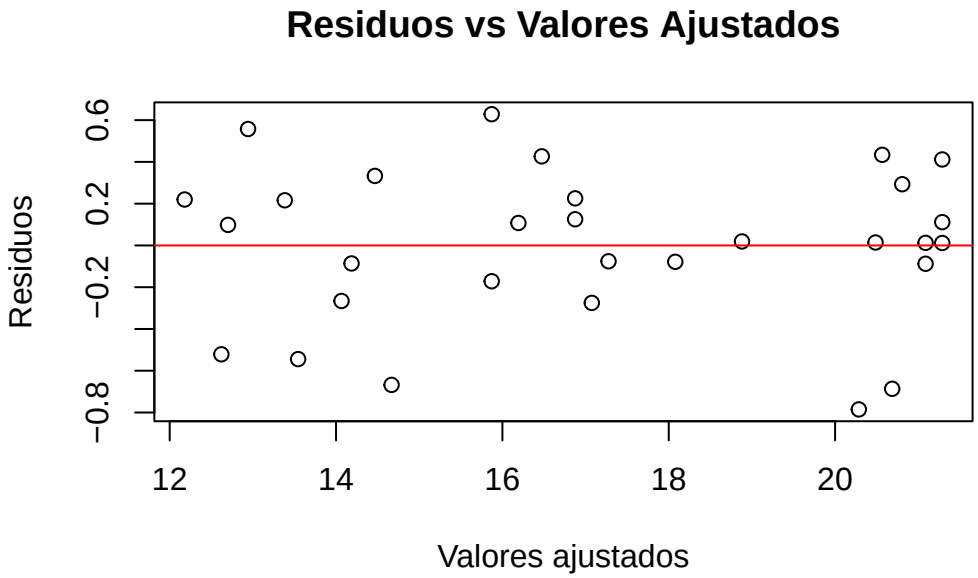
El diagnóstico del modelo de regresión lineal es una etapa esencial para validar los resultados obtenidos y garantizar que las inferencias realizadas sean confiables. Este proceso consiste en verificar que se cumplan los supuestos fundamentales del modelo, identificar posibles valores atípicos o influyentes y evaluar la calidad del ajuste (Kutner et al., 2005; Montgomery et al., 2012).

Los principales supuestos que deben cumplirse en un modelo de regresión lineal simple son: linealidad, independencia, homocedasticidad y normalidad de los residuos.

20.6.1 Linealidad

Se asume que la relación entre la variable independiente (cantidad de esporofitos germinados) y la variable dependiente (altura de los esporofitos) es lineal. Para verificar este supuesto, se recomienda observar el diagrama de dispersión y el gráfico de residuos versus valores ajustados.

```
# Gráfico de residuos vs valores ajustados
plot(modelo$fitted.values, modelo$residuals,
     xlab = "Valores ajustados",
     ylab = "Residuos",
     main = "Residuos vs Valores Ajustados")
abline(h = 0, col = "red")
```



Un patrón aleatorio alrededor de la línea horizontal en cero indica que el supuesto de linealidad es razonable.

20.6.2 Independencia de los residuos

La independencia de los residuos puede evaluarse mediante el test de Durbin-Watson, disponible en el paquete `lmtest` (Montgomery et al., 2012).

```
# Evaluación de la independencia de los residuos
dwtest(modelo)
```

```
    Durbin-Watson test

data:  modelo
DW = 2.7293, p-value = 0.9725
alternative hypothesis: true autocorrelation is greater than 0
```

Un valor p alto (mayor al nivel de significancia, que en investigación agricola normalmente es 0.05) en la prueba indica que hay evidencia de independencia de los residuos.

20.6.3 Homocedasticidad (igualdad de varianzas)

La homocedasticidad implica que la varianza de los residuos es constante a lo largo de los valores ajustados. Se puede evaluar visualmente con el gráfico de residuos y formalmente con la prueba de Breusch-Pagan.

```r
# Prueba de Breusch-Pagan
bptest(modelo)
```

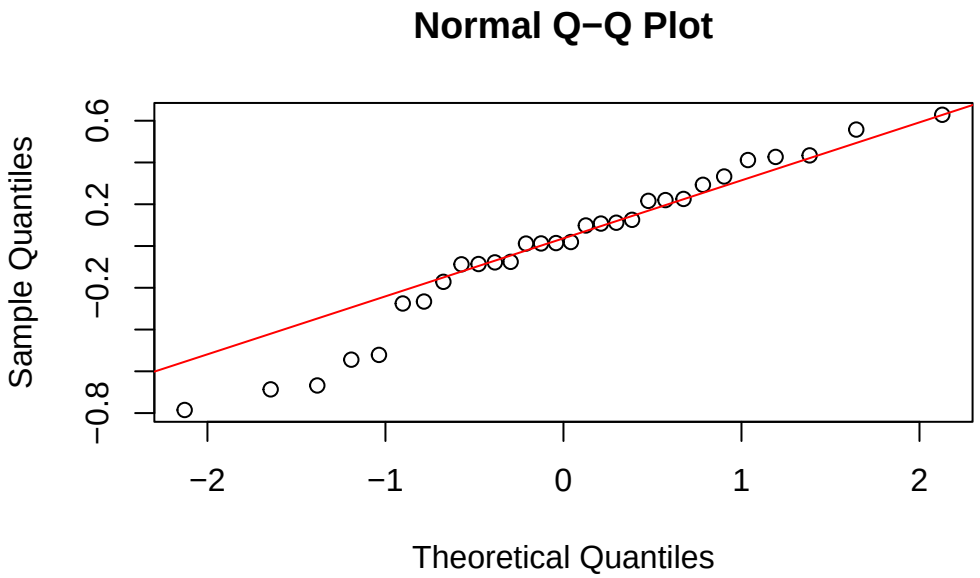
```
    studentized Breusch-Pagan test

data:  modelo
BP = 0.095527, df = 1, p-value = 0.7573
```

Un valor p alto (mayor al nivel de significancia, que en investigación agricola normalmente es 0.05) en la prueba indica que no hay evidencia de heterocedasticidad.

20.6.4 Normalidad de los residuos

La normalidad de los residuos puede evaluarse mediante un gráfico Q-Q y pruebas estadísticas como Shapiro-Wilk o Anderson-Darling.

```r
# Gráfico Q-Q
qqnorm(modelo$residuals)
qqline(modelo$residuals, col = "red")
```




```r
# Prueba de Shapiro-Wilk
shapiro.test(modelo$residuals)
```

```
    Shapiro-Wilk normality test
```

```
data:  modelo$residuals
W = 0.95651, p-value = 0.2516
```

Si los puntos del gráfico Q-Q se alinean aproximadamente sobre la línea y el valor p de la prueba es mayor a 0.05, se puede asumir normalidad de los residuos (Kutner et al., 2005).

20.6.5 Evaluación Global del Modelo

Los supuestos se cumplen se puede concluir que el modelo es adecuado para describir la relación entre la cantidad de esporofitos germinados y la altura de los esporofitos. En caso contrario, se debería considerar transformaciones de las variables, exclusión de valores atípicos o el uso de modelos alternativos (Kutner et al., 2005).

20.7 Predicciones

Una vez ajustado y validado el modelo de regresión lineal, es posible utilizarlo para realizar predicciones sobre la altura de los esporofitos a partir de nuevos valores de la cantidad de esporofitos germinados. Este proceso es fundamental para la aplicación práctica del modelo, ya que permite estimar resultados bajo diferentes escenarios experimentales (Kutner et al., 2005).

20.7.1 Creación de Nuevas Predicciones

Para generar predicciones, se utiliza la función `predict()` de R, la cual permite estimar el valor de la variable dependiente (altura) para nuevos valores de la variable independiente (cantidad). A continuación se muestra cómo crear un conjunto de nuevos datos y obtener las predicciones correspondientes:

```r
# Crear nuevos datos para predicción
nuevos_datos <- data.frame(cantidad = c(25, 50, 75))

# Realizar predicciones de altura
predicciones <- predict(modelo, newdata = nuevos_datos)
print(predicciones)
```

```
       1        2        3
21.89022 20.88710 19.88398
```

En este ejemplo, se predice la altura de los esporofitos para frascos con 25, 50 y 75 esporofitos germinados. El resultado será un vector con los valores estimados de altura para cada caso.

20.7.2 Intervalos de Confianza y Predicción

Además de las predicciones puntuales, es recomendable calcular intervalos de confianza e intervalos de predicción para cuantificar la incertidumbre asociada a las estimaciones (Kutner et al., 2005; Montgomery et al., 2012):

1. **Intervalo de confianza:** Indica el rango en el que se espera que se encuentre la media poblacional de la altura para un valor dado de la cantidad de esporofitos germinados, con un nivel de confianza especificado (por defecto, 95%).
2. **Intervalo de predicción:** Indica el rango en el que se espera que se encuentre un valor individual de la altura para un nuevo frasco con una cantidad específica de esporofitos germinados, considerando tanto la incertidumbre del modelo como la variabilidad individual.

El siguiente código muestra cómo obtener ambos intervalos:

```
# Intervalos de confianza para la media
intervalos_confianza <- predict(modelo,
                                newdata = nuevos_datos,
                                interval = "confidence")
print(intervalos_confianza)
```

```
       fit      lwr      upr
1 21.89022 21.63288 22.14756
2 20.88710 20.66627 21.10793
3 19.88398 19.69584 20.07212
```

```
# Intervalos de predicción para valores individuales
intervalos_prediccion <- predict(modelo,
                                 newdata = nuevos_datos,
                                 interval = "prediction")
print(intervalos_prediccion)
```

```
       fit      lwr      upr
1 21.89022 21.07957 22.70087
2 20.88710 20.08729 21.68691
3 19.88398 19.09257 20.67539
```

Los resultados mostrarán, para cada valor de cantidad, la predicción puntual, el límite inferior y el límite superior del intervalo correspondiente.

Estos procedimientos permiten aplicar el modelo de regresión lineal para estimar la altura de los esporofitos bajo diferentes condiciones experimentales y cuantificar la precisión de dichas estimaciones, lo que resulta fundamental para la toma de decisiones en el contexto de la micropropagación y el manejo experimental (Kutner et al., 2005; Montgomery et al., 2012).

20.8 Conclusiones

El análisis de regresión lineal simple realizado para evaluar la relación entre la cantidad de esporofitos germinados y la altura de los esporofitos en el experimento de micropropagación de calahuala permitió obtener resultados estadísticamente robustos y relevantes. A continuación, se presentan las conclusiones principales del estudio, considerando que el modelo ajustado cumplió con todos los supuestos fundamentales de la regresión lineal (linealidad, independencia, homocedasticidad y normalidad de los residuos).

En primer lugar, se identificó una relación lineal negativa y significativa entre la cantidad de esporofitos germinados y la altura de los esporofitos. Específicamente, el modelo estimó que por cada esporofito germinado adicional, la altura promedio de los esporofitos disminuye en aproximadamente 0.04 mm. Este hallazgo sugiere que, bajo las condiciones experimentales descritas, un mayor número de esporofitos germinados en un frasco se asocia con un menor crecimiento en altura de los mismos, lo que podría estar relacionado con la competencia por recursos en el medio de cultivo (Rosales Castillo, 2005).

El coeficiente de determinación (R2R^2R2) obtenido fue de 0.9866, lo que indica que el modelo explica el 98.66% de la variabilidad observada en la altura de los esporofitos. Este valor refleja un ajuste excelente y respalda la utilidad del modelo para describir y predecir la altura de los esporofitos a partir de la cantidad de esporofitos germinados.

La verificación de los supuestos del modelo confirmó la validez de las inferencias realizadas. No se detectaron problemas de linealidad, independencia, homocedasticidad ni normalidad de los residuos, lo que refuerza la confiabilidad de los resultados obtenidos (Kutner et al., 2005; Montgomery et al., 2012).

21 Regresión múltiple usando R

La regresión lineal múltiple es una técnica estadística fundamental para analizar la relación entre una variable dependiente y varias variables independientes. Su aplicación permite modelar fenómenos complejos en los que múltiples factores influyen simultáneamente en el resultado de interés. En el ámbito agronómico, comprender cómo las características del suelo afectan la producción de biomasa vegetal resulta esencial para optimizar la producción y diseñar estrategias de manejo sostenible (James et al., 2013).

21.1 Supuestos de la regresión lineal múltiple

Para que un modelo de regresión lineal múltiple sea válido, segun Field (2018) deben cumplirse los siguientes supuestos:

1. **Linealidad**: Debe existir una relación lineal entre la variable dependiente y cada una de las variables independientes.
2. **Independencia**: Los errores (residuos) del modelo deben ser independientes entre sí.
3. **Homocedasticidad**: La varianza de los errores debe ser constante para todos los valores de las variables independientes.
4. **Normalidad**: Los residuos deben seguir una distribución normal.
5. **Ausencia de multicolinealidad**: Las variables independientes no deben estar altamente correlacionadas entre sí.
6. **Ausencia de valores influyentes**: No deben existir observaciones que tengan una influencia desproporcionada en los resultados del modelo.

21.2 Contexto de la base de datos

El objetivo de este análisis es determinar el modelo de regresión lineal múltiple que mejor explica la relación entre la producción de biomasa de una especie forrajera y las características del suelo donde crece, específicamente el pH, la salinidad, el contenido de zinc (Zn) y el contenido de potasio (K). Para ello, se dispone de una base de datos con 45 observaciones, en las que se registraron los valores de biomasa (en gramos) y de las variables mencionadas para cada muestra de suelo (Trujillo Sierra, 2022).

Acceso a recursos: El script completo con los ejemplos desarrollados y la base de datos sobre biomasa están disponibles para su consulta y descarga en el siguiente repositorio: https://github.com/Ludwing-MJ/Reg_multiple_EJ

21.3 Preparación del entorno de trabajo

Siguiendo las buenas prácticas recomendadas en el manual, se inicia el análisis con la preparación del entorno de trabajo en R, asegurando la instalación y carga de los paquetes necesarios para la exploración y modelado de los datos. Se utiliza el paquete `DataExplorer` para la exploración inicial y el paquete `car` para la evaluación de supuestos del modelo.

```
# Instalación y carga de los paquetes utilizados en el análisis
if(!require("DataExplorer")) install.packages(DataExplorer)
if(!require("car")) install.packages(car)

# Importar base de datos
data <- read.csv("Biomasa.csv", sep = ";")
```

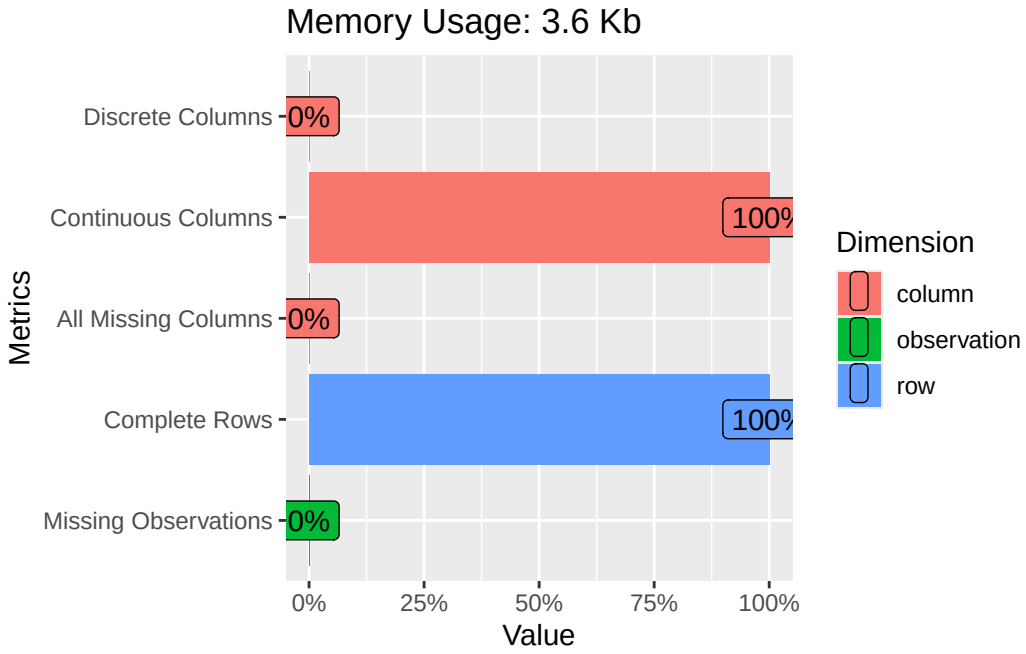
21.4 Exploración inicial de los datos

Se recomienda revisar la estructura de los datos y realizar una exploración gráfica inicial para identificar posibles datos faltantes o inconsistencias. Esta exploración es crucial para asegurar la calidad de los datos y la validez del análisis posterior.

```
# ANÁLISIS EXPLORATORIO DE LOS DATOS
# Revisar la estructura de la base de datos
str(data)
```

```
'data.frame':   45 obs. of  5 variables:
 $ Biomasa  : num  765 954 828 755 896 ...
 $ pH       : num  5 4.7 4.2 4.4 5.55 5.5 4.25 4.45 4.75 4.6 ...
 $ Salinidad: num  33 35 32 30 33 33 36 30 38 30 ...
 $ Zinc     : num  16.4 14 15.3 17.3 22.3 ...
 $ potasio  : num  1442 1299 1154 1045 522 ...
```

```
# Exploración gráfica de la base de datos
plot_intro(data)
```

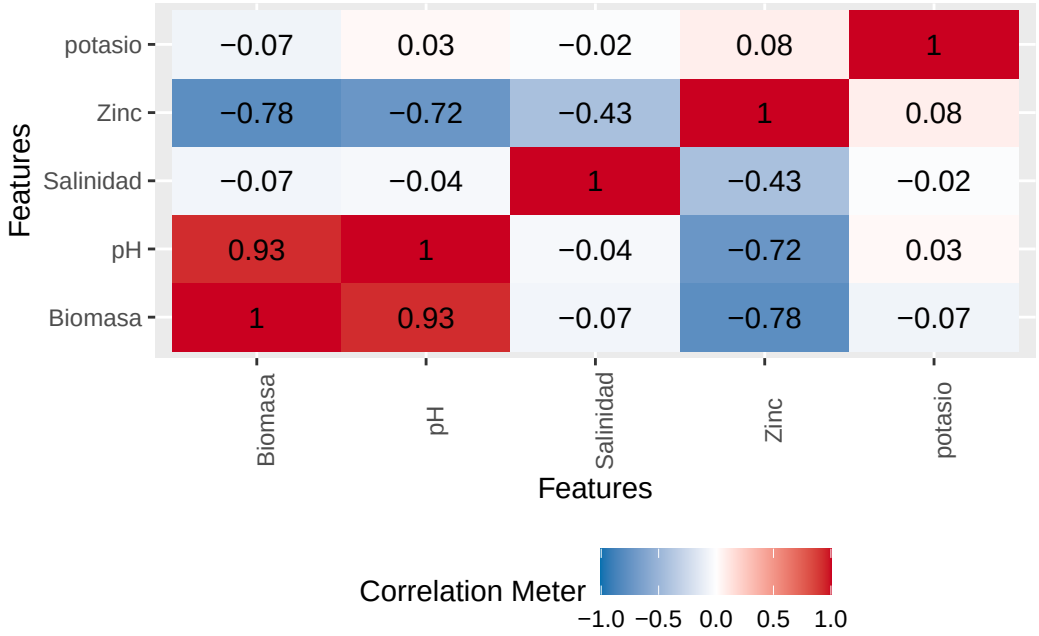


La función `str(data)` muestra la estructura de la base de datos, incluyendo el tipo de cada variable y las primeras observaciones. La función `plot_intro(data)` del paquete `DataExplorer` genera gráficos que resumen la distribución de las variables y la presencia de datos faltantes. En este caso, no se detectaron datos faltantes, lo que facilita el análisis posterior.

21.5 Análisis exploratorio y matriz de correlaciones

Antes de ajustar el modelo, se analiza la relación entre las variables mediante una matriz de correlaciones. Este paso permite identificar relaciones lineales y posibles problemas de multicolinealidad entre los predictores, lo que es crucial para la interpretación y estabilidad del modelo (James et al., 2013).

```
# Elaboración de una matriz de correlaciones
plot_correlation(data)
```



La interpretación de la matriz de correlaciones debe centrarse en detectar correlaciones elevadas entre variables independientes, ya que esto podría indicar multicolinealidad, afectando la precisión de las estimaciones de los coeficientes.

21.6 Ajuste del modelo de regresión lineal múltiple

Se ajusta un modelo inicial considerando todas las variables predictoras:

```
# ANALISIS DE REGRESION LINEAL MULTIPLE
# Uso de Attach para no colocar el signo de dolar con todas las variables
attach(data)
# Elaborar un modelo empleando todas las variables como predictoras
modelo_completo <- lm(Biomasa ~ pH + Zinc +
                        potasio + Salinidad, data = data)
# Revisar el modelo
summary(modelo_completo)
```

```
Call:
lm(formula = Biomasa ~ pH + Zinc + potasio + Salinidad, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-294.07  -88.87   -9.48   88.08  387.22

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 1492.43223  453.60067   3.290 0.002095 **
```

```
pH           262.88941   33.73382   7.793 1.51e-09 ***
Zinc         -28.96756    5.66425  -5.114 8.23e-06 ***
potasio       -0.11501    0.08191  -1.404 0.168007
Salinidad    -33.49121    8.65231  -3.871 0.000392 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 158.9 on 40 degrees of freedom
Multiple R-squared:  0.9231,    Adjusted R-squared:  0.9154
F-statistic:   120 on 4 and 40 DF,  p-value: < 2.2e-16
```

La función `attach(data)` permite acceder a las variables de la base de datos directamente por su nombre, sin necesidad de usar el operador `$`. La salida del modelo (`summary(modelo_completo)`) proporciona los coeficientes estimados, errores estándar, valores t y p-valores para cada predictor. Un p-valor menor a 0.05 indica que la variable correspondiente tiene un efecto estadísticamente significativo sobre la biomasa (en este ejemplo únicamente la variable potasio no tiene un efecto estadísticamente significativo sobre la biomasa), manteniendo constantes las demás variables. El coeficiente de determinación ajustado (Adjusted R-squared) indica el porcentaje de variabilidad de la biomasa explicado por el modelo siendo este del 91.54% para el modelo incial.

21.7 Selección del modelo mediante el método paso a paso (stepwise)

Para seleccionar el modelo más parsimonioso y evitar el sobreajuste, se emplea el método paso a paso (stepwise), utilizando el criterio de información bayesiano (BIC) como criterio de selección. El BIC penaliza la complejidad del modelo de manera más estricta que el AIC, favoreciendo modelos más simples y robustos, especialmente recomendable cuando se dispone de un número limitado de observaciones, como en este caso (Burnham & Anderson, 2004).

```r
# Aplicar stepwise (BIC como criterio)
modelo_final <- step(modelo_completo, direction = "both",
                     k = log(nrow(data)), trace = 0)

summary(modelo_final)
```

```
Call:
lm(formula = Biomasa ~ pH + Zinc + Salinidad, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-363.52 -107.13    8.46   78.41  398.30

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept) 1501.972    458.892   3.273 0.002165 **
pH           255.014     33.656   7.577 2.56e-09 ***
Zinc         -30.403      5.637  -5.394 3.14e-06 ***
Salinidad    -34.791      8.704  -3.997 0.000261 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 160.8 on 41 degrees of freedom
Multiple R-squared:  0.9193,    Adjusted R-squared:  0.9134
F-statistic: 155.6 on 3 and 41 DF,  p-value: < 2.2e-16
```

El modelo final seleccionado incluye únicamente las variables que contribuyen significativamente a explicar la variabilidad de la biomasa, considerando la penalización por complejidad. Siendo el modelo final el siguiente:

```
Biomasa = 1501.97 + 255.01 (pH) - 30.40 (Zinc) - 34.79 (Salinidad)
```

Este modelo tiene un coeficiente de determinación ajustado del 91.34% y tanto el modelo como todos sus estimadores son estadísticamente significativo lo que implica que este es un modelo bastante confiable.

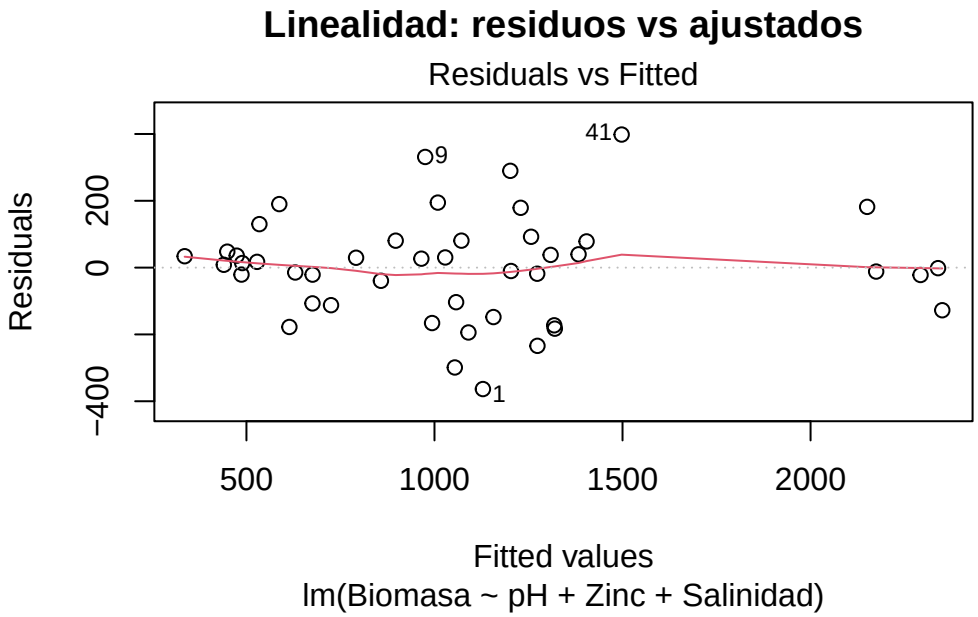
21.8 Evaluación de los supuestos del modelo final

La validez inferencial de un modelo de regresión lineal múltiple descansa en el cumplimiento simultáneo de seis supuestos clásicos: linealidad, independencia de los errores, homocedasticidad, normalidad de los residuos, ausencia de multicolinealidad y ausencia de valores influyentes. A continuación se desarrolla cada uno de estos supuestos y se discuten posibles acciones correctivas cuando se detectan violaciones (Field, 2018; James et al., 2013; Venables & Ripley, 2002).

21.8.1 Linealidad

La forma funcional que vincula la variable dependiente con cada predictor debe ser lineal. Si la relación real es curvilínea, las estimaciones serán sesgadas y los intervalos de confianza perderán validez (James et al., 2013).

```
# Gráfico de residuos versus valores ajustados
plot(modelo_final, 1,
     main = "Linealidad: residuos vs ajustados")
```



Interpretación: Un patrón en forma de "U" o " " indica no linealidad. En ese caso se recomiendan transformaciones (log, raíz cuadrada), introducción de términos polinómicos o métodos no paramétricos (James et al., 2013). Para este caso la ausencia de patrones sistemáticos en el gráfico anterior indica que la relación es aproximadamente lineal.

21.8.2 Independencia de los errores

Los residuos deben ser incorrelados entre sí. La autocorrelación es habitual en series temporales o datos espaciales y conduce a varianzas de los coeficientes subestimadas (Field, 2018).

```
# Prueba de Durbin-Watson
dwt(modelo_final)
```

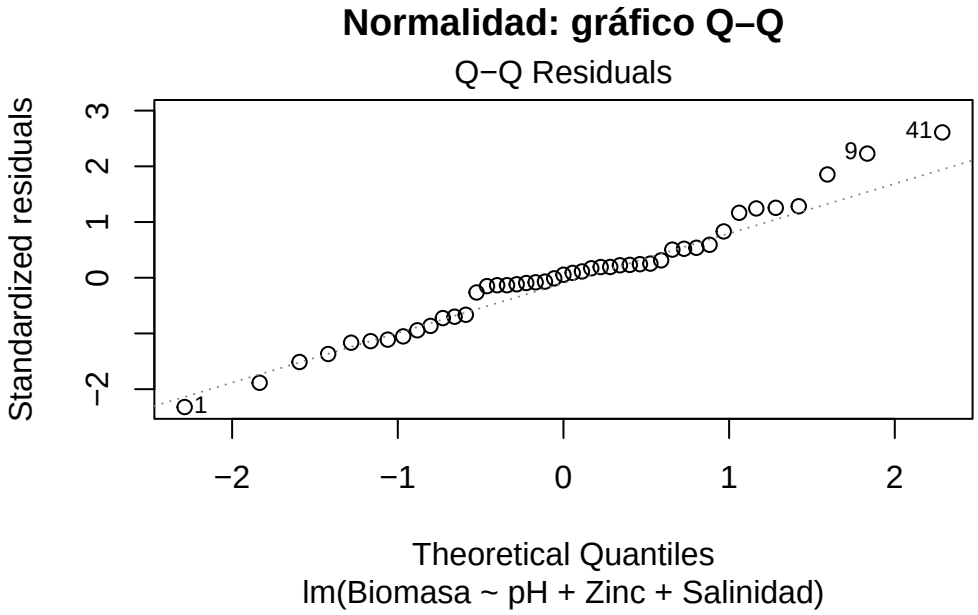
```
 lag Autocorrelation D-W Statistic p-value
   1       0.1225464      1.598719   0.118
 Alternative hypothesis: rho != 0
```

Interpretación: Un estadístico D-W próximo a 2 indica independencia; valores < 1.5 sugieren autocorrelación positiva (Field, 2018). Si se detecta autocorrelación, puede recurrirse a modelos con estructura de correlación (GLS), regresión con errores AR(1) o incluir variables de tiempo/espacio (Venables & Ripley, 2002). También se puede interpretar únicamente el p-valor que para este ejemplo al ser mayor que el nivel de significancia (0.12>0.05) indica se cumple con el supuesto de independencia de los errores.

21.8.3 Normalidad de los residuos

Se requiere que los residuos sigan una distribución normal para garantizar la validez de los contrastes t y F (Venables & Ripley, 2002). La normalidad es menos crítica con tamaños muestrales grandes, pero se verifica por completitud.

```
# Q-Q plot
plot(modelo_final, 2,
     main = "Normalidad: gráfico Q-Q")
```




```
# Test de Shapiro-Wilk
shapiro.test(residuals(modelo_final))
```

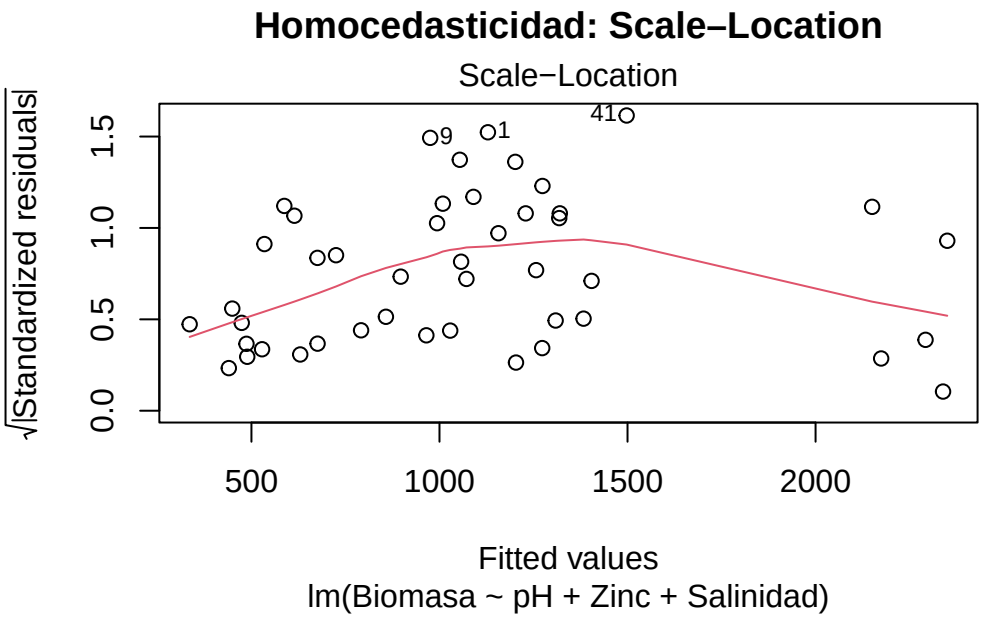
```
    Shapiro-Wilk normality test

data:  residuals(modelo_final)
W = 0.97443, p-value = 0.4146
```

Interpretación: Un p-valor > 0.05 indica que no se rechaza la normalidad (Cleveland, 1993). Si el supuesto falla, suele bastar con transformaciones o con el uso de intervalos de confianza bootstrap, menos sensibles a la no normalidad (James et al., 2013).

21.8.4 Homocedasticidad

La varianza de los errores debe permanecer constante a lo largo del rango de valores predichos. La heterocedasticidad provoca estimaciones ineficientes y p-valores poco fiables (James et al., 2013).

```r
# Gráfico Scale-Location (√|residuos| vs ajustados)
plot(modelo_final, 3,
     main = "Homocedasticidad: Scale-Location")
```




```r
# Test de Breusch-Pagan
ncvTest(modelo_final)
```

```
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 0.7243543, Df = 1, p = 0.39472
```

Interpretación: Un p-valor > 0.05 respalda la homocedasticidad. Cuando se viola este supuesto, son útiles las transformaciones de la respuesta (log), la estimación con errores estándar robustos (`vcovHC`) o el empleo de modelos ponderados (Weighted Least Squares) (Field, 2018).

21.8.5 Ausencia de multicolinealidad

La colinealidad incrementa la varianza de las estimaciones y dificulta la interpretación de los coeficientes (Field, 2018).

```r
# Revision de ausencia de multicolinealidad
# Calcular el VIF para cada variable independiente
vif(modelo_final)
```

```
       pH      Zinc Salinidad
 3.035319  3.703030  1.784158
```

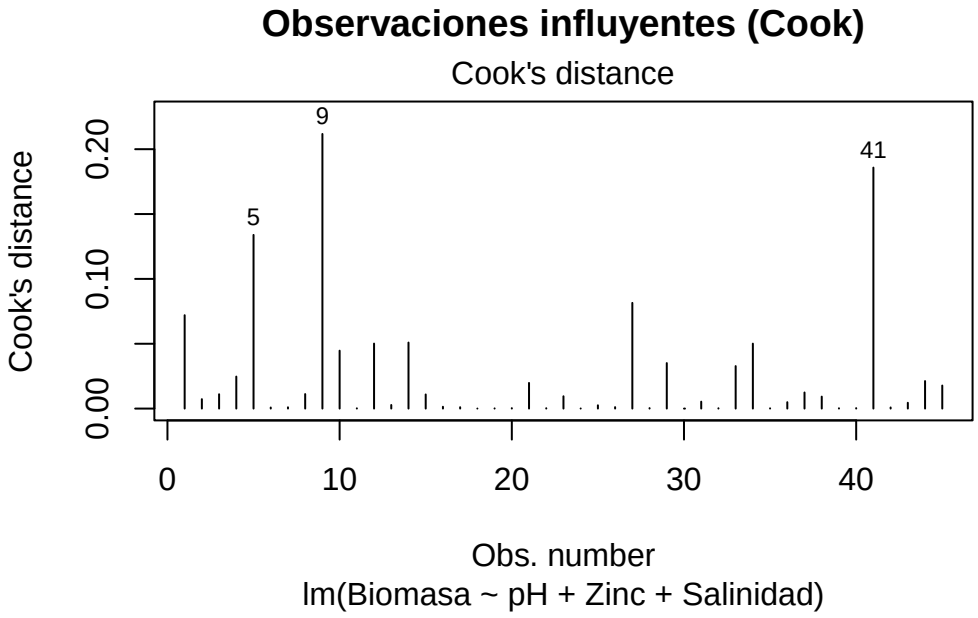
Criterios: VIF < 5 se considera aceptable; valores entre 5 y 10 requieren atención; > 10 indican colinealidad severa (James et al., 2013).

Interpretación: Las tres variables independientes presentan un VIF < 5, esto se considera aceptable y sugiere ausencia de colinealidad entre las variables independientes. Entre las estrategias de mitigación cuando no se cumple el supuesto se incluyen eliminar o combinar variables, centrar/estandarizar predictores o utilizar componentes principales (PCR).

21.8.6 Ausencia de valores influyentes

Observaciones con alta influencia pueden distorsionar drásticamente el ajuste (Field, 2018).

```
# Gráfico de la distancia de Cook
plot(modelo_final, 4,
     main = "Observaciones influyentes (Cook)")
```




```
# Identificación numérica de casos influyentes
which(cooks.distance(modelo_final) >
      4 / nrow(data))
```

```
 5  9 41
 5  9 41
```

Regla práctica: Valores de Cook $> 4/n$ sugieren influencia notable (James et al., 2013). Ante detección, se recomienda comprobar errores de registro, justificar su inclusión o aplicar métodos robustos (RLM).

Interpretación: En este modelo se identificaron los registros 5, 9 y 41 como observaciones potencialmente influyentes, de acuerdo con los valores obtenidos en la distancia de Cook.

Se recomienda al lector realizar una comprobación sistemática del impacto de las observaciones influyentes en el modelo de regresión. Para ello, se sugiere ajustar el modelo dos veces: la primera, utilizando el conjunto completo de datos; y la segunda, excluyendo los registros identificados como influyentes (en este caso, los registros 5, 9 y 41). Posteriormente, se debe comparar los coeficientes estimados, los errores estándar, el coeficiente de determinación ajustado (R^2) y los valores p de ambos modelos.

Este procedimiento permite evaluar la robustez de los resultados y determinar en qué medida las observaciones influyentes afectan la interpretación y la validez del modelo. En caso de observar diferencias sustanciales entre los modelos, se recomienda discutir las posibles causas y considerar alternativas metodológicas, como la aplicación de técnicas de regresión robusta o la revisión detallada de los datos involucrados (Field, 2018; James et al., 2013).