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

P. Agr. Ludwing Isaí Marroquín Jiménez

3 abr 2025

Tabla de contenidos

Introducción

La ciencia de datos, con un enfoque estadístico, permite transformar datos en bruto en información comprensible y útil para la toma de decisiones. Este manual está diseñado para principiantes y busca introducir las herramientas fundamentales de R, un lenguaje ampliamente utilizado en el análisis estadístico y la ciencia de datos. A lo largo del texto, se abordan conceptos básicos y se presentan ejemplos prácticos que facilitan la comprensión de técnicas estadísticas esenciales. El objetivo es proporcionar una base sólida que permita aplicar R de manera efectiva en el análisis de datos, incluso sin experiencia previa.

¿En qué consiste este manual?

El presente manual, tiene como objetivo proporcionar una guía práctica y estructurada para el aprendizaje y uso del lenguaje de programación R. Este se diseñó para facilitar la comprensión de conceptos fundamentales, desde la instalación de R y RStudio hasta la manipulación de datos, visualización gráfica y exportación de resultados.

R, como herramienta de código abierto, se ha consolidado como un estándar en el análisis estadístico y la ciencia de datos, gracias a su flexibilidad, extensibilidad y capacidad para garantizar la reproducibilidad científica. A lo largo de este documento, se exploran las principales características de R y su entorno de desarrollo integrado (IDE), RStudio, destacando su utilidad en proyectos académicos y profesionales. Además, se incluyen ejemplos prácticos, recomendaciones de buenas prácticas y recursos adicionales para profundizar en el aprendizaje.

Este manual está dirigido a estudiantes, investigadores y profesionales interesados en adquirir habilidades en análisis de datos y programación estadística, con un enfoque en la claridad, la organización y la reproducibilidad.

¿Cómo esta organizado este manual?

Este manual está diseñado para guiar a principiantes en el uso del lenguaje de programación R, con un enfoque en el análisis estadístico. Su estructura sigue un enfoque progresivo, comenzando con los conceptos más básicos y avanzando hacia herramientas y técnicas más complejas. Cada sección incluye explicaciones claras, ejemplos prácticos y ejercicios que permiten aplicar lo aprendido. Además, se han incorporado recomendaciones y buenas prácticas para facilitar el aprendizaje y fomentar la reproducibilidad en los análisis.

El contenido se organiza en los siguientes capítulos principales:

1. **Introducción a R y RStudio**: Se presenta qué es R, sus características principales y cómo instalar tanto R como RStudio. También se explica cómo configurar el entorno de trabajo.
2. **Conceptos básicos de R**: Se abordan los fundamentos del lenguaje, como la creación de objetos, tipos de datos y operadores.
3. **Manipulación de datos**: Se introduce el uso de herramientas como `dplyr` y `tidyr` para transformar y organizar datos de manera eficiente.
4. **Visualización de datos**: Se enseña a crear gráficos básicos y personalizados utilizando el paquete `ggplot2`.
5. **Exportación de resultados**: Se explica cómo guardar gráficos, tablas y otros resultados en formatos útiles para informes y presentaciones.
6. **Material de apoyo y referencias**: Se incluyen recursos adicionales para profundizar en el aprendizaje de R.

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.

Colofón

La versión en línea de este manual se encuentra disponible en https://introduccion-r-cete.vercel.app/. La fuente del manual en español está alojada en el repositorio de GitHub: https://github.com/Ludwing-MJ/introduccion_R_CETE. Este manual ha sido desarrollado utilizando *Quarto*, una herramienta diseñada para convertir archivos con extensión `.qmd` en formatos publicables como HTML, PDF y EPUB. Además, *Quarto* ofrece una interfaz visual intuitiva y amigable para el desarrollador, especialmente cuando se utiliza

desde RStudio, lo que facilita la creación de contenido reproducible, accesible y de alta calidad.

Este manual fue construido con:

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

```
Warning in system2("quarto", "-V", stdout = TRUE, env = paste0("TMPDIR=", : el
comando ejecutado '"quarto"
TMPDIR=C:/Users/Usuario/AppData/Local/Temp/RtmpiMIqJS/file55b4365b2285 -V'
tiene el estatus 1


- Session info -------------------------------------------------------------
 setting  value
 version  R version 4.4.2 (2024-10-31 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     2025-04-03
 pandoc   3.2 @ C:/Program Files/RStudio/resources/app/bin/quarto/bin/tools/ (via rmarkdo
 quarto   NA @ C:\\Users\\Usuario\\AppData\\Local\\Programs\\Quarto\\bin\\quarto.exe

- Packages -----------------------------------------------------------------
 package     * version date (UTC) lib source
 cachem        1.1.0   2024-05-16 [1] CRAN (R 4.4.2)
 cli           3.6.3   2024-06-21 [1] CRAN (R 4.4.2)
 devtools      2.4.5   2022-10-11 [1] CRAN (R 4.4.3)
 digest        0.6.37  2024-08-19 [1] CRAN (R 4.4.2)
 ellipsis      0.3.2   2021-04-29 [1] CRAN (R 4.4.3)
 evaluate      1.0.3   2025-01-10 [1] CRAN (R 4.4.2)
 fastmap       1.2.0   2024-05-15 [1] CRAN (R 4.4.2)
 fs            1.6.5   2024-10-30 [1] CRAN (R 4.4.2)
 glue          1.8.0   2024-09-30 [1] CRAN (R 4.4.2)
 htmltools     0.5.8.1 2024-04-04 [1] CRAN (R 4.4.2)
 htmlwidgets   1.6.4   2023-12-06 [1] CRAN (R 4.4.3)
 httpuv        1.6.15  2024-03-26 [1] CRAN (R 4.4.3)
 jsonlite      1.8.9   2024-09-20 [1] CRAN (R 4.4.2)
 knitr         1.49    2024-11-08 [1] CRAN (R 4.4.2)
 later         1.4.1   2024-11-27 [1] CRAN (R 4.4.3)
 lifecycle     1.0.4   2023-11-07 [1] CRAN (R 4.4.2)
 magrittr      2.0.3   2022-03-30 [1] CRAN (R 4.4.2)
 memoise       2.0.1   2021-11-26 [1] CRAN (R 4.4.2)
 mime          0.12    2021-09-28 [1] CRAN (R 4.4.0)
 miniUI        0.1.1.1 2018-05-18 [1] CRAN (R 4.4.3)
```

```
 pkgbuild      1.4.6   2025-01-16 [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 4.4.2)
 R6            2.5.1   2021-08-19 [1] CRAN (R 4.4.2)
 Rcpp          1.0.14  2025-01-12 [1] CRAN (R 4.4.2)
 remotes       2.5.0   2024-03-17 [1] CRAN (R 4.4.3)
 rlang         1.1.5   2025-01-17 [1] CRAN (R 4.4.2)
 rmarkdown     2.29    2024-11-04 [1] CRAN (R 4.4.2)
 rstudioapi    0.17.1  2024-10-22 [1] CRAN (R 4.4.2)
 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 4.4.2)
 xfun          0.50    2025-01-07 [1] CRAN (R 4.4.2)
 xtable        1.8-4   2019-04-21 [1] CRAN (R 4.4.3)

 [1] C:/Users/Usuario/AppData/Local/R/win-library/4.4
 [2] C:/Program Files/R/R-4.4.2/library

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

Parte 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 ampliamente utilizado en el análisis estadístico, la ciencia de datos y la visualización científica. Fue desarrollado por Ross Ihaka y Robert Gentleman en 1996 con el propósito de ofrecer una herramienta poderosa y flexible para realizar análisis reproducibles y visualizaciones de alta calidad. Desde su creación, R se ha consolidado como una de las herramientas más populares en las comunidades científica, académica y profesional (Ihaka & Gentleman, 1996).

1.1.1 Características principales de R

Especialización en análisis estadístico: R está diseñado para realizar análisis estadísticos complejos, desde pruebas básicas como t-tests y ANOVA hasta modelos avanzados como regresión, análisis multivariado y aprendizaje automático.

Visualización de datos: Incluye herramientas para crear gráficos de alta calidad y personalizables. Con paquetes como `ggplot2`, es posible generar visualizaciones avanzadas que permiten explorar y comunicar patrones en los datos de manera efectiva.

Lenguaje de código abierto: R es un software de código abierto, lo que lo hace gratuito y accesible para todos. Esto fomenta la colaboración y el desarrollo continuo por parte de una comunidad global de usuarios y desarrolladores.

Extensibilidad mediante paquetes: R cuenta con una amplia colección de paquetes (más de 19,000 disponibles en CRAN hasta 2023) que amplían sus capacidades. Estos paquetes permiten realizar tareas específicas, como análisis genómico, minería de texto, modelado espacial y más (R Core Team, 2023).

Reproducibilidad: R promueve la investigación reproducible al permitir que los análisis se documenten en scripts, asegurando que los resultados puedan ser replicados por otros investigadores o por el mismo usuario en el futuro.

Interoperabilidad: R puede integrarse con otros lenguajes de programación como Python, C++ y SQL, y es compatible con múltiples formatos de datos, como CSV, Excel, JSON y bases de datos relacionales.

1.1.2 ¿Por qué es especial R?

R no solo es una herramienta para realizar cálculos estadísticos, sino que también es un entorno completo para la manipulación de datos, la creación de gráficos y la automatización de flujos de trabajo. Su flexibilidad y capacidad de personalización lo convierten en una opción ideal para investigadores, analistas de datos y profesionales de diversas disciplinas.

Además, R es altamente extensible gracias a su comunidad activa, que constantemente desarrolla nuevos paquetes y recursos. Esto lo convierte en una herramienta en constante evolución, capaz de adaptarse a las necesidades cambiantes de la ciencia y la industria.

En resumen, R es mucho más que un lenguaje de programación: es una plataforma integral para el análisis de datos, la visualización y la investigación reproducible, lo que lo hace indispensable en el ámbito académico y profesional.

1.2 ¿Qué es RStudio?

RStudio es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) diseñado específicamente para trabajar con el lenguaje de programación R. Este entorno proporciona una interfaz amigable y herramientas avanzadas que optimizan el flujo de trabajo en R, facilitando tanto el análisis estadístico como la visualización de datos. Su diseño está orientado a mejorar la productividad y la reproducibilidad en proyectos de análisis de datos, investigación científica y desarrollo de aplicaciones (Allaire et al. 2022).

1.2.1 Características principales de RStudio

Interfaz intuitiva y organizada: RStudio divide su interfaz en paneles que permiten acceder fácilmente a diferentes herramientas y funciones. Estos paneles incluyen:

1. **Editor de scripts**: Donde se escribe y edita el código R.
2. **Consola**: Para ejecutar comandos y ver resultados en tiempo real.
3. **Environment/History**: Muestra los objetos creados en la sesión actual y el historial de comandos ejecutados.
4. **Plots/Files/Packages/Help**: Panel multifuncional para visualizar gráficos, gestionar archivos, instalar paquetes y acceder a documentación.

Sistema de proyectos: RStudio permite organizar el trabajo en proyectos, lo que facilita la gestión de archivos, scripts y datos relacionados con un análisis específico. Cada proyecto tiene su propio directorio de trabajo, lo que mejora la organización y la reproducibilidad.

Compatibilidad con múltiples formatos: RStudio soporta la importación y exportación de datos en diversos formatos, como CSV, Excel, html y bases de datos SQL. Además, permite trabajar con gráficos interactivos y aplicaciones web mediante paquetes como `shiny` y `plotly`.

Integración con paquetes y extensiones: RStudio facilita la instalación y el uso de paquetes de R, como `ggplot2` para gráficos, `dplyr` para manipulación de datos y `tidyr` para transformación de datos. También permite gestionar dependencias y actualizar paquetes de manera sencilla.

Soporte multiplataforma: RStudio está disponible para sistemas operativos Windows, macOS y Linux, lo que lo hace accesible para una amplia variedad de usuarios.

Personalización y extensibilidad: Los usuarios pueden personalizar la apariencia y el comportamiento de RStudio, como cambiar temas, atajos de teclado y configuraciones de paneles. Además, se pueden integrar herramientas externas, como Git para control de versiones.

1.2.2 Beneficios de usar RStudio

Eficiencia: Su diseño permite realizar tareas de análisis de datos de manera más rápida y organizada.

Reproducibilidad: Las herramientas integradas, como R Markdown y el sistema de proyectos, garantizan que los análisis puedan ser replicados fácilmente.

Accesibilidad: Su interfaz gráfica es ideal tanto para principiantes como para usuarios avanzados.

Flexibilidad: Permite trabajar con datos, gráficos, modelos estadísticos y aplicaciones interactivas en un solo entorno.

1.3 Reproducibilidad y replicabilidad en la investigación científica

El 64% de los investigadores enfrenta dificultades para replicar estudios previos debido a una documentación insuficiente (Baker, 2016).

Problemática: En herramientas como Excel o Infostat, los cálculos suelen realizarse en celdas ocultas y los gráficos se ajustan manualmente. Esto dificulta la replicación exacta de los análisis, incluso por parte del mismo autor después de un tiempo.

Solución: el uso de R permite documentar cada paso del análisis mediante scripts, lo que garantiza que los procedimientos puedan ser replicados y reinterpretados en el futuro.

Figura 1.1: Un script de R puede compararse con una receta, ya que permite ser seguido paso a paso para reproducir el mismo análisis. De manera similar a una receta que puede ser reinterpretada al modificar los ingredientes, un script puede adaptarse para realizar un análisis diferente utilizando nuevos datos.

1.3.1 Reproducibilidad

La reproducibilidad se define como la capacidad de obtener los mismos resultados utilizando los mismos datos y métodos que se emplearon en un análisis original. Este concepto es esencial en la investigación científica, ya que permite verificar y validar los resultados de un estudio. Según el informe de las *National Academies of Sciences, Engineering, and Medicine* (2019), la reproducibilidad garantiza que los resultados puedan ser replicados por otros investigadores o por el mismo autor en el futuro, siempre que se disponga de los datos y métodos originales.

1.3.1.1 Características clave de la reproducibilidad:

Uso de los mismos datos: Los datos originales deben estar disponibles y ser accesibles para que otros puedan replicar el análisis.

Métodos documentados: Es necesario que los pasos, herramientas y configuraciones utilizadas en el análisis estén claramente documentados.

Resultados consistentes: Al repetir el análisis con los mismos datos y métodos, los resultados deben ser idénticos.

1.3.1.2 Importancia de la reproducibilidad:

Transparencia: Permite que otros investigadores comprendan cómo se obtuvieron los resultados.

Verificación: Facilita la identificación de errores o inconsistencias en el análisis original.

Colaboración: Proporciona una base sólida para que otros investigadores puedan construir sobre el trabajo existente.

1.3.2 Replicabilidad

La replicabilidad, por otro lado, se refiere a la capacidad de obtener resultados consistentes al realizar un estudio similar en un contexto diferente, utilizando nuevos datos o métodos ligeramente modificados. Este concepto evalúa la generalización de los hallazgos y su aplicabilidad en diferentes escenarios. De acuerdo con las *National Academies of Sciences, Engineering, and Medicine* (2019), la replicabilidad es fundamental para evaluar la robustez y la aplicabilidad de los resultados científicos en nuevos contextos.

1.3.2.1 Características clave de la replicabilidad:

Nuevos datos: Se utilizan datos diferentes a los del estudio original, pero que representan un contexto similar.

Métodos adaptados: Los métodos pueden ser ajustados o modificados para adaptarse a las características de los nuevos datos.

Resultados consistentes: Los hallazgos deben ser coherentes con los del estudio original, aunque no necesariamente idénticos.

1.3.2.2 Importancia de la replicabilidad:

Generalización: Evalúa si los resultados del estudio original son aplicables en otros contextos o poblaciones.

Credibilidad científica: Refuerza la confianza en los hallazgos al demostrar que no son producto de circunstancias específicas.

Avance del conocimiento: Permite explorar nuevas aplicaciones o extensiones de los hallazgos originales.

1.3.3 Beneficios de la adopción de R para la Ciencia

Transparencia: El código generado en R es accesible para revisión por pares, lo que fomenta la transparencia en los análisis (The Turing Way Community, 2023).

Eficiencia: Los métodos desarrollados en R pueden ser reutilizados en nuevos estudios, lo que optimiza los recursos y el tiempo (Gentleman & Temple Lang, 2007).

Credibilidad: El uso de R facilita el cumplimiento de los principios FAIR (Findable, Accessible, Interoperable, Reusable), promoviendo la gestión adecuada de los datos científicos (Wilkinson et al., 2016).

2 Instalación y configuración

2.1 Descarga de R y RStudio

Para comenzar a trabajar con R y RStudio, es necesario descargar ambos programas. R es un lenguaje de programación y entorno computacional, mientras que RStudio es un Entorno de Desarrollo Integrado (IDE) que facilita el uso de R.

2.1.1 Descarga de R

Se recomienda descargar una versión estable de R para evitar problemas de compatibilidad con paquetes que aún no han sido actualizados para las versiones más recientes. Por ejemplo, en este curso se utiliza la versión R 4.4.2, ya que es una versión estable conocida.

El repositorio oficial de R se encuentra en el siguiente enlace: https://cran.r-project.org/bin/windows/base/old/.

En esta página, se puede acceder a un directorio con todas las versiones de R disponibles. Para descargar una versión específica, basta con hacer clic en el nombre de la versión deseada. Esto abrirá un directorio con la documentación y los archivos correspondientes. El archivo que se debe descargar tiene una terminación `-win.exe`, y al hacer clic en él, se descargará automáticamente el instalador.

2.1.2 Descarga de RStudio

Para obtener RStudio, se debe visitar la página oficial de descargas de RStudio.

En esta página, se puede descargar la versión más reciente de RStudio haciendo clic en el botón *"Download RStudio Desktop for Windows"*.

Si el dispositivo utiliza un sistema operativo diferente a Windows, en la misma página se encuentran las versiones compatibles con otros sistemas operativos.

2.2 Instalación de R y RStudio

La instalación de R y RStudio debe realizarse en un orden específico para evitar conflictos y errores:

2.2.1 Instalación de R

1. Una vez descargado el instalador de R, se debe ejecutar el archivo `.exe` y seguir las instrucciones del asistente de instalación.
2. Durante el proceso, se pueden aceptar las configuraciones predeterminadas.

2.2.2 Instalación de RStudio

1. Después de instalar R, se debe ejecutar el instalador de RStudio descargado previamente.
2. Al igual que con R, se pueden aceptar las configuraciones predeterminadas durante la instalación.

Es importante mencionar que en un mismo dispositivo pueden coexistir varias versiones de R. RStudio permite seleccionar cuál de estas versiones se utilizará en cada proyecto desde su configuración.

2.3 Configuración inicial

Una vez instalados R y RStudio, es recomendable realizar una configuración inicial para optimizar el entorno de trabajo:

2.3.1 Seleccionar la versión de R

En RStudio, se puede elegir la versión de R que se utilizará. Esto es útil si se tienen múltiples versiones instaladas. Para configurarlo, se debe ir a *Tools* > *Global Options* > *General* y seleccionar la versión deseada en el apartado *R version*.

2.3.2 Configurar la apariencia de RStudio

RStudio permite personalizar su apariencia para adaptarse a las preferencias del usuario y mejorar la experiencia de trabajo. A continuación, se describen los pasos para configurar la apariencia de la interfaz:

2.3.2.1 Cambiar el tema de la interfaz

1. En la barra de menú, se debe seleccionar **Tools > Global Options**.
2. En la ventana emergente, se accede a la pestaña **Appearance**.
3. En esta sección, es posible elegir entre diferentes temas para la interfaz, como temas claros u oscuros. Por ejemplo, los temas oscuros como *Cobalt* son recomendables para reducir la fatiga visual durante sesiones prolongadas.

4. También se pueden ajustar el tamaño y el tipo de fuente para facilitar la lectura del código, según las preferencias del usuario.

2.3.2.2 Configurar el panel de trabajo

La interfaz de RStudio está organizada en cuatro paneles principales: el editor de scripts, la consola, el entorno/archivos y los gráficos/ayuda. Estos paneles pueden reorganizarse según las necesidades del usuario.

1. Desde la barra de menú, se selecciona **Tools > Global Options > Pane Layout**.
2. En esta sección, se ajusta la disposición de los paneles. Por ejemplo, se puede colocar el editor de scripts en la parte superior izquierda y la consola en la parte inferior para facilitar el acceso.
3. Una vez realizada la configuración, se guardan los cambios para aplicar la nueva disposición.

2.3.2.3 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.

1. Se accede a **Tools > Global Options > Code > Display**.
2. En esta sección, se marca la casilla **Show line numbers** para activar la numeración de líneas.

2.3.2.4 Activar el ajuste de línea (Word Wrap)

Cuando se trabaja con líneas de código largas, habilitar el ajuste de línea evita que el texto se desborde fuera del área visible.

1. En la barra de menú, se selecciona **Tools > Global Options > Code > Display**.
2. Se activa la opción **Soft-wrap R source files** para habilitar el ajuste de línea.

2.4 Organización de proyectos

La organización de proyectos en RStudio es fundamental para mantener un flujo de trabajo eficiente y reproducible. A continuación, se presentan las mejores prácticas para organizar proyectos:

2.4.1 Crear un proyecto en RStudio

1. En la barra de menú, selecciona **File > New Project**.
2. Elige entre las opciones disponibles:
    a. **New Directory**: Crea un proyecto desde cero en una nueva carpeta.
    b. **Existing Directory**: Convierte una carpeta existente en un proyecto de RStudio.
    c. **Version Control**: Clona un repositorio de Git para trabajar en un proyecto versionado.
3. Configura el nombre y la ubicación del proyecto, y haz clic en **Create Project**.

2.4.2 Establecer un directorio de trabajo

El directorio de trabajo es la carpeta donde R buscará automáticamente los archivos y guardará los resultados.

Para establecer el directorio de trabajo, utiliza la función:

```
setwd("ruta/del/directorio")
```

Alternativamente, en un proyecto de RStudio, el directorio de trabajo se configura automáticamente al abrir el proyecto.

2.4.3 Uso de archivos `.Rproj`

El archivo `.Rproj` es el núcleo del proyecto en RStudio. Al abrir este archivo, se cargará automáticamente el entorno de trabajo, el directorio y las configuraciones específicas del proyecto.

2.4.4 Beneficios de la organización de proyectos

Reproducibilidad: Facilita que otros usuarios (o el propio usuario en el futuro) comprendan y reproduzcan el análisis.

Eficiencia: Reduce el tiempo perdido buscando archivos o configurando rutas manualmente.

Colaboración: Mejora la comunicación y el trabajo en equipo al mantener una estructura clara y consistente.

Personalizar la apariencia y organizar los proyectos en RStudio no solo mejora la comodidad visual, sino que también optimiza el flujo de trabajo, permitiendo que el usuario se enfoque en el análisis de datos de manera más eficiente y profesional.

Parte II

Conceptos básicos de R

3 Primeros pasos en R

Iniciar el trabajo en R y RStudio puede parecer desafiante al principio, pero con una guía clara y organizada, el proceso se vuelve mucho más accesible. Esta sección está diseñada para acompañar al usuario en sus primeros pasos dentro de este entorno de programación, abordando desde la creación de scripts hasta la comprensión de los objetos básicos en R. Un script en RStudio no solo es un espacio para escribir código, sino también una herramienta esencial para documentar y reproducir análisis de datos de manera eficiente.

Además, se explorarán las mejores prácticas para guardar y organizar archivos, lo que garantiza un flujo de trabajo ordenado y reproducible. También se introducirá el concepto de objetos en R, fundamentales para almacenar y manipular datos, junto con una descripción de los principales tipos de objetos (numéricos, texto, factores y lógicos). Este conocimiento inicial permitirá al usuario sentar las bases del conocimiento para utilizar R para realizar análisis estadístico y visualizaciones más adelante.

3.1 Creación de scripts en RStudio

Para comenzar a trabajar en RStudio, es fundamental crear un script donde se pueda escribir y guardar el código. Un script es un archivo que contiene las instrucciones que se ejecutarán en R. Existen dos métodos principales para crear un script en RStudio:

1. **Manualmente:** El usuario debe desplegar la pestaña 'File' en la barra de opciones superior, seleccionar 'New File' y luego elegir 'R Script'.
2. **Utilizando atajos de teclado:** Se puede presionar la combinación de teclas Ctrl+Shift+N para crear un nuevo script de manera rápida.

Una vez creado, el script se convierte en un espacio de trabajo donde se desarrollarán los análisis y se documentarán los pasos realizados.

3.2 Guardado y organización de archivos

Es importante guardar el script desde el inicio para evitar la pérdida de trabajo. Para ello, se debe seleccionar la opción "Save As..." en la pestaña "File", lo que permitirá elegir la ubicación y el nombre del archivo. Se recomienda utilizar nombres descriptivos y consistentes para facilitar la identificación de los archivos en el futuro.

Además, se sugiere organizar los archivos en carpetas específicas para cada proyecto. Esto incluye para proyectos demasiado grandes separar los scripts, datos y resultados en directorios bien definidos. Si antes de crear nuestro script creamos un proyecto de R `.Rproj` al que ya sea le asignamos una carpeta como directorio de trabajo, al guardar nuestros scripts

pertenecientes a ese proyecto lo debemos hacer en la misma carpeta donde se encuentra el archivo `.Rproj`.

La organización adecuada no solo ahorra tiempo, sino que también mejora la reproducibilidad del análisis, permitiendo que otros colaboradores o al propio usuario en el futuro entender y replicar el trabajo realizado.

3.2.1 Recomendaciones para la organización de archivos

La organización de archivos en RStudio es fundamental para garantizar un flujo de trabajo eficiente y reproducible. Así como en una cocina bien organizada se separan los utensilios, ingredientes frescos y especias en cajones etiquetados, en RStudio es necesario establecer un sistema claro para guardar scripts, datos y resultados. Un proyecto desorganizado puede compararse con una alacena caótica: se pierde tiempo buscando archivos y se corre el riesgo de cometer errores. Para evitar estos problemas, se recomienda seguir las siguientes prácticas:

3.2.1.1 Utilizar nombres descriptivos y consistentes

Es importante asignar nombres que describan claramente el contenido del archivo y que sigan un formato uniforme. Por ejemplo:

Para scripts: *analisis1.R* o *20231015_analisis_rendimiento_maiz.R*.

Para datos: *datos_finales.xlsx* o *datos_suelo_antigua_2023.csv*.

3.2.1.2 Evitar espacios y caracteres especiales

Los nombres de los archivos y los objetos en R no deben incluir espacios ni caracteres especiales, ya que esto puede generar problemas al trabajar con ellos en R. En su lugar, se recomienda usar guiones bajos (_) o guiones medios (-). Por ejemplo:

En lugar de *analisis suelo.R*, usar *analisis_suelo.R*.

3.2.1.3 Incluir fechas en un formato estandarizado

Incorporar fechas en los nombres de los archivos es una práctica útil para identificar rápidamente versiones o actualizaciones. Se sugiere utilizar el formato estándar *YYYY-MM-DD*. Por ejemplo:

2023-10-15_importacion_datos.R.

3.2.1.4 Crear una carpeta específica para cada proyecto

Es recomendable organizar los archivos de cada proyecto en carpetas separadas. Esto facilita el acceso y asegura que todos los elementos relacionados con un proyecto estén agrupados en un solo lugar.

Adoptar estas prácticas permite que cualquier colaborador, o incluso uno mismo en el futuro, pueda encontrar y comprender rápidamente los archivos necesarios. La organización no es un detalle menor, sino la base para trabajar de manera eficiente y garantizar la reproducibilidad de los análisis.

3.3 Introducción a los objetos en R

En R, todo se maneja como un objeto. Un objeto es una estructura que almacena datos y tiene atributos como nombre y tipo. Los objetos son fundamentales para trabajar en R, ya que permiten almacenar y manipular información de manera eficiente.

Para crear un objeto, se utiliza un operador de asignación, que puede ser '=' o '<-'. Sin embargo, se recomienda el uso de '<-' por ser el estándar en la comunidad de R. Por ejemplo, para asignar el valor 10 a un objeto llamado 'x', se puede escribir:

```
# Creación del primer objeto en R
x <- 10
```

3.4 Tipos de objetos en R (numéricos, texto, factores, lógicos)

3.4.1 Objetos Numéricos

Los objetos numéricos almacenan datos como números enteros o decimales. Son útiles para representar variables cuantitativas como edad, altura o peso. Por ejemplo, se puede crear un objeto numérico de la siguiente manera:

```
# Creación de objetos numéricos
edad <- 21
altura_m <- 1.70
peso_lb <- 150
```

Nota: En R, el símbolo numeral (#) se utiliza para incluir comentarios dentro del código. Las líneas que comienzan con este símbolo no son ejecutadas por el programa, ya que su propósito es servir como anotaciones que explican o documentan el código. Esto resulta especialmente útil para describir los pasos realizados, aclarar la funcionalidad de ciertas líneas o facilitar la comprensión del script a otros usuarios o al propio autor en el futuro.

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. Por ejemplo, para almacenar el nombre y color favorito de una persona, se puede escribir el siguiente codigo:

```
# Creación de objetos tipo carácter
nombre <- "Juan"
color_favorito <- "azul"
```

3.4.3 Objetos de Tipo Factor

Los objetos de tipo factor se utilizan para almacenar variables categóricas con niveles definidos, como escalas o categorías. Por ejemplo, para representar el estado civil y sexo de una persona, se puede escribir:

```
# Creación de objetos tipo factor
estado_civil <- factor("soltero")
sexo <- factor("masculino")
```

3.4.4 Objetos Lógicos

Los objetos lógicos almacenan valores de tipo TRUE o FALSE, que resultan de comparaciones lógicas. Por ejemplo, para verificar si la persona del ejemplo que se esta desarrollando es mayor de edad, se puede resolver con el siguiente código:

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

```
[1] TRUE
```

Estos objetos son útiles para aplicar filtros y realizar análisis condicionales en los datos.

4 Estructura de datos en R

En R, las estructuras de datos son fundamentales para organizar, almacenar y manipular información de manera eficiente. Estas estructuras permiten trabajar con diferentes tipos de datos, desde valores individuales hasta colecciones complejas, facilitando el análisis estadístico y la visualización. Entre las principales estructuras de datos en R se encuentran los vectores, data frames y listas, cada una con características específicas que las hacen adecuadas para distintas tareas.

4.1 Vectores

Un vector es la estructura de datos más básica y fundamental en R. Es una colección ordenada de elementos del mismo tipo, como números, texto o valores lógicos. Los vectores son unidimensionales, lo que significa que los datos se almacenan en una sola fila o columna. Por ejemplo, un vector puede representar una lista de edades, nombres o resultados de un experimento.

En R, los vectores son esenciales porque muchas otras estructuras de datos, como los data frames, están construidas a partir de ellos. Todos los elementos de un vector deben ser del mismo tipo de dato, lo que garantiza consistencia en las operaciones realizadas sobre ellos.

4.1.1 Creación de vectores

Para crear un vector, se utiliza la función `c()`, donde los elementos se separan por comas. Por ejemplo:

```r
# Creación de un vector numérico
edades <- c(17, 20, 18, 25)

# Creación de un vector de texto
nombres <- c("Juan", "Ana", "Luis", "María")

# Creación de un vector lógico
mayores_de_edad <- edades >= 18
```

4.1.2 Manipulación de vectores

Los vectores permiten realizar operaciones matemáticas, lógicas y de manipulación de datos. Por ejemplo, se pueden filtrar elementos, realizar cálculos o combinar vectores. Algunas operaciones comunes incluyen:

1. **Acceso a elementos específicos:** Utilizando índices entre corchetes `[]`

```
# Acceder al primer elemento del vector
edades[1]
```

```
[1] 17
```

2. **Filtrado de elementos:** Aplicando condiciones lógicas.

```
# Filtrar edades mayores a 20
edades[edades > 20]
```

```
[1] 25
```

3. **Combinación de vectores:**

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

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

4.2 Data frames

Un data frame es una estructura de datos tabular en R, similar a una hoja de cálculo o una tabla de base de datos. Organiza los datos en filas y columnas, donde cada columna es un vector y puede contener un tipo de dato diferente (por ejemplo, números, texto o factores). Cada fila representa una observación, y cada columna representa una variable.

Los data frames son ideales para trabajar con datos estructurados, como encuestas, experimentos o bases de datos, ya que permiten realizar análisis estadísticos y visualizaciones de manera eficiente. Además, son compatibles con muchas funciones y paquetes en R, lo que los convierte en una de las estructuras más utilizadas.

4.2.1 Creación de data frames

Para crear un data frame, se utiliza la función `data.frame()`, combinando vectores de igual longitud separados por comas. Por ejemplo:

```r
# Creación de un data frame con vectores
datos <- data.frame(nombres,  edades,  mayores_de_edad)

# Visualización del data frame
datos
```

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

4.2.2 Ventajas de un data frame

1. **Estructura clara:** Cada fila representa una observación y cada columna una variable.
2. **Compatibilidad:** Es compatible con funciones estadísticas y de visualización.
3. **Flexibilidad:** Permite almacenar diferentes tipos de datos en columnas.

4.2.3 Manipulación de data frames

Los data frames pueden manipularse fácilmente utilizando funciones como `filter()`, `select()` o `mutate()` del paquete `dplyr`. También se puede acceder a columnas específicas utilizando el operador `$`. Por ejemplo:

```r
# Acceso a una columna
datos$nombres
```

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

```r
# 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
```

Los data frames son una herramienta poderosa para organizar y analizar datos, y su versatilidad los hace indispensables en el trabajo con R.

4.3 Listas

Las listas son estructuras de datos más flexibles que los data frames, ya que pueden contener elementos de diferentes tipos y longitudes, como vectores, data frames, matrices o incluso funciones. Para crear una lista, se utiliza la función `list()`:

```
# Creación de una lista
mi_lista <- list(
  nombres = c("Juan", "Ana"),
  edades = c(18, 20),
  datos_completos = datos
)
```

Las listas son útiles para almacenar resultados complejos o datos heterogéneos. Los elementos de una lista pueden accederse mediante índices o nombres:

```
# Acceso a un elemento por nombre
mi_lista$nombres
```

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

```
# Acceso a un elemento por índice
mi_lista[[1]]
```

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

4.4 Comparación entre data frames y listas

Característica	Data Frame	Lista
Estructura	Tabular (filas y columnas)	Colección de objetos heterogéneos
Tipos de datos	Columnas con tipos diferentes	Elementos de cualquier tipo
Uso principal	Análisis estadístico y visualización	Almacenamiento de resultados complejos
Acceso a elementos	Por columnas o índices	Por nombres o índices

Ambas estructuras son fundamentales en R, y su elección depende del tipo de datos y del análisis que se desee realizar. En análisis estadístico habitualmente se emplean los data frames para almacenar datos y las listas se emplean para almacenar los resultados de pruebas y análisis.

5 Importación de datos

La importación de datos es uno de los primeros pasos esenciales en cualquier análisis estadístico. En R, los datos pueden provenir de diversas fuentes, como archivos CSV, Excel o incluso páginas web en formato HTML. La capacidad de importar datos de manera eficiente y reproducible permite trabajar con grandes volúmenes de información sin necesidad de manipularlos manualmente. Además, establecer un directorio de trabajo adecuado facilita la organización y asegura que los scripts sean portables y reproducibles en diferentes entornos.

En esta sección, se explicará cómo importar datos desde archivos CSV, Excel y HTML, así como la configuración del directorio de trabajo cuando se trabaja en un script que no pertenece a ningún proyecto `.Rproj`.

5.1 Importación de archivos CSV

Los archivos CSV (Comma-Separated Values) son una de las formas más comunes de almacenar datos tabulares. Estos archivos son ligeros, universales y fáciles de manejar en R. Para importar un archivo CSV, se utiliza la función `read.csv()`, que permite leer el contenido del archivo y almacenarlo como un data frame.

5.1.1 Pasos para importar de un archivo CSV

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

# Visualizar los primeros registros del data frame
head(datos)
```

`header = TRUE`: Indica que el archivo tiene una fila de encabezado con los nombres de las columnas.

`sep = ","`: Especifica que los valores están separados por comas (el argumento de esta parte de la función puede variar si el archivo utiliza un separador distinto como por ejemplo **";"**).

Nota: Es importante asegurarse de que el archivo esté en el directorio de trabajo o proporcionar la ruta completa.

5.1.2 Ejemplo de importación de un archivo CSV

En 2002, se llevó a cabo un estudio en la Universidad de San Carlos de Guatemala, en el que se recopilaron datos de 460 estudiantes de diversas facultades. Esta base de datos, disponible para su descarga en formato CSV a través del siguiente enlace, incluye una amplia variedad de variables, lo que la convierte en un recurso ideal para los ejercicios prácticos de este manual. A lo largo del documento, se utilizará esta base de datos para aplicar las herramientas y conceptos desarrollados. Para realizar el siguiente ejemplo, es necesario que el usuario descargue el archivo y lo guarde en la carpeta correspondiente al proyecto en curso.

```
# Importar el archivo CSV con los datos
datos <- read.csv("datos_estudiantes.csv", header = TRUE, sep = ",")

# Visualizar los primeros registros del data frame
head(datos)
```

La función `head()` en R se utiliza para visualizar las primeras filas de un data frame, vector, matriz u otro objeto de datos. Su propósito principal es proporcionar una vista rápida de los datos, permitiendo al usuario verificar la estructura, los nombres de las columnas y los primeros valores sin tener que imprimir todo el conjunto de datos.

Nota: En el ejemplo para importar el archivo solamente se colocó el nombre del archivo (incluyendo su extensión) sin necesidad de colocar la ruta exacta del archivo y R interpretó automáticamente que debía importar el archivo con ese nombre ubicado en la misma carpeta que el script. Esto se debe a que al estar trabajando en un proyecto con un directorio de trabajo establecido no hay necesidad de colocar la ruta de los archivos que se quieran importar siempre que estos se encuentren en la misma carpeta.

5.2 Importación de archivos Excel

Los archivos Excel son ampliamente utilizados para almacenar datos en hojas de cálculo. En R, se puede importar este tipo de archivos utilizando el paquete `readxl`, que permite leer datos de archivos `.xlsx` sin necesidad de convertirlos previamente a otro formato.

5.2.1 Pasos para importar un archivo Excel

1. Instalar y cargar el paquete `readxl`.

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

Explicación de la linea de código anterior

a. `require("nombre del paquete")`: Esta función intenta cargar el paquete especificado. Si el paquete está instalado, lo carga en la sesión actual y devuelve `TRUE`. Si el paquete no está instalado, devuelve `FALSE`.

b. `!require("nombre del paquete")`: El signo de exclamación (`!`) niega el resultado de la función `require()`. Por lo tanto, esta expresión será `TRUE` si el paquete no está instalado o no puede ser cargado.

c. `if (...) install.packages("nombre del paquete")`: Esta es una estructura condicional. Si la condición entre paréntesis es `TRUE` (es decir, si el paquete no está instalado o no puede ser cargado), entonces se ejecuta `install.packages("nombre del paquete")`, que descarga e instala el paquete desde el repositorio de CRAN.

2. Importar el archivo Excel

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

# Visualizar los primeros registros del data frame
head(datos_excel)
```

Explicación de la linea de código anterior

a. **`sheet = "Hoja1"`**: Especifica la hoja del archivo que se desea importar.

b. **`col_names = TRUE/FALSE`**: Indica si la primera fila contiene los nombres de las columnas

5.2.2 Ejemplo de importación de un archivo Excel

En este ejemplo, se utilizará la misma base de datos empleada en el caso de importación de archivos CSV. Esta base de datos, ahora en formato Excel, está disponible para su descarga a través de este enlace. Al igual que en el ejemplo anterior para realizar este ejemplo, es necesario que el usuario descargue el archivo y lo guarde en la carpeta correspondiente al proyecto en curso.

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

# Importar un archivo Excel
datos_excel <- read_excel("datos_estudiantes_2002.xlsx",
                          sheet = "datos",
                          col_names = TRUE)

# Visualizar los primeros registros del data frame
head(datos_excel)
```

5.3 Directorio de trabajo o Working Directory

El directorio de trabajo en R puede considerarse como el "punto de partida" en un mapa: es la carpeta donde el software buscará automáticamente los archivos necesarios (datos, scripts) y guardará los resultados generados. Cuando se trabaja con un script que no forma parte de un proyecto en RStudio, establecer un directorio de trabajo es esencial para garantizar la organización y la reproducibilidad del análisis. Si no se define correctamente, se corre el riesgo de perder tiempo buscando rutas manualmente y de que el código deje de funcionar al mover el proyecto a otra ubicación o computadora.

5.3.1 Razones para establecer un directorio de trabajo

1. **Reproducibilidad:** Permite que el código funcione en cualquier computadora sin depender de rutas absolutas como `C:/Usuario/MiPC/...`.
2. **Organización:** Reduce errores como "archivo no encontrado" al mantener todos los elementos (datos, scripts, resultados) en una estructura clara y accesible.
3. **Eficiencia:** Facilita el acceso a los archivos, ya que no es necesario escribir rutas completas, solo los nombres de los archivos.

5.3.2 Configuración automática del directorio de trabajo

Cuando se trabaja con scripts independientes, se puede utilizar la siguiente línea de código para establecer automáticamente el directorio de trabajo como la carpeta donde está guardado el script:

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

# Establecer el directorio de trabajo
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
```

5.3.2.1 Explicación del código

1. **`if (...) install.packages("rstudioapi")` :** Esta estructura condicional instala y carga el paquete rstudioapi necesario para que la linea de código que estable el directorio de trabajo pueda funcionar.
2. **`rstudioapi::getActiveDocumentContext()$path`:** Obtiene la ruta completa del script actual (por ejemplo, `C:/proyecto/scripts/analisis.R`).
3. **`dirname()`:** Extrae la carpeta que contiene el script (por ejemplo, `C:/proyecto/scripts/` → `C:/proyecto/`).
4. **`setwd()`:** Establece esa carpeta como el directorio de trabajo.

5.3.2.2 Beneficios de establecer un directorio de trabajo

1. **Portabilidad:** Si se mueve toda la carpeta del proyecto a otra ubicación, el código seguirá funcionando sin necesidad de ajustes.
2. **Automatización:** No es necesario modificar manualmente las rutas al compartir el script con otros usuarios.

5.3.2.3 Consecuencias de no establecer un directorio de trabajo

1. **Errores frecuentes:** R buscará archivos en una ubicación predeterminada (como la carpeta "Documentos"), lo que puede generar errores si los datos no están allí.
2. **Código no reproducible:** Si otra persona ejecuta el script, será necesario modificar manualmente todas las rutas para que funcione.

Recomendación: Antes de ejecutar la línea automática, es importante guardar el script, ya que R necesita conocer su ubicación para establecer el directorio de trabajo correctamente. Además, se puede verificar el directorio actual utilizando la función `getwd()`.

6 Operadores en R

En R, los operadores son herramientas fundamentales que permiten realizar cálculos, comparaciones y manipulaciones de datos. Son el equivalente a las herramientas básicas de un taller, como destornilladores o martillos, que se combinan para construir soluciones más complejas. Los operadores en R se utilizan para realizar operaciones matemáticas, evaluar condiciones lógicas, manipular datos y realizar asignaciones. Su correcta comprensión es esencial para aprovechar al máximo las capacidades del lenguaje en el análisis estadístico y la programación.

En R, los operadores se clasifican en diferentes categorías según su función. A continuación, se describen los principales tipos de operadores disponibles en el lenguaje:

Tipo de Operador	Ejemplo	Descripción
Aritméticos	`+`, `-`, `*`, `/`	Realizan operaciones matemáticas básicas como suma, resta, multiplicación, etc.
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

Los operadores de asignación se utilizan para crear objetos y almacenar valores en ellos. En R, los operadores más comunes son `<-` y `=`. Aunque ambos cumplen la misma función, el uso de `<-` es el estándar recomendado en la comunidad de R, ya que evita conflictos con otros operadores lógicos.

6.1.1 Ejemplo práctico

```
# Asignación de valores a objetos
x <- 10          # Asignar el valor 10 al objeto x
y = 20           # Asignar el valor 20 al objeto y (menos recomendado)
```

```r
# Uso de objetos
suma <- x + y    # Resultado: 30
```

Nota: Aunque = puede ser utilizado para asignar valores, su uso no es recomendado en contextos profesionales debido a posibles confusiones con el operador lógico de igualdad (==).

6.2 Operadores aritméticos

Los operadores aritméticos permiten realizar operaciones matemáticas básicas y avanzadas. Son fundamentales para trabajar con datos numéricos y realizar cálculos en análisis estadísticos. Estos operadores operan sobre valores numéricos y devuelven resultados numéricos.

6.2.1 Lista de operadores aritméticos

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

6.2.2 Ejemplo práctico

En este ejemplo, se observa cómo los operadores aritméticos pueden ser utilizados tanto para cálculos simples como para operaciones más específicas, como obtener el cociente y el residuo de una división. Estas operaciones son útiles en contextos como la creación de nuevas variables derivadas o el análisis de datos numéricos.

```r
# Ejemplo práctico del uso de operadores aritméticos
# Operaciones básicas
resultado_suma <- 5 + 3        # Resultado: 8
resultado_resta <- 10 - 4      # Resultado: 6
resultado_mult <- 6 * 2        # Resultado: 12
resultado_div <- 15 / 3        # Resultado: 5
resultado_pot <- 2 ^ 3         # Resultado: 8

# División entera y residuo
cociente <- 17 %/% 5           # Resultado: 3
residuo <- 17 %% 5             # Resultado: 2
```

6.3 Operadores lógicos

Los operadores lógicos permiten realizar comparaciones y evaluaciones condicionales. Son esenciales para la toma de decisiones en el código, como filtrar datos o establecer reglas condicionales. Estos operadores trabajan con valores lógicos (`TRUE` o `FALSE`) y se utilizan para evaluar condiciones.

6.3.1 Lista de operadores lógicos

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`

6.3.2 Ejemplo práctico

```r
# Ejemplo práctico del uso de operadores lógicos
# Comparaciones simples
edad <- 25
es_mayor <- edad > 18                # Resultado: TRUE
es_menor <- edad < 30                # Resultado: TRUE
es_igual <- edad == 25               # Resultado: TRUE
es_diferente <- edad != 20           # Resultado: TRUE

# Operaciones lógicas compuestas
peso_Kg <- 70
altura <- 1.75
imc <- peso_Kg / (altura^2)

sobrepeso <- imc >= 25 & imc < 30    # Evaluación de sobrepeso
peso_normal <- imc >= 18.5 & imc < 25 # Evaluación de peso normal
```

En este ejemplo, se observa cómo los operadores lógicos pueden ser utilizados para evaluar condiciones simples y compuestas. Por ejemplo, se calcula el índice de masa corporal (IMC) y se evalúa si el valor corresponde a un rango de peso normal o sobrepeso.

6.4 Operadores de Manipulación de Datos

Los operadores de manipulación de datos permiten acceder, seleccionar y modificar elementos dentro de estructuras de datos como vectores, listas o data frames. Estos operadores son esenciales para trabajar con datos organizados y realizar análisis estadísticos.

6.4.1 Principales operadores de manipulación de datos

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

6.4.2 Ejemplo práctico

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

# Acceder al primer elemento
primer_elemento <- vector[1]       # Resultado: 10

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

# Acceder a una columna
columna_edad <- data$edad      # Resultado: c(25, 30, 22)

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

En este ejemplo, se observa cómo los operadores de manipulación de datos permiten acceder a elementos específicos dentro de estructuras de datos. Esto es especialmente útil para filtrar, transformar y analizar datos en R.

7 Funciones en R

Las funciones son uno de los pilares fundamentales de la programación en R. Constituyen bloques de código que encapsulan una serie de instrucciones diseñadas para realizar tareas específicas. Estas permiten automatizar procesos, reducir la repetición de código y mejorar la legibilidad de los scripts. En este capítulo, se explorará qué son las funciones, cómo se utilizan y cómo se pueden crear funciones personalizadas para resolver problemas específicos.

7.1 Definición y características de las funciones en R

7.1.1 ¿Qué es una función?

Una función en R es un objeto que toma uno o más valores de entrada (llamados argumentos), realiza una serie de operaciones con ellos y devuelve un resultado. Las funciones son esenciales para estructurar el código de manera eficiente y reutilizable.

Cada función en R tiene tres componentes principales:

1. **Nombre**: Es el identificador que se utiliza para llamar a la función.
2. **Argumentos**: Son los valores de entrada que la función necesita para realizar sus operaciones.
3. **Cuerpo**: Es el conjunto de instrucciones que define lo que la función hace.

7.1.2 Tipos de funciones

En R, existen dos tipos principales de funciones:

7.1.2.1 Funciones predefinidas (built-in)

Las funciones predefinidas son aquellas que ya vienen incluidas en R o en paquetes adicionales. Estas funciones están diseñadas para realizar tareas comunes, como cálculos matemáticos, operaciones estadísticas, manipulación de datos y visualización.

Ejemplos de funciones predefinidas:

`mean()`: Calcula la media aritmética de un conjunto de datos.

```r
# Ejemplo del uso de la función mean
datos <- c(1, 2, 3, 4, 5)
media <- mean(datos)
media  # Resultado:
```

```
[1] 3
```

`sum()`: Calcula la suma de los elementos de un vector.

```r
# Ejemplo del uso de la función sum
suma <- sum(datos)
suma  # Resultado:
```

```
[1] 15
```

`sd()`: Calcula la desviación estándar.

```r
# Ejemplo del uso de la función sd
desviacion <- sd(datos)
desviacion  # Resultado:
```

```
[1] 1.581139
```

`summary()`: Proporciona un resumen estadístico de un conjunto de datos.

```r
# Ejemplo del uso de la función summary
resumen <- summary(datos)
resumen  # Resultado:
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      1       2       3       3       4       5 
```

Estas funciones son ampliamente utilizadas y no requieren que el usuario las defina, ya que están disponibles de forma predeterminada.

7.1.2.2 Funciones personalizadas (user-defined)

Las funciones personalizadas son aquellas que el usuario crea para realizar tareas específicas que no están cubiertas por las funciones predefinidas. Estas funciones son útiles cuando se necesita automatizar procesos repetitivos o realizar cálculos complejos que no están disponibles en las funciones estándar.

Ejemplo de una función personalizada:

Se puede crear una función para calcular el área de un círculo dado su radio:

```r
# Función para calcular el area de un circulo
calcular_area_circulo <- function(radio) {
  area <- pi * radio^2
  return(area)
}

# Uso de la función
area <- calcular_area_circulo(5)
area  # Resultado:
```

```
[1] 78.53982
```

En este caso, el usuario define la lógica de la función, especifica los argumentos necesarios (`radio`) y utiliza la función para realizar cálculos.

7.1.3 Diferencias entre funciones predefinidas y personalizadas

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()`	`calcular_area_circulo()`, funciones específicas del usuario
Reutilización	Reutilizables en cualquier script	Reutilizables si se definen en el entorno o se guardan en un archivo

7.2 Usos y beneficios de las funciones en R

Las funciones en R ofrecen múltiples beneficios que las convierten en herramientas indispensables para cualquier programador o analista de datos. Entre los principales usos y ventajas se encuentran:

1. **Reutilización de código**: Una vez que se define una función, esta puede ser utilizada en diferentes partes de un proyecto o incluso en otros proyectos, evitando la repetición de código.
2. **Modularidad**: Las funciones permiten dividir problemas complejos en partes más pequeñas y manejables, lo que facilita la organización del código.
3. **Legibilidad**: Al encapsular operaciones complejas dentro de funciones, el código se vuelve más fácil de leer y entender.

4. **Automatización**: Las funciones permiten automatizar tareas repetitivas, ahorrando tiempo y esfuerzo.

Ejemplo de automatización con funciones personalizadas:

Supongamos que se necesita calcular el área de varios círculos con diferentes radios. En lugar de repetir el cálculo manualmente, se puede usar una función personalizada:

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

# Aplicar la función a cada radio
areas <- calcular_area_circulo(radios)
areas  # Resultado:
```

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

7.3 Cómo crear funciones en R: Sintaxis y ejemplos básicos

7.3.1 Sintaxis básica

La creación de funciones en R sigue una estructura sencilla:

```
nombre_funcion <- function(argumento1, argumento2, ...) {
  # Cuerpo de la función
  # Operaciones
  return(resultado)
}
```

7.3.2 Elementos clave de una función

1. **Nombre de la función**: Debe ser descriptivo y reflejar la tarea que realiza.
2. **Argumentos**: Son los valores de entrada que la función necesita. Pueden tener valores por defecto.
3. **Cuerpo de la función**: Contiene las operaciones que se ejecutan cuando se llama a la función.
4. **Valor de retorno**: Especificado con `return()`, aunque no es obligatorio. Si no se usa, la función devuelve el último valor calculado.

7.3.3 Ejemplo básico

Se puede crear una función para convertir grados Celsius a Fahrenheit:

```r
# Fucnión para convertir de grados celsius a fahrenheit
celsius_a_fahrenheit <- function(celsius) {
  fahrenheit <- (celsius * 9/5) + 32
  return(fahrenheit)
}

# Uso de la función
temperatura <- celsius_a_fahrenheit(25)
temperatura  # Resultado:
```

```
[1] 77
```

8 Paquetes en R

Los paquetes en R son una de las características más poderosas del lenguaje, ya que permiten extender sus capacidades básicas para realizar tareas específicas de manera eficiente. A continuación, se desarrolla esta sección donde se amplia esta característica tan importante de R.

8.1 ¿Qué es un paquete en R?

Un paquete en R es una colección de funciones, datos y documentación que amplía las capacidades del entorno base de R. Estos paquetes son desarrollados por la comunidad de usuarios y están diseñados para resolver problemas específicos, desde la manipulación de datos hasta la visualización avanzada o el análisis estadístico especializado.

8.1.1 Características principales de los paquetes

1. **Funciones especializadas:** Cada paquete incluye funciones diseñadas para tareas específicas, como crear gráficos, realizar análisis estadísticos o manipular datos.
2. **Documentación:** Los paquetes incluyen documentación detallada que explica cómo utilizarlos, con ejemplos prácticos.
3. **Datos de ejemplo:** Muchos paquetes incluyen conjuntos de datos que permiten practicar y entender su funcionalidad.

8.1.2 ¿Por qué usar paquetes en R?

Los paquetes son esenciales para aprovechar al máximo el potencial de R. Algunas de las razones para utilizarlos incluyen:

1. **Extensibilidad:** Permiten realizar tareas que no están disponibles en el entorno base de R.
2. **Eficiencia:** Simplifican procesos complejos, reduciendo el tiempo necesario para realizar análisis.
3. **Especialización:** Existen paquetes diseñados para áreas específicas, como la agronomía (*agricolae*), la biología (*vegan*) o la economía (*forecast*).
4. **Comunidad activa:** La comunidad de R desarrolla y mantiene una amplia variedad de paquetes, lo que garantiza su actualización y soporte.

8.2 Instalación y carga de paquetes

La instalación de paquetes en R se realiza principalmente desde CRAN (Comprehensive R Archive Network), el repositorio oficial que alberga más de 19,000 paquetes. Para instalar un paquete, se utiliza la función `install.packages()`.

8.2.1 Ejemplo básico de instalación

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

8.2.2 Carga de paquetes

Una vez instalado, un paquete debe cargarse en la sesión actual para poder utilizar sus funciones. Esto se realiza con la función `library()`.

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

Es importante destacar que la instalación de un paquete solo se realiza una vez, pero debe cargarse en cada nueva sesión de trabajo.

8.2.3 Automatización de la instalación y carga

Para garantizar que un paquete esté disponible en el entorno de trabajo, se puede utilizar la siguiente "receta mágica", que verifica si el paquete está instalado y, en caso contrario, lo instala y carga al entorno de trabajo automáticamente:

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

Esta estructura es útil para mantener el código reproducible y evitar errores al compartir scripts con otros usuarios.

8.3 Paquetes recomendados para tareas específicas

En el contexto del análisis estadístico, algunos paquetes son especialmente útiles. A continuación, se presenta una lista de paquetes recomendados, junto con una breve descripción de su funcionalidad:

8.3.1 Manipulación de datos

dplyr: Simplifica la manipulación de datos mediante funciones intuitivas para filtrar, seleccionar y resumir datos.

tidyr: Facilita la transformación de datos entre formatos ancho y largo.

8.3.2 Visualización de datos

ggplot2: Permite crear gráficos personalizados y de alta calidad basados en la gramática de gráficos.

8.3.3 Análisis estadístico básico

stats: Incluye funciones base para realizar pruebas t, ANOVA y regresiones.

Análisis estadístico agronómico

agricolae: Diseñado para análisis estadísticos en agronomía, como diseños experimentales y pruebas de comparación múltiple.

8.3.4 Ejemplo práctico: Instalación y carga de paquetes esenciales

```
# Instalación y carga de paquetes esenciales

  # Incluye ggplot2, dplyr, tidyr
if (!require("tidyverse")) install.packages("tidyverse")
  # Diseños experimentales agrícolas
if (!require("agricolae")) install.packages("agricolae")
  # Importación de archivos Excel
if (!require("readxl")) install.packages("readxl")
  # Exportación a Excel
if (!require("writexl")) install.packages("writexl")
  # Establecer directorio de trabajo automaticamente
if (!require("rstudioapi")) install.packages("rstudioapi")
```

8.4 Ejemplo práctico: Uso de paquetes en un flujo de trabajo

A continuación, se presenta un ejemplo práctico que integra diversos paquetes de R para realizar un análisis estadístico de datos. El script correspondiente a este ejemplo está disponible en el siguiente repositorio: https://github.com/Ludwing-MJ/Paquetes_Ej. Este ejercicio se basa en el ejemplo 2.2.8 del libro *Diseños y análisis de experimentos* de López y González (2016).

Contexto: Un silvicultor quiso comparar los efectos de cinco tratamientos de preparación del terreno sobre el crecimiento inicial en altura de plántulas de pino maximinoii. Dispuso

de 25 parcelas y aplicó cada tratamiento a cinco parcelas seleccionadas al azar. La plantación fue realizada manualmente y, al final de cinco años, se midió la altura de todos los pinos y se calculó la altura promedio de cada parcela. Las medidas de las parcelas (en pies) fueron las siguientes.

8.4.1 Preparación del área de trabajo

Antes de iniciar un nuevo análisis, es fundamental crear un proyecto en R para garantizar una adecuada organización y reproducibilidad del trabajo. Se recomienda seguir las pautas de guardado y organización de archivos descritas en la sección 3.2. Una vez creado y guardado el proyecto, así como el script donde se desarrollará el análisis, se procede a instalar y cargar los paquetes necesarios. A continuación, se presenta un ejemplo práctico del código correspondiente:

```
# Ejemplo práctico: Uso de paquetes
# NOTA: Antes de trabajar, es necesario crear y guardar un nuevo script.

# Instalación y carga de paquetes esenciales

# Paquete que incluye ggplot2, dplyr, tidyr
if (!require("tidyverse")) install.packages("tidyverse")

# Paquete para diseños experimentales agrícolas
if (!require("agricolae")) install.packages("agricolae")

# Paquete para la importación de archivos Excel
if (!require("readxl")) install.packages("readxl")

# Paquete para la exportación de datos a Excel
if (!require("writexl")) install.packages("writexl")

# Paquete para establecer el directorio de trabajo automáticamente
if (!require("rstudioapi")) install.packages("rstudioapi")
```

8.4.2 Importación de los datos

Para importar un archivo Excel con los datos de altura de las parcelas de pino, se utiliza el paquete readxl. Antes de realizar la importación, se establece el directorio de trabajo de manera automática utilizando el paquete rstudioapi. Esto asegura que los archivos se encuentren en la ubicación correcta para su procesamiento.

```
# Establecer directorio de trabajo
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
```

```
# Importar datos desde un archivo Excel
altura_pino <- read_excel("datos_arboles.xlsx")
```

8.4.3 Análisis de la varianza

El análisis de varianza (ANOVA) se realiza con el paquete `agricolae` para evaluar el efecto de diferentes tratamientos sobre la altura promedio de las parcelas. Además, se aplica la prueba de Tukey para realizar comparaciones múltiples entre los tratamientos.

```
# Análisis de varianza
modelo_anova <- aov(altura_ft ~ tratamiento, data = altura_pino)
summary(modelo_anova)
```

```
            Df Sum Sq Mean Sq F value  Pr(>F)
tratamiento  4  34.64    8.66   5.851 0.00276 **
Residuals   20  29.60    1.48
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

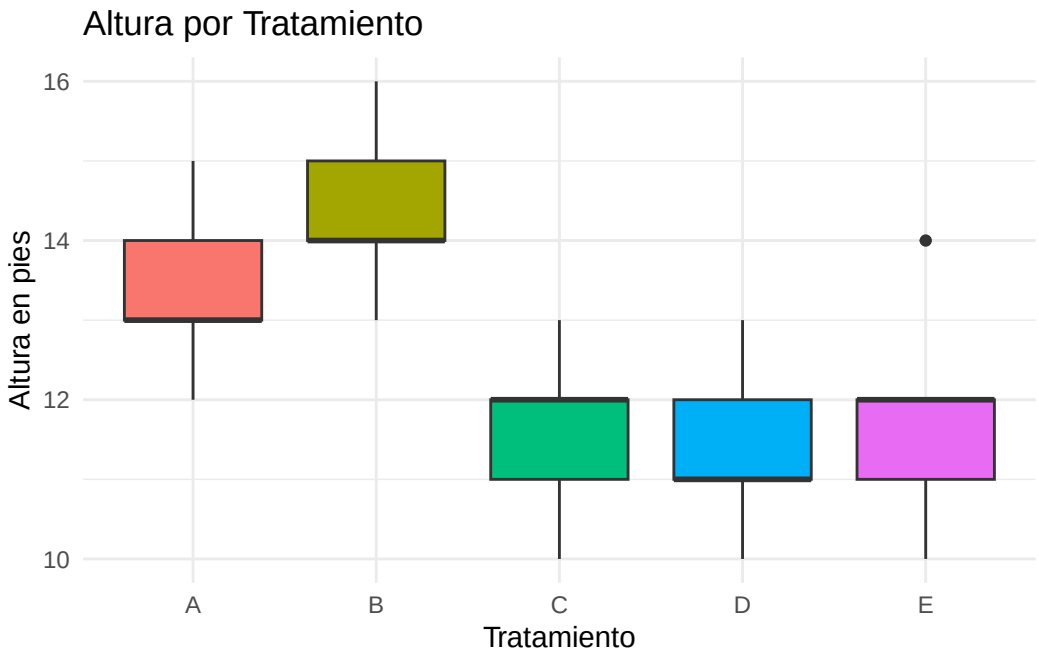
```
# Prueba de Tukey
comparacion_tukey <- HSD.test(modelo_anova, "tratamiento")
print(comparacion_tukey$groups)
```

```
  altura_ft groups
B      14.4      a
A      13.4     ab
E      11.8      b
C      11.6      b
D      11.4      b
```

8.4.4 Visualización de resultados

Para visualizar los resultados, se utiliza el paquete `ggplot2`, que permite crear un gráfico de cajas (boxplot) que muestra la distribución de la altura promedio por tratamiento. Este gráfico incluye un diseño minimalista y elimina la leyenda redundante.

```
# Visualización de resultados
ggplot(altura_pino, aes(x = tratamiento,
                        y = altura_ft,
                        fill = tratamiento)) +
  geom_boxplot() +
  labs(title = "Altura por Tratamiento",
       x = "Tratamiento",
       y = "Altura en pies")+
  theme_minimal()+ # Establece el tema del gráfico
  theme(legend.position = "none")  # Remueve la leyenda redundante
```



8.4.5 Exportación de resultados

Finalmente, los resultados de la prueba de Tukey se exportan a un archivo Excel utilizando el paquete `writexl`. Además, el gráfico generado se guarda en formato PNG con la función `ggsave`.

```
# Exportar resultados a Excel
write_xlsx(comparacion_tukey$groups,
           "resultados_tukey.xlsx",
           col_names = TRUE,
           format_headers = TRUE,
           use_zip64 = FALSE)

# Exportar gráficos
ggsave("ggplot_pino.png")
```

Parte III

Manipulación de datos

9 Manipulación de datos con dplyr y tidyr

Los paquetes **dplyr** y **tidyr** son componentes fundamentales del ecosistema tidyverse, diseñados para simplificar y optimizar la manipulación y transformación de datos en R. Estas herramientas permiten realizar tareas comunes de análisis de datos de manera eficiente, reproducible y con una sintaxis clara e intuitiva.

9.1 Introducción a los paquetes `dplyr` y `tidyr`

El paquete **dplyr** está especializado en la manipulación de datos tabulares, ofreciendo funciones específicas para realizar operaciones como filtrar filas, seleccionar columnas, crear nuevas variables y resumir datos. Su diseño está optimizado para trabajar con estructuras como data frames y tibbles, proporcionando un rendimiento superior y una sintaxis más legible en comparación con las funciones base de R.

Por otro lado, el paquete **tidyr** se centra en la reorganización de datos, facilitando la transformación entre formatos "ancho" y "largo". Estas transformaciones son esenciales para preparar los datos de manera adecuada antes de su análisis o visualización, asegurando que estén en el formato más conveniente para las herramientas de análisis.

Para ilustrar el uso de estas herramientas, en esta sección se desarrollará un ejemplo práctico que permitirá explorar las principales funciones de manipulación de datos. El script correspondiente a este ejemplo está disponible en el siguiente repositorio: Repositorio de ejemplo - Manipulación de datos.

9.2 Operaciones básicas con dplyr

9.2.1 Filtrar filas con `filter()`

La función `filter()` permite seleccionar filas de un data frame que cumplen con una o más condiciones lógicas.

Sintaxis básica:

```
filter(data, condición)
```

Ejemplo práctico: Filtrar estudiantes con un peso mayor a 65 kg.

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

# Crear un data frame de ejemplo
datos <- data.frame(
  nombre = c("Juan", "Ana", "Luis", "María"),
  edad = c(18, 22, 20, 19),
  peso = c(70, 55, 80, 60),
  altura = c(1.75, 1.60, 1.80, 1.65)
)

# Filtrar estudiantes con peso mayor a 65 kg
estudiantes_pesados <- datos %>%
  filter(peso > 65)
```

Explicación: El resultado es un data frame que incluye únicamente las filas donde la variable `peso` es mayor a 65.

9.2.2 Seleccionar columnas

La función `select()` se utiliza para extraer columnas específicas de un data frame.

Sintaxis básica:

```r
select(data, columnas)
```

Ejemplo práctico: Seleccionar las columnas `nombre` y `edad`.

```r
# Seleccionar columnas específicas
datos_reducidos <- datos %>%
  select(nombre, edad)
```

Explicación: El resultado es un data frame con solo las columnas `nombre` y `edad`, lo que puede ser útil para reducir la cantidad de datos visibles o exportar información específica.

9.2.3 Crear nuevas columnas

La función `mutate()` permite añadir nuevas columnas calculadas a un data frame.

Sintaxis básica:

```r
mutate(data, nueva_columna = expresión)
```

Ejemplo práctico: Calcular el índice de masa corporal (IMC) de los estudiantes.

```r
# Calcular el IMC
datos <- datos %>%
  mutate(IMC = peso / (altura^2))
```

Explicación: Se añade una nueva columna llamada `IMC` al data frame, calculada como el peso dividido por el cuadrado de la altura.

9.2.4 Agrupar y resumir datos

La combinación de `group_by()` y `summarize()` permite calcular estadísticas por grupo.

Sintaxis básica:

```r
  data %>%
  group_by(grupo) %>%
  summarize(resumen = función(variable))
```

Ejemplo práctico: Calcular el peso promedio por grupo de edad (mayores y menores de 20 años).

```r
# Calcular peso promedio por grupo de edad
peso_promedio <- datos %>%
  mutate(grupo_edad = ifelse(edad >= 20,
                             "Mayor o igual a 20",
                             "Menor a 20")) %>%
  group_by(grupo_edad) %>%
  summarize(peso_promedio = mean(peso))
```

Explicación: Se crea una nueva variable `grupo_edad` para clasificar a los estudiantes, y luego se calcula el peso promedio para cada grupo.

9.3 Uso de pipes (%>%) para mejorar la legibilidad del código

El operador pipe (`%>%`) es una herramienta clave en el tidyverse que permite encadenar funciones de manera legible. En lugar de anidar funciones, el pipe pasa el resultado de una función como entrada a la siguiente.

Ejemplo sin pipes:

```r
# Crear columna grupo_edad usando mutate
datos <- datos %>%
  mutate(grupo_edad = ifelse(edad >= 20,
                             "Mayor o igual a 20",
                             "Menor a 20"))
# Agrupar y resumir los datos sin pipes
resultado <- summarize(group_by(datos, grupo_edad),
                       peso_promedio = mean(peso))
```

Ejemplo con pipes:

```
# Agrupar y resumir los datos con pipes
resultado <- datos %>%
  group_by(grupo_edad) %>%
  summarize(peso_promedio = mean(peso))
```

9.3.1 Ventajas del uso de pipes

El uso de pipes (`%>%`) en R, introducido por el paquete **magrittr** y ampliamente adoptado en el ecosistema **tidyverse**, ofrece múltiples ventajas que mejoran significativamente la experiencia de programación y análisis de datos. En primer lugar, los pipes mejoran la legibilidad del código al permitir que las operaciones se encadenen de manera secuencial y lógica. Esto elimina la necesidad de anidar funciones, lo que puede resultar confuso y difícil de interpretar, especialmente en flujos de trabajo complejos. En lugar de leer el código de adentro hacia afuera, los pipes permiten que las instrucciones se lean de arriba hacia abajo, siguiendo un orden natural que refleja el proceso de análisis.

Además, los pipes facilitan la depuración y el seguimiento de cada paso del análisis. Al dividir el flujo de trabajo en pasos claros y separados, es más sencillo identificar dónde ocurre un error o verificar los resultados intermedios. Esto es especialmente útil cuando se trabaja con grandes conjuntos de datos o procesos que involucran múltiples transformaciones, ya que cada paso puede ser evaluado de forma independiente.

Por último, los pipes permiten construir flujos de trabajo complejos de manera modular. Cada operación puede considerarse como un "bloque" que se conecta al siguiente, lo que fomenta un enfoque estructurado y organizado. Esto no solo facilita la comprensión del código por parte de otros colaboradores, sino que también permite realizar modificaciones o ajustes en pasos específicos sin afectar el resto del análisis. En resumen, el uso de pipes no solo optimiza la escritura del código, sino que también mejora la claridad, la eficiencia y la reproducibilidad del análisis de datos en R.

9.4 Transformación de datos con tidyr

El paquete **tidyr** se utiliza para reorganizar datos entre formatos "ancho" y "largo", lo que es esencial para ciertos tipos de análisis y visualización.

9.4.1 Transformar datos con `pivot_longer()` y `pivot_wider()`

`pivot_longer()`: Convierte columnas en filas, útil para transformar datos de formato ancho a largo.

`pivot_wider()`: Convierte filas en columnas, útil para transformar datos de formato largo a ancho.

9.4.2 Ejemplo práctico

Supongamos que se tiene un data frame con las calificaciones de estudiantes en diferentes materias:

```
# Data frame de ejemplo
calificaciones <- data.frame(
  nombre = c("Juan", "Ana", "Luis"),
  matematicas = c(85, 90, 78),
  ciencias = c(88, 92, 80)
)
```

Transformar de formato ancho a largo con `pivot_longer()`:

```
# Transformar a formato largo
library(tidyr)
calificaciones_largo <- calificaciones %>%
  pivot_longer(cols = c(matematicas, ciencias),
               names_to = "materia",
               values_to = "calificacion")
calificaciones_largo
```

```
# A tibble: 6 x 3
  nombre materia     calificacion
  <chr>  <chr>              <dbl>
1 Juan   matematicas           85
2 Juan   ciencias              88
3 Ana    matematicas           90
4 Ana    ciencias              92
5 Luis   matematicas           78
6 Luis   ciencias              80
```

Resultado: El data frame ahora tiene una fila por cada combinación de estudiante y materia.

Transformar de formato largo a ancho con `pivot_wider()`:

```
# Transformar de vuelta a formato ancho
calificaciones_ancho <- calificaciones_largo %>%
  pivot_wider(names_from = "materia", values_from = "calificacion")
calificaciones_ancho
```

```
# A tibble: 3 x 3
  nombre matematicas ciencias
  <chr>        <dbl>    <dbl>
1 Juan            85       88
2 Ana             90       92
3 Luis            78       80
```

Explicación: Estas funciones permiten reorganizar los datos según las necesidades del análisis o la visualización.

Parte IV

Visualización de datos

10 Visualización de datos

La visualización de datos es una herramienta fundamental en el análisis estadístico, ya que permite explorar patrones, identificar relaciones y comunicar resultados de manera efectiva. En R, el paquete `ggplot2` es ampliamente utilizado debido a su flexibilidad y capacidad para generar gráficos de alta calidad. Este capítulo detalla los conceptos básicos de `ggplot2`, la creación de gráficos comunes y las opciones de personalización disponibles.

10.1 Contexto de la base de datos utilizada

En 2002, se llevó a cabo un estudio en la Universidad de San Carlos de Guatemala, en el que se recopilaron datos de 460 estudiantes de diversas facultades. Esta base de datos incluye una amplia variedad de variables, como facultad, edad, sexo, estado civil, si trabajan o no, jornada de estudio, año de ingreso, peso en libras, talla, y hábitos como fumar o consumo de alcohol. Este conjunto de datos, disponible para su descarga en formato CSV a través del siguiente enlace, será utilizado a lo largo de esta sección para aplicar las herramientas y conceptos desarrollados. El script con los ejemplos correspondientes se encuentra alojado en el siguiente repositorio: https://github.com/Ludwing-MJ/Visualiz

Nota: Para realizar los ejemplos, es necesario descargar el archivo y guardarlo en la carpeta correspondiente al proyecto en curso.

10.2 Introducción al paquete `ggplot2`

ggplot2 es una herramienta poderosa y versátil para la visualización de datos en R, que permite crear gráficos de calidad profesional con un alto nivel de personalización. Su diseño modular basado en capas facilita la adición de elementos como líneas, etiquetas y temas, adaptándose tanto a gráficos simples como a visualizaciones complejas. Además, su integración con el ecosistema **tidyverse** simplifica los flujos de trabajo al combinarse fácilmente con paquetes como **dplyr** y **tidyr** para la manipulación de datos. Es compatible con una amplia variedad de tipos de gráficos y puede extenderse mediante paquetes adicionales como **ggthemes** o **plotly** para gráficos interactivos. Finalmente, su comunidad activa y extensa documentación lo convierten en una herramienta accesible y ampliamente utilizada en análisis de datos.

10.2.1 Estructura básica de un gráfico en ggplot2

La creación de gráficos en **ggplot2** se basa en una estructura modular que permite construir visualizaciones de manera flexible y escalable. La sintaxis general para crear un gráfico en **ggplot2** es la siguiente:

```
ggplot(data = DATOS, aes(x = VARIABLE_X, y = VARIABLE_Y)) +
  GEOM_FUNCION() +
  labs(title = "Título del gráfico",
       x = "Etiqueta eje X",
       y = "Etiqueta eje Y")
```

A continuación, se describen los componentes principales de esta estructura:

1. **data:** Este argumento define el conjunto de datos que se utilizará para construir el gráfico. Debe ser un *data frame* o un objeto compatible con este formato. Es el punto de partida para cualquier visualización, ya que contiene las variables que se representarán gráficamente.

2. **aes():** La función `aes()` (abreviatura de *aesthetics*) especifica el mapeo estético, es decir, cómo las variables del conjunto de datos se asignan a los elementos visuales del gráfico. Algunos de los mapeos más comunes incluyen:

   `x`: Variable asignada al eje horizontal.

   `y`: Variable asignada al eje vertical.

   `color`: Variable que define el color de los elementos.

   `size`: Variable que define el tamaño de los elementos.

   `shape`: Variable que define la forma de los puntos (en gráficos de dispersión, por ejemplo).

   `fill`: Variable que define el color de relleno (en gráficos como barras o áreas).

3. **GEOM_FUNCION():** La función geométrica (`geom_`) define el tipo de gráfico que se desea crear. Cada tipo de gráfico tiene su propia función en **ggplot2**, como:

   `geom_point()`: Gráfico de puntos (dispersión).

   `geom_bar()`: Gráfico de barras.

   `geom_line()`: Gráfico de líneas.

   `geom_histogram()`: Histograma.

   `geom_boxplot()`: Diagrama de cajas (*boxplot*).

4. **labs():** La función `labs()` se utiliza para añadir etiquetas y títulos al gráfico. Esto incluye:

   `title`: Título principal del gráfico.

   `x`: Etiqueta del eje X.

   `y`: Etiqueta del eje Y.

   `subtitle`: Subtítulo del gráfico.

   `caption`: Texto adicional, como la fuente de los datos.

5. **Operador +:** El operador + es fundamental en **ggplot2**, ya que permite combinar diferentes capas (*layers*) en un gráfico. Cada capa puede añadir elementos adicionales, como líneas de tendencia, etiquetas o temas personalizados.
6. **Personalización adicional:** Además de los elementos básicos, **ggplot2** permite personalizar los gráficos mediante temas (`theme()`) y escalas (`scale_`). Por ejemplo, se puede cambiar el esquema de colores o ajustar el diseño general del gráfico:

10.3 Creación de gráficos básicos

A continuación, se presentan ejemplos de gráficos comunes que se pueden crear con **ggplot2**, junto con su sintaxis y una explicación detallada. Antes de proceder con la creación de gráficos, es fundamental importar la base de datos y cargar los paquetes necesarios.

10.3.1 Importación de la base de datos

Para trabajar con **ggplot2** y otros paquetes relacionados, es necesario instalar y cargar los paquetes requeridos, así como importar el conjunto de datos en formato CSV. El siguiente código muestra cómo realizar estos pasos de manera eficiente:

```
# Ejemplo práctico: Uso de paquetes para visualización
# NOTA: Antes de trabajar, es necesario crear y guardar un nuevo script.

# Instalación y carga de paquetes esenciales

# Paquete que incluye ggplot2, dplyr, tidyr
if (!require("tidyverse")) install.packages("tidyverse")

# Paquete para establecer el directorio de trabajo automáticamente
if (!require("rstudioapi")) install.packages("rstudioapi")

# Importar la base de datos
datos <- read_csv("datos_estudiantes.csv")

# Ver las primeras filas del conjunto de datos
head(datos)
```

10.3.1.1 Explicación del código

Instalación y carga de paquetes: Se utiliza la función `if (!require (...))` para verificar si los paquetes están instalados. Si no lo están, se instalan automáticamente con `install.packages()`. Automáticamente, se cargan. El paquete **tidyverse** incluye herramientas esenciales para la manipulación y visualización de datos, como **ggplot2**, **dplyr** y **tidyr**. El paquete **rstudioapi** permite establecer el directorio de trabajo automáticamente, lo que facilita la organización de los archivos.

Establecimiento del directorio de trabajo: La función `setwd()` establece el directorio de trabajo en la ubicación del script actual, utilizando `rstudioapi::getActiveDocumentContext()$path`. Esto asegura que los archivos se encuentren en la misma carpeta que el script, mejorando la reproducibilidad.

Importación de datos: La función `read_csv()` del paquete **readr** se utiliza para leer archivos CSV. La función `head()` permite visualizar las primeras filas del conjunto de datos, proporcionando una vista previa de su estructura.

10.3.1.2 Notas importantes

1. Es fundamental asegurarse de que el archivo `datos_estudiantes.csv` esté ubicado en el directorio de trabajo establecido.
2. Si el archivo no se encuentra en el directorio especificado, se generará un error. En ese caso, se puede verificar la ubicación del archivo con `getwd()` o establecer manualmente el directorio con `setwd("ruta/del/directorio")`.

Con esta configuración inicial, se puede proceder a la creación de gráficos básicos utilizando **ggplot2**.

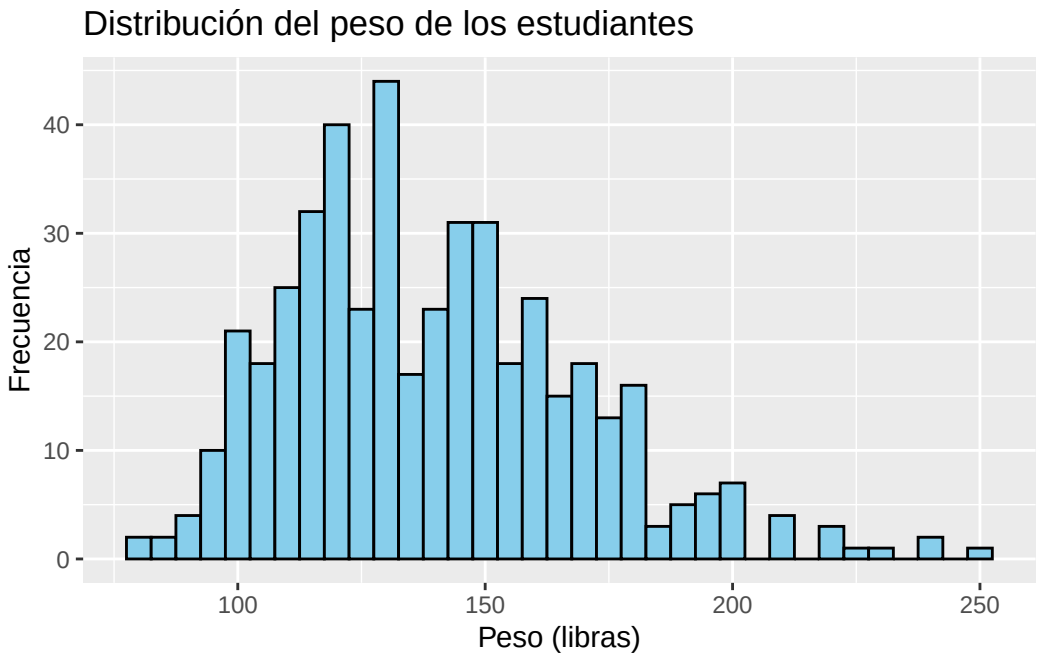
10.3.2 Histogramas

Los histogramas son gráficos que permiten visualizar la distribución de una variable numérica, mostrando cómo se agrupan los valores en intervalos específicos. Son útiles para identificar patrones, como la simetría, la dispersión, la presencia de valores atípicos o la forma general de la distribución (e.g., normal, sesgada, etc.).

10.3.2.1 Ejemplo práctico: Creación de un histograma

El siguiente código muestra cómo crear un histograma utilizando **ggplot2** para explorar la distribución del peso de los estudiantes:

```
# Ejemplo práctico: Creación de un histograma
ggplot(data = datos, aes(x = PESO_lbs)) +
  geom_histogram(binwidth = 5, fill = "skyblue", color = "black") +
  labs(title = "Distribución del peso de los estudiantes",
       x = "Peso (libras)",
       y = "Frecuencia")
```



Explicación del código:

1. **ggplot(data = datos, aes(x = PESO_lbs)):** Se define el conjunto de datos (`datos`) y se especifica la variable numérica que se desea analizar (`PESO_lbs`) dentro de la función `aes()`. Esta variable se asigna al eje X, ya que el histograma representa la frecuencia de los valores en este eje.

2. **geom_histogram():** Esta función geométrica es la encargada de crear el histograma. Cada barra representa la frecuencia de los valores que caen dentro de un intervalo específico.

3. **Argumento binwidth:** El parámetro `binwidth` define el ancho de los intervalos (o "bins") en los que se agrupan los datos. En este caso, se establece un ancho de 5 unidades, lo que significa que cada barra del histograma abarca un rango de 5 libras.

   Un valor más pequeño de `binwidth` genera más barras, proporcionando mayor detalle, mientras que un valor más grande agrupa los datos en menos barras, mostrando una visión más general.

4. **Argumentos fill y color:**

   `fill`: Define el color de relleno de las barras. En este ejemplo, se utiliza el color "skyblue" para un diseño visualmente atractivo.

   `color`: Define el color del borde de las barras, que en este caso es negro ("black"). Esto ayuda a diferenciar claramente las barras entre sí.

5. **labs():** La función `labs()` se utiliza para añadir etiquetas descriptivas al gráfico:

   `title`: Título principal del gráfico, que describe el propósito del histograma.

   `x`: Etiqueta del eje X, que indica la variable representada (en este caso, el peso en libras).

`y`: Etiqueta del eje Y, que muestra la frecuencia de los valores.

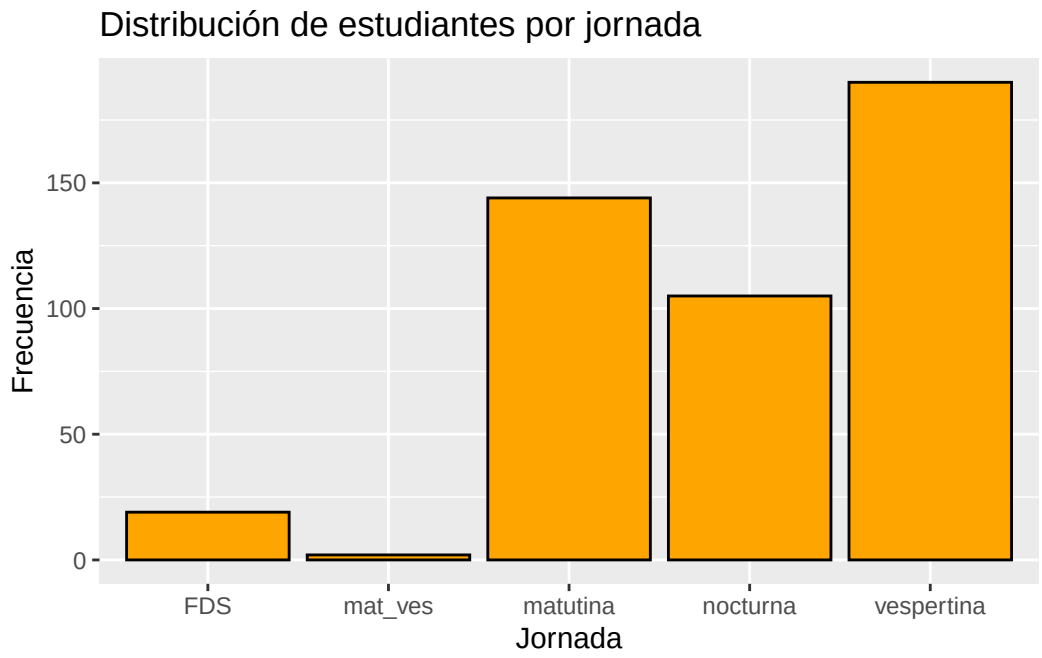
10.3.3 Gráficos de barras

Los gráficos de barras son ideales para representar datos categóricos, mostrando la frecuencia o el conteo de observaciones en cada categoría. Este tipo de gráfico es útil para comparar grupos o categorías de manera visual y sencilla.

10.3.3.1 Ejemplo práctico: Creación de un gráfico de barras

El siguiente código muestra cómo crear un gráfico de barras utilizando **ggplot2** para analizar la distribución de estudiantes según su jornada:

```
# Ejemplo práctico: Creación de un gráfico de barras
ggplot(data = datos, aes(x = JORNADA)) +
  geom_bar(fill = "orange", color = "black") +
  labs(title = "Distribución de estudiantes por jornada",
       x = "Jornada",
       y = "Frecuencia")
```



Explicación del código:

1. **ggplot(data = datos, aes(x = JORNADA))**: Se define el conjunto de datos (`datos`) y se especifica la variable categórica `JORNADA` dentro de la función `aes()`. Esta variable se asigna al eje X, ya que el gráfico de barras representa las categorías en este eje.

2. **geom_bar():** Esta función geométrica genera el gráfico de barras. Por defecto, `geom_bar()` cuenta automáticamente las observaciones en cada categoría de la variable especificada en el eje X.

3. **Argumentos fill y color**:

   `fill`: Define el color de relleno de las barras. En este ejemplo, se utiliza el color "orange" para un diseño llamativo.

   `color`: Define el color del borde de las barras, que en este caso es negro ("black"). Esto ayuda a resaltar las barras y separarlas visualmente.

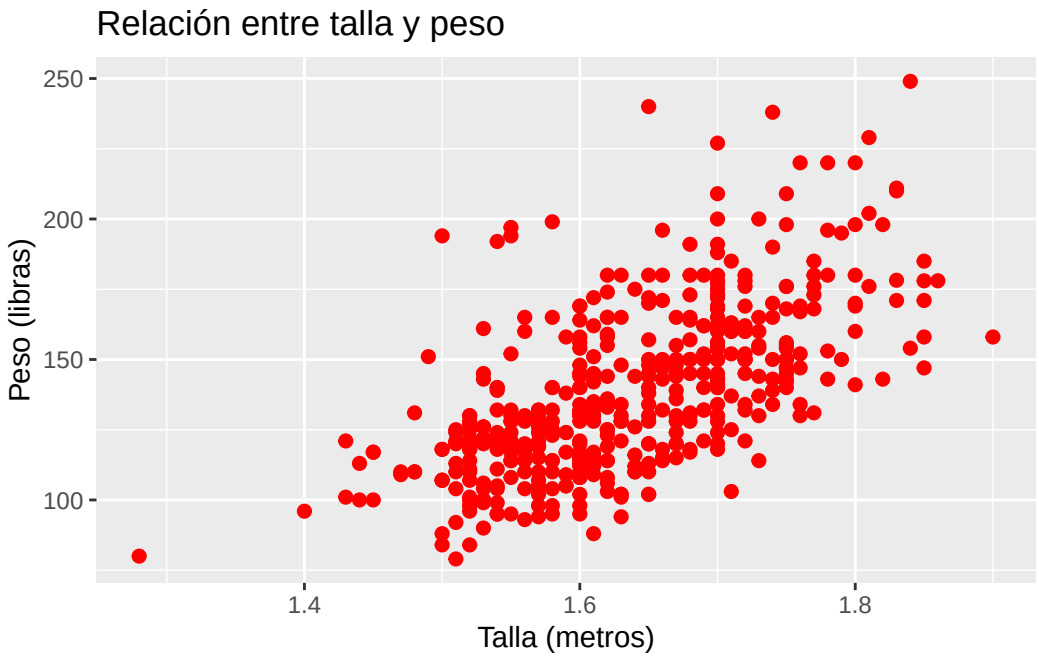
10.3.4 Gráficos de dispersión (scatterplots)

Los gráficos de dispersión son herramientas visuales que permiten analizar la relación entre dos variables numéricas. Cada punto en el gráfico representa una observación, donde la posición en el eje X corresponde al valor de una variable y la posición en el eje Y al valor de la otra. Este tipo de gráfico es útil para identificar patrones, tendencias, correlaciones y posibles valores atípicos.

10.3.4.1 Ejemplo práctico: Creación de un gráfico de dispersión

El siguiente código muestra cómo crear un gráfico de dispersión utilizando **ggplot2** para explorar la relación entre la talla y el peso de los estudiantes:

```
# Ejemplo práctico: Creación de un gráfico de dispersión
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs)) +
  geom_point(color = "red", size = 2) +
  labs(title = "Relación entre talla y peso",
       x = "Talla (metros)",
       y = "Peso (libras)")
```



1. **ggplot(data = datos, aes(x = TALLA, y = PESO_lbs))**: Se define el conjunto de datos (`datos`) y se especifican las variables numéricas que se desean analizar:

   `TALLA`: Variable asignada al eje X (talla en metros).

   `PESO_lbs`: Variable asignada al eje Y (peso en libras).

2. **geom_point()**: Esta función geométrica dibuja los puntos en el gráfico, representando cada observación del conjunto de datos.

3. **Argumentos color y size**:

   `color`: Define el color de los puntos. En este ejemplo, los puntos se dibujan en rojo ("red").

   `size`: Define el tamaño de los puntos. Aquí se establece un tamaño de 2 para que los puntos sean más visibles.

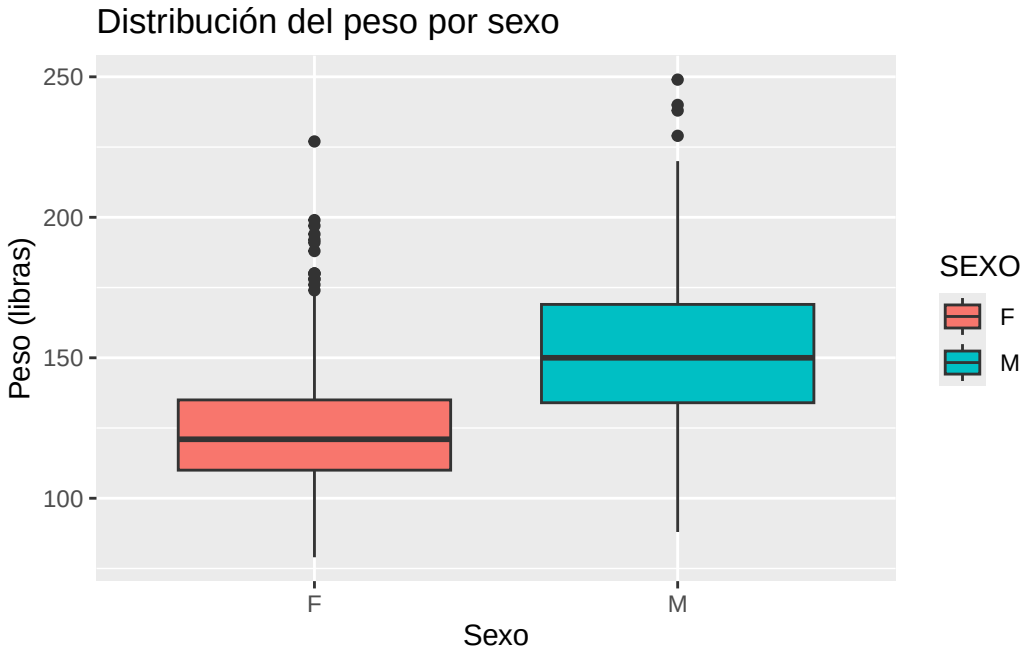
10.3.5 Boxplots

Los *boxplots* (o diagramas de caja y bigotes) son gráficos que permiten visualizar la distribución de una variable numérica y compararla entre diferentes grupos categóricos. Este tipo de gráfico es útil para identificar la mediana, la dispersión, los valores atípicos y la simetría de los datos dentro de cada grupo.

10.3.5.1 Ejemplo práctico: Creación de un boxplot

El siguiente código muestra cómo crear un *boxplot* utilizando **ggplot2** para analizar la distribución del peso de los estudiantes según su sexo:

```r
ggplot(data = datos, aes(x = SEXO, y = PESO_lbs, fill = SEXO)) +
  geom_boxplot() +
  labs(title = "Distribución del peso por sexo",
       x = "Sexo",
       y = "Peso (libras)")
```



Explicación del código

1. **ggplot(data = datos, aes(x = SEXO, y = PESO_lbs, fill = SEXO))**: Se define el conjunto de datos (`datos`) y se especifican las variables:

   `SEXO`: Variable categórica asignada al eje X, que define los grupos a comparar.

   `PESO_lbs`: Variable numérica asignada al eje Y, cuya distribución se analiza dentro de cada grupo.

   `fill`: Argumento opcional que asigna un color de relleno diferente a cada grupo basado en la variable `SEXO`.

2. **geom_boxplot()**: Esta función geométrica genera el *boxplot*. Cada caja representa la distribución de la variable numérica dentro de un grupo categórico.

10.3.5.2 Elementos clave de un boxplot

Un *boxplot* incluye los siguientes elementos visuales:

1. **Caja (box)**: Representa el rango intercuartílico (IQR), que abarca del primer cuartil (Q1) al tercer cuartil (Q3).
2. **Línea dentro de la caja**: Indica la mediana de los datos.

3. **Bigotes (whiskers)**: Extienden los valores hasta 1.5 veces el IQR desde los cuartiles Q1 y Q3.
4. **Puntos fuera de los bigotes**: Representan valores atípicos (*outliers*), que están fuera del rango esperado.

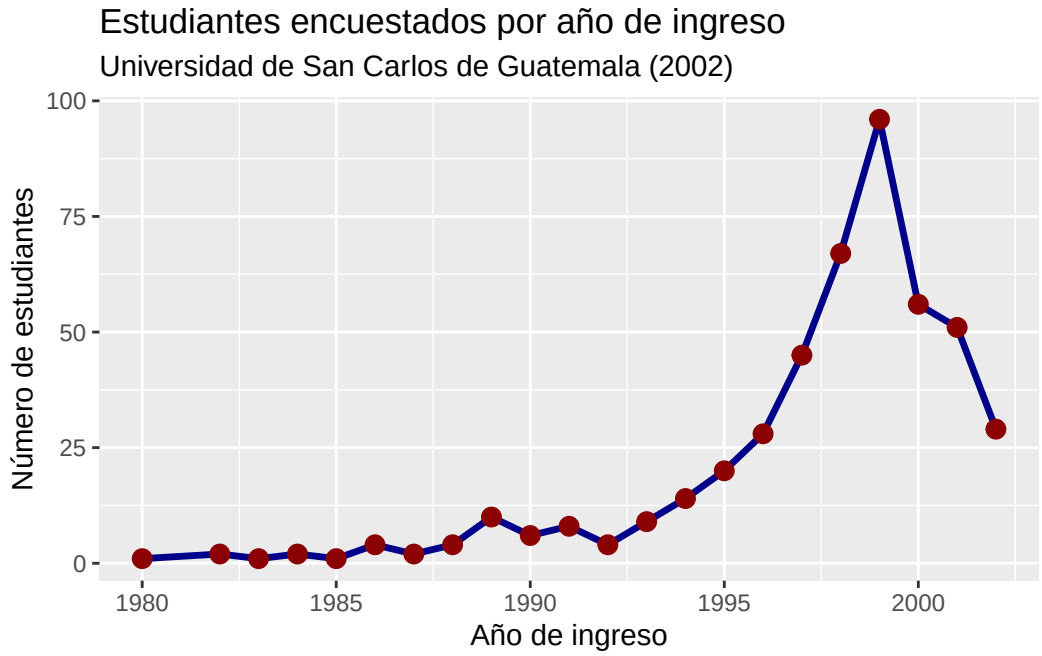
10.3.6 Gráfico de líneas

Los gráficos de líneas son ideales para visualizar tendencias a lo largo del tiempo o en secuencias de datos ordenados. Este tipo de gráfico es especialmente útil para identificar patrones, como aumentos, disminuciones o fluctuaciones en los datos.

10.3.6.1 Ejemplo práctico: Creación de un gráfico de líneas

El siguiente código muestra cómo crear un gráfico de líneas utilizando **ggplot2** para visualizar la cantidad de estudiantes encuestados por año de ingreso:

```
# Crear un gráfico de líneas de estudiantes por año de ingreso
ggplot(data = datos, aes(x = AÑO_ING)) +
  geom_line(stat = "count", color = "darkblue", linewidth = 1.2) +
  geom_point(stat = "count", color = "darkred", size = 3) +
  labs(title = "Estudiantes encuestados por año de ingreso",
       subtitle = "Universidad de San Carlos de Guatemala (2002)",
       x = "Año de ingreso",
       y = "Número de estudiantes")
```



Explicación del código

1. **ggplot(data = datos, aes(x = AÑO_ING))**: Se define el conjunto de datos (`datos`) y se especifica la variable `AÑO_ING` como el eje X, que representa los años de ingreso de los estudiantes. En este caso, no se especifica una variable para el eje Y, ya que el conteo de estudiantes por año se calcula automáticamente con `stat = "count"`.

2. **geom_line(stat = "count", color = "darkblue", linewidth = 1.2)**: La función `geom_line()` genera la línea que conecta los puntos correspondientes al conteo de estudiantes por año.

   `stat = "count"` indica que se debe contar automáticamente el número de observaciones en cada categoría del eje X.

   `color`: Define el color de la línea (en este caso, azul oscuro).

   `linewidth`: Ajusta el grosor de la línea (1.2 en este ejemplo).

3. **geom_point(stat = "count", color = "darkred", size = 3)**: La función `geom_point()` añade puntos en cada categoría del eje X, representando el conteo de estudiantes.

   `stat = "count"` asegura que los puntos correspondan al conteo calculado.

   `color`: Define el color de los puntos (en este caso, rojo oscuro).

   `size`: Ajusta el tamaño de los puntos (3 en este ejemplo).

10.4 Personalización de gráficos

La personalización de gráficos en **ggplot2** permite adaptarlos a diferentes necesidades, mejorando tanto su presentación como su capacidad para comunicar información de manera efectiva. A continuación, se describen algunas de las opciones más comunes, comenzando con la personalización de colores.

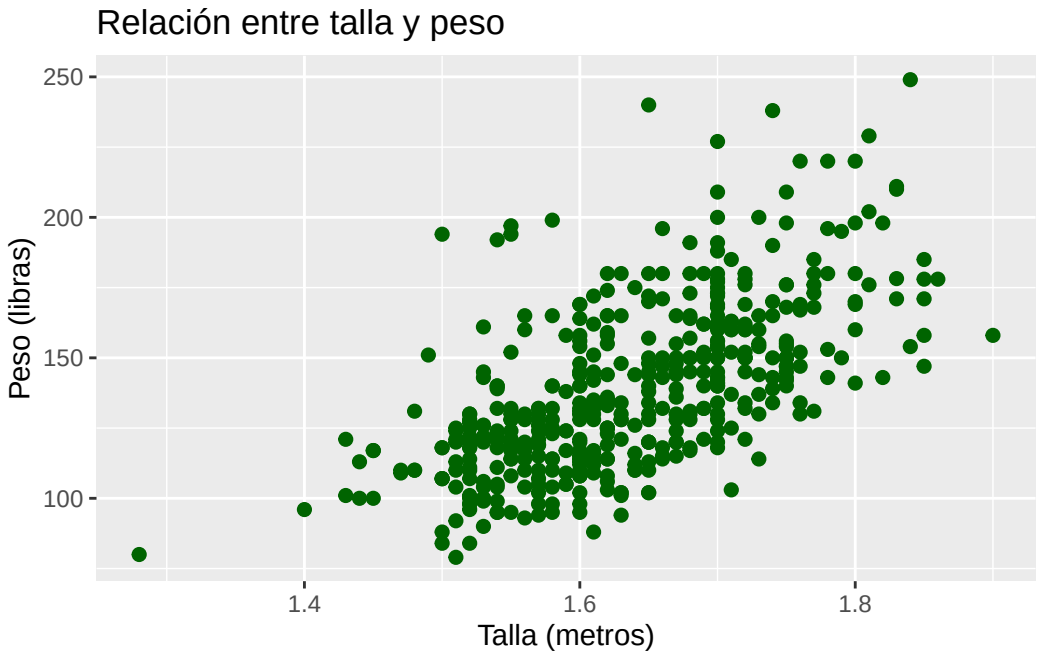
10.4.1 Personalización de colores

En **ggplot2**, es posible modificar los colores de los elementos del gráfico, como puntos, barras o líneas, para destacar información clave o mejorar la estética general. Esto se puede lograr utilizando argumentos como `color` (para bordes o contornos) y `fill` (para colores de relleno).

10.4.1.1 Personalización de colores en un gráfico de dispersión

El siguiente código muestra cómo personalizar el color de los puntos en un gráfico de dispersión:

```r
# Personalización de colores en un gráfico de dispersión
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs)) +
  geom_point(color = "darkgreen", size = 2) +
  labs(title = "Relación entre talla y peso",
       x = "Talla (metros)",
       y = "Peso (libras)")
```



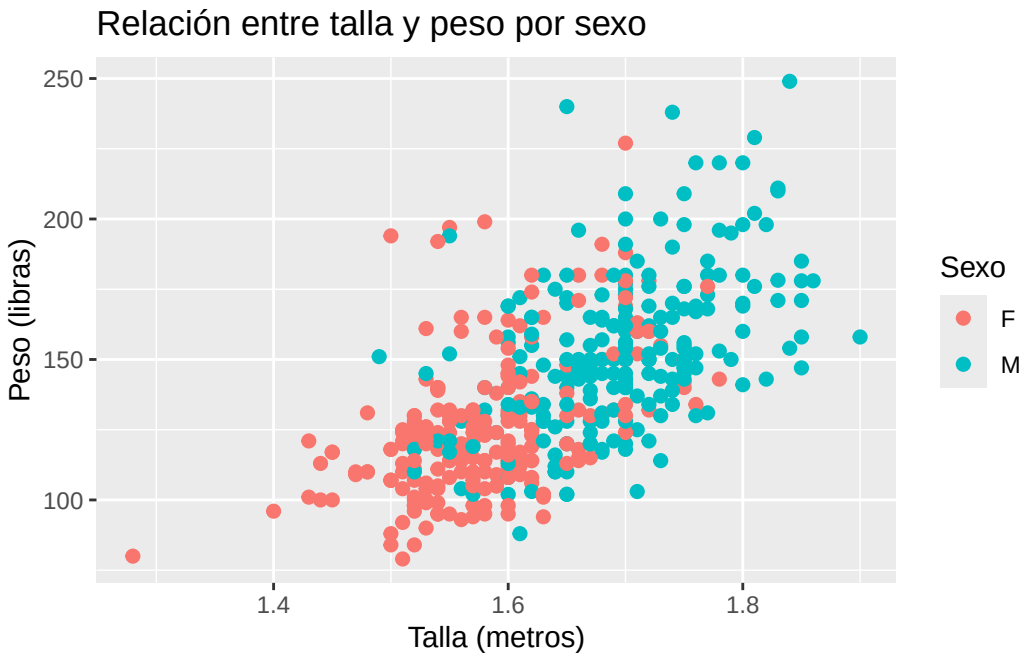
En este ejemplo:

1. `color = "darkgreen"` define el color de los puntos como verde oscuro.
2. `size = 2` ajusta el tamaño de los puntos para mejorar su visibilidad.

10.4.1.2 Personalización de colores por grupo

Si se desea asignar colores diferentes a los puntos según una variable categórica, se puede incluir el argumento `color` dentro de `aes()`:

```r
# Personalización de colores por grupo
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs, color = SEXO)) +
  geom_point(size = 2) +
  labs(title = "Relación entre talla y peso por sexo",
       x = "Talla (metros)",
       y = "Peso (libras)",
       color = "Sexo")
```



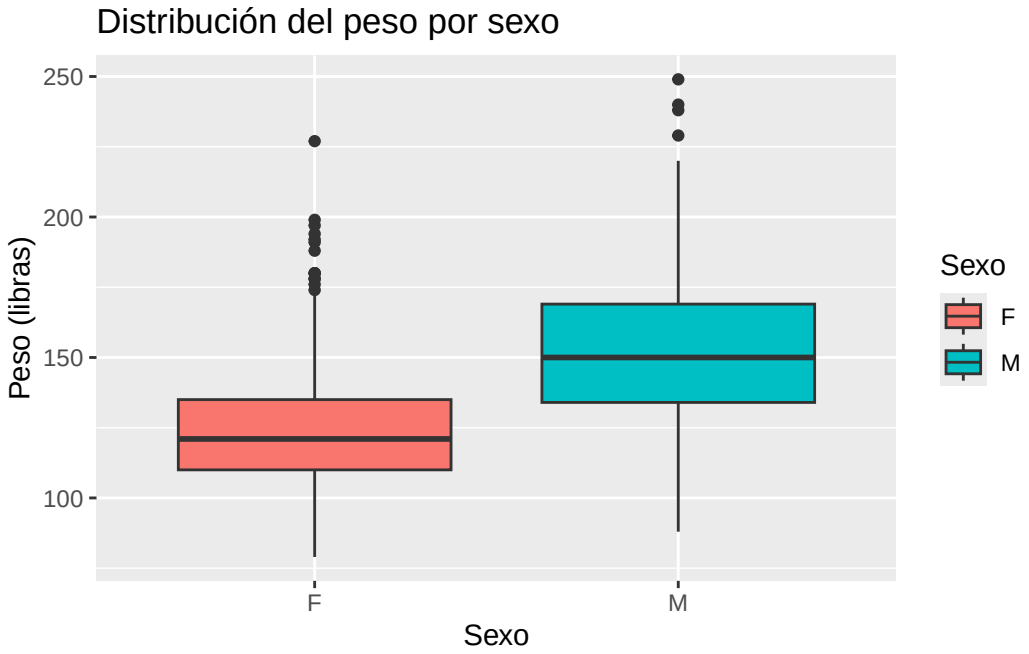
En este ejemplo:

1. Los puntos se colorean automáticamente según los valores de la variable `SEXO`.
2. La leyenda se genera de forma automática para indicar el significado de los colores.

10.4.1.3 Personalización de colores en gráficos con relleno

En gráficos como barras o *boxplots*, se utiliza el argumento `fill` para personalizar el color de relleno:

```
# Personalización de colores en gráficos con relleno
ggplot(data = datos, aes(x = SEXO, y = PESO_lbs, fill = SEXO)) +
  geom_boxplot() +
  labs(title = "Distribución del peso por sexo",
       x = "Sexo",
       y = "Peso (libras)",
       fill = "Sexo")
```

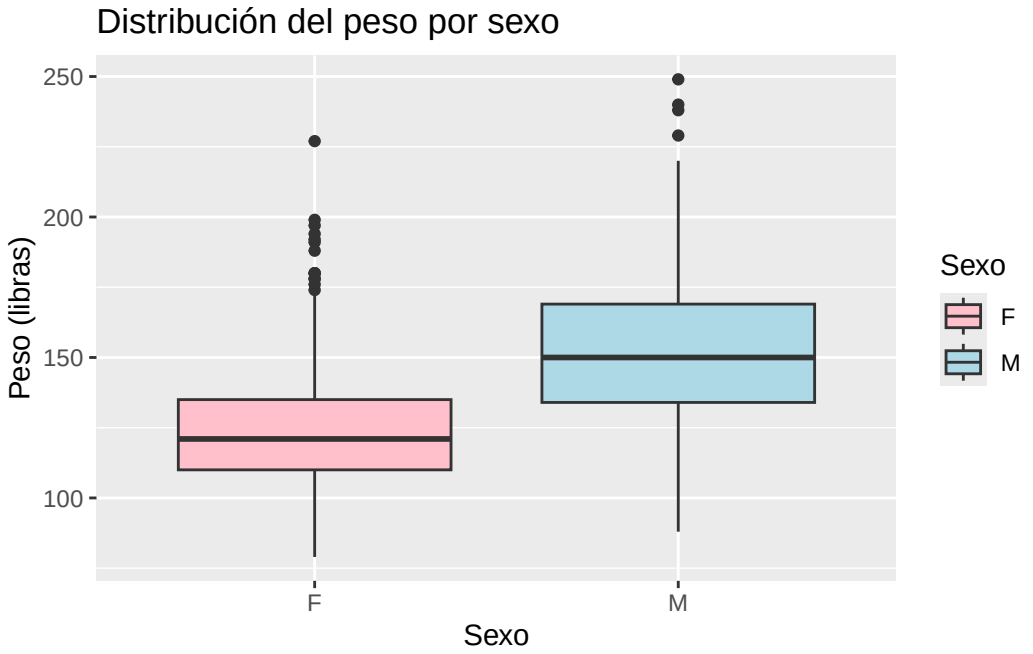


En este ejemplo, las cajas del *boxplot* se rellenan con colores diferentes según la variable `SEXO`.

10.4.1.4 Escalas de color personalizadas

Para un mayor control sobre los colores, se pueden definir escalas personalizadas utilizando funciones como `scale_color_manual()` o `scale_fill_manual()`:

```
# Escalas de color personalizadas
ggplot(data = datos, aes(x = SEXO, y = PESO_lbs, fill = SEXO)) +
  geom_boxplot() +
  scale_fill_manual(values = c(  "pink", "lightblue")) +
  labs(title = "Distribución del peso por sexo",
       x = "Sexo",
       y = "Peso (libras)",
       fill = "Sexo")
```



En este ejemplo, se asignan colores específicos a cada categoría de la variable `SEXO`, R realiza la realiza la asignación de los colores de la escala en orden alfabético de las variables categóricas.

10.4.2 Etiquetas y títulos

En **ggplot2**, es posible añadir y personalizar títulos, subtítulos y etiquetas de los ejes para mejorar la claridad y presentación de los gráficos. Estas etiquetas ayudan a contextualizar la información y a facilitar su interpretación.

10.4.2.1 Personalización de títulos y etiquetas

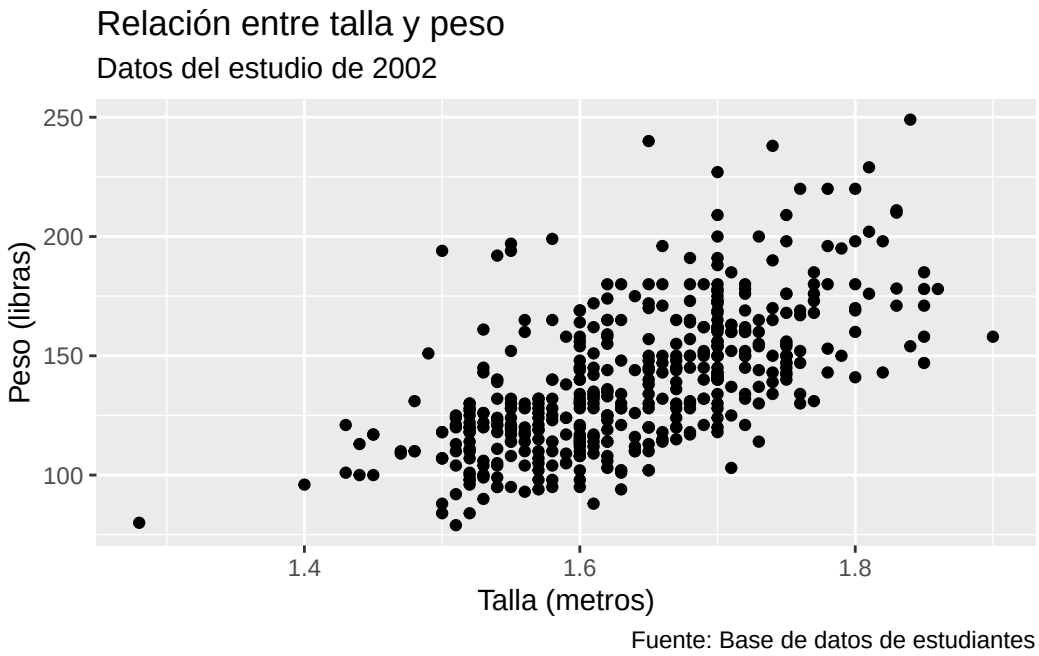
La función `labs()` se utiliza para añadir y personalizar los siguientes elementos:

1. **`title`**: Título principal del gráfico, que describe su propósito o contenido.
2. **`subtitle`**: Subtítulo que proporciona información adicional o contexto.
3. **`x`**: Etiqueta del eje X, que describe la variable representada en este eje.
4. **`y`**: Etiqueta del eje Y, que describe la variable representada en este eje.
5. **`caption`** (opcional): Texto adicional, como la fuente de los datos o notas aclaratorias.

10.4.2.2 Añadir subtítulos y etiquetas personalizadas

El siguiente código muestra cómo personalizar títulos, subtítulos y etiquetas de los ejes en un gráfico de dispersión:

```
# Añadir subtítulos y etiquetas personalizadas
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs)) +
  geom_point() +
  labs(title = "Relación entre talla y peso",
       subtitle = "Datos del estudio de 2002",
       x = "Talla (metros)",
       y = "Peso (libras)",
       caption = "Fuente: Base de datos de estudiantes")
```



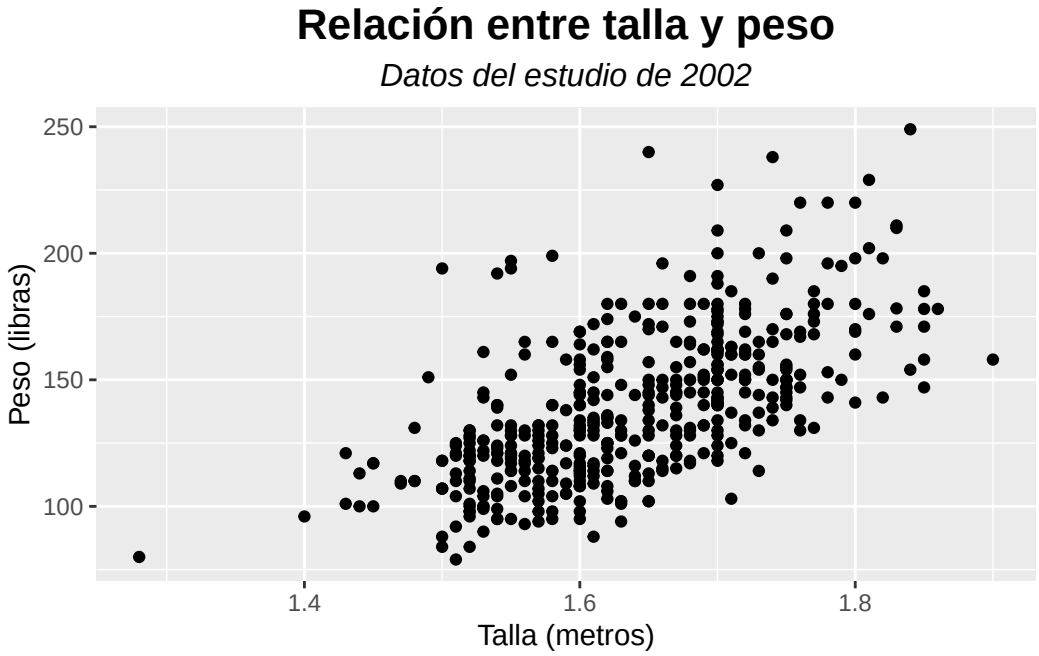
En este ejemplo:

1. `title` añade un título descriptivo al gráfico.
2. `subtitle` proporciona contexto adicional, como el año del estudio.
3. `x` y `y` personalizan las etiquetas de los ejes, indicando las unidades de medida.
4. `caption` incluye una nota al pie con la fuente de los datos.

10.4.2.3 Ajuste de la posición y estilo de los títulos

Se puede personalizar la posición, tamaño y estilo de los títulos utilizando la función `theme()`:

```r
# Ajuste de la posición y estilo de los títulos
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs)) +
  geom_point() +
  labs(title = "Relación entre talla y peso",
       subtitle = "Datos del estudio de 2002",
       x = "Talla (metros)",
       y = "Peso (libras)") +
  theme(plot.title = element_text(hjust = 0.5,
                                  size = 16,
                                  face = "bold"),
        plot.subtitle = element_text(hjust = 0.5,
                                     size = 12,
                                     face = "italic"))
```



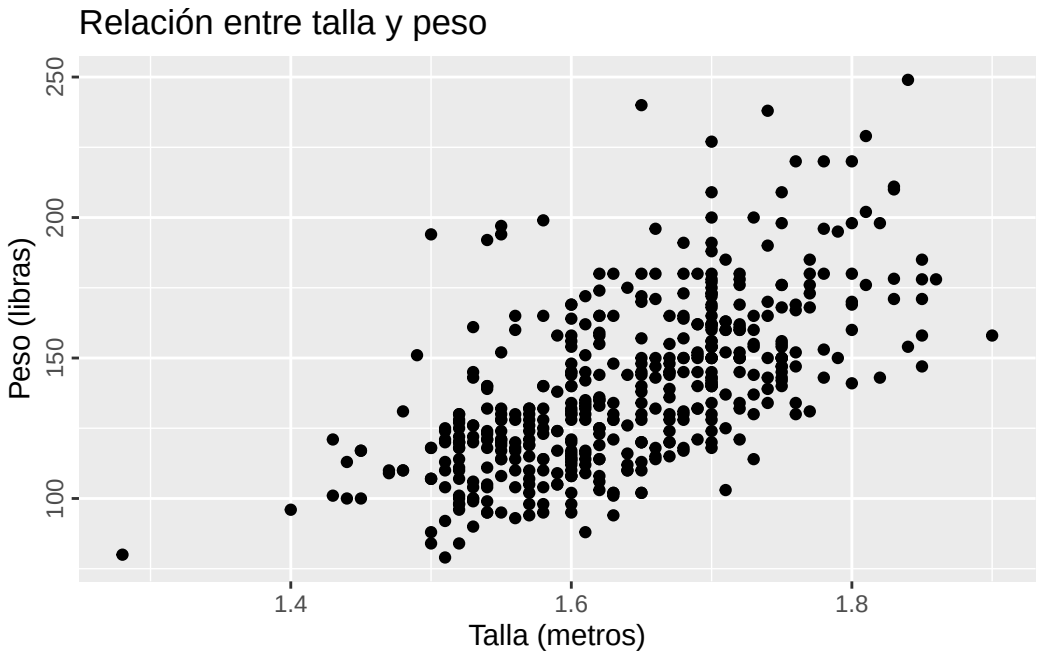
En este ejemplo:

1. `hjust = 0.5` centra el título y el subtítulo.
2. `size` ajusta el tamaño del texto.
3. `face` define el estilo del texto (`"bold"` para negrita, `"italic"` para cursiva).

10.4.2.4 Rotación de etiquetas en los ejes

Si las etiquetas del eje X son largas o numerosas, se pueden rotar para mejorar la legibilidad:

```r
# Rotación de etiquetas en los ejes
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs)) +
  geom_point() +
  labs(title = "Relación entre talla y peso",
       x = "Talla (metros)",
       y = "Peso (libras)") +
  theme(axis.text.y = element_text(angle = 90, hjust = 0.5))
```



En este ejemplo, las etiquetas del eje Y se rotan 90 grados.

10.4.3 Temas

En **ggplot2**, los temas permiten modificar el estilo general de un gráfico, ajustando elementos como el fondo, las líneas de los ejes, las fuentes y la disposición de los textos. Esto facilita la creación de gráficos con un diseño coherente y adaptado a diferentes propósitos, como presentaciones, informes o publicaciones.

10.4.3.1 Aplicación de temas predefinidos

ggplot2 incluye varios temas predefinidos que se pueden aplicar directamente para cambiar el estilo del gráfico. Algunos de los más comunes son:

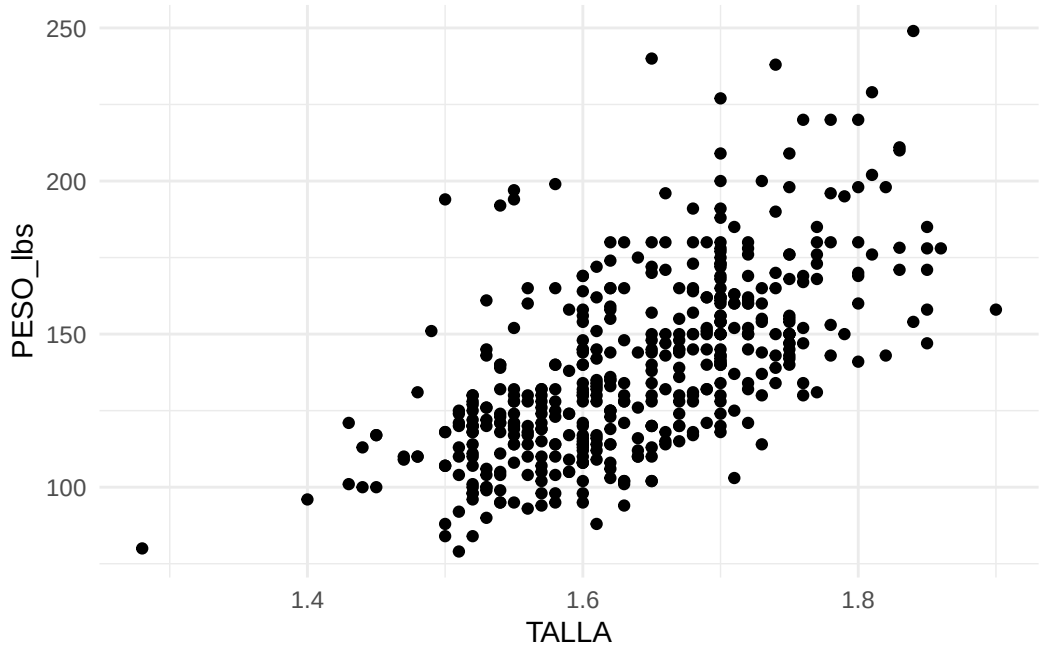
1. `theme_minimal()`: Un diseño limpio y moderno, con un fondo blanco y líneas simples.
2. `theme_classic()`: Un estilo clásico con líneas de ejes visibles y sin cuadrícula.
3. `theme_light()`: Similar a `theme_minimal()`, pero con cuadrículas más visibles.

4. **`theme_dark()`**: Un diseño con fondo oscuro, ideal para presentaciones.
5. **`theme_void()`**: Un gráfico sin ejes ni cuadrículas, útil para gráficos personalizados.

10.4.3.2 Ejemplo: Aplicar un tema minimalista

El siguiente código muestra cómo aplicar el tema `theme_minimal()` a un gráfico de dispersión:

```
# Ejemplo: Aplicar un tema minimalista
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs)) +
  geom_point() +
  theme_minimal()
```



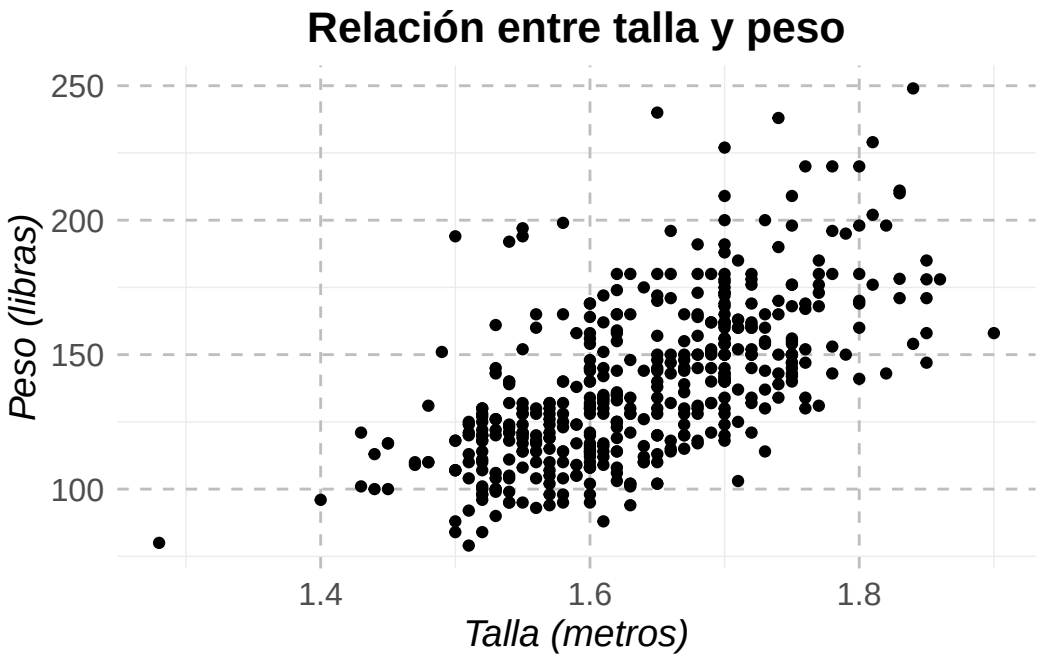
En este ejemplo: `theme_minimal()` elimina elementos innecesarios, como bordes y fondos grises, dejando un diseño limpio y profesional.

10.4.3.3 Ejemplo: personalización de temas

Además de los temas predefinidos, es posible personalizar elementos específicos del gráfico utilizando la función `theme()`. Por ejemplo:

```
# Personalización de temas
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs)) +
  geom_point() +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
```

```
    axis.text = element_text(size = 12),
    axis.title = element_text(size = 14, face = "italic"),
    panel.grid.major = element_line(color = "gray", linetype = "dashed")
  ) +
  labs(title = "Relación entre talla y peso",
       x = "Talla (metros)",
       y = "Peso (libras)")
```



En este ejemplo:

1. `plot.title` centra el título y ajusta su tamaño y estilo.
2. `axis.text` y `axis.title` modifican el tamaño y estilo de las etiquetas de los ejes.
3. `panel.grid.major` personaliza las líneas de la cuadrícula principal, cambiando su color y estilo.

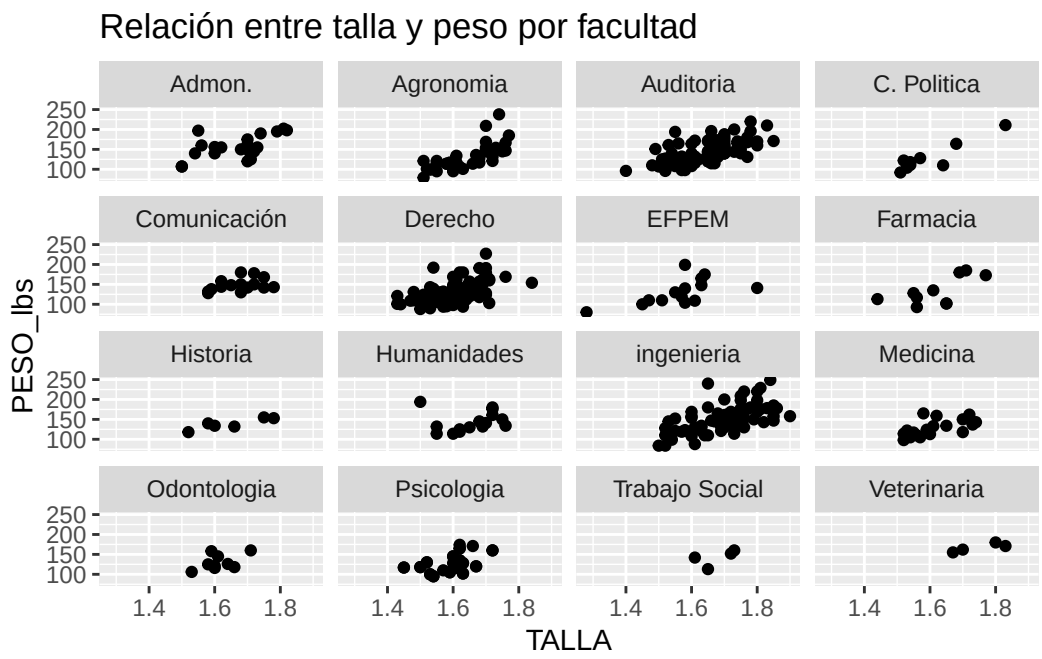
10.4.4 Facetas

Las facetas en **ggplot2** permiten dividir un gráfico en múltiples subgráficos basados en los valores de una variable categórica. Esto es especialmente útil para comparar patrones o relaciones entre diferentes grupos dentro de un conjunto de datos, manteniendo la coherencia visual.

10.4.4.1 Ejemplo: relación entre talla y peso por facultad

El siguiente código muestra cómo utilizar facetas para analizar la relación entre talla y peso, separando los datos por facultad:

```r
# Ejemplo: relación entre talla y peso por facultad
ggplot(data = datos, aes(x = TALLA, y = PESO_lbs)) +
  geom_point() +
  facet_wrap(~ FACULTAD) +
  labs(title = "Relación entre talla y peso por facultad")
```



En este ejemplo:

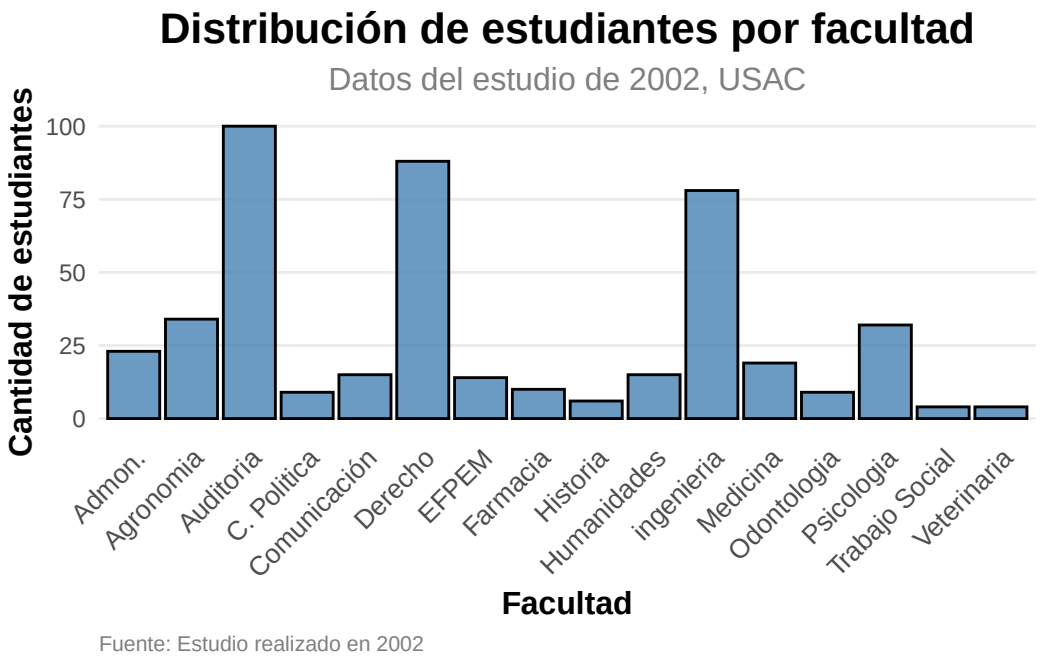
1. `facet_wrap(~ FACULTAD)` divide el gráfico en subgráficos, uno para cada valor único de la variable `FACULTAD`.
2. Cada subgráfico muestra la relación entre `TALLA` y `PESO_lbs` para una facultad específica.

10.4.5 Ejemplo avanzado de personalización

La flexibilidad y la lógica de capas de **ggplot2** permiten crear gráficos con un alto grado de personalización, adaptados a necesidades específicas y con un diseño profesional. A continuación, se presenta un ejemplo de un gráfico de barras con personalización detallada:

```r
# Ejemplo avanzado de personalización
ggplot(data = datos, aes(x = FACULTAD)) +
  geom_bar(fill = "steelblue", color = "black", alpha = 0.8) +
  labs(
    title = "Distribución de estudiantes por facultad",
    subtitle = "Datos del estudio de 2002, USAC",
    x = "Facultad",
    y = "Cantidad de estudiantes",
```

```
    caption = "Fuente: Estudio realizado en 2002"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    plot.subtitle = element_text(size = 12, hjust = 0.5, color = "gray50"),
    axis.title = element_text(size = 12, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 10),
    panel.grid.major.x = element_blank(),
    panel.grid.minor = element_blank(),
    plot.caption = element_text(hjust = 0, size = 8, color = "gray50")
  )
```



10.4.5.1 Explicación del código

1. **Datos y mapeo estético**: `aes(x = FACULTAD)`: Se utiliza la variable `FACULTAD` directamente desde la base de datos para el eje X.
2. **Gráfico de barras**: `geom_bar()`: Genera automáticamente las barras basándose en la frecuencia de cada categoría. En el ejemplo se utilizaron los siguientes argumentos:

   `fill = "steelblue"`: Define el color de relleno de las barras como azul acero.

   `color = "black"`: Establece el color de los bordes de las barras en negro.

   `alpha = 0.8`: Ajusta la transparencia de las barras, permitiendo un diseño más suave.
3. **Etiquetas y títulos**: `labs()`: Añade un título principal, subtítulo, etiquetas para los ejes y una nota al pie con la fuente de los datos.

4. **Tema profesional**: `theme_minimal()`: Aplica un diseño limpio y moderno.
5. **Personalización específica:** `theme()`: Personaliza elementos específicos del gráfico empleando los siguiente argumentos:

   `plot.title`: Ajusta el tamaño, estilo (negrita) y posición (centrado) del título.

   `plot.subtitle`: Cambia el tamaño, posición y color del subtítulo.

   `axis.title`: Modifica el tamaño y estilo de las etiquetas de los ejes.

   `axis.text.x`: Rota las etiquetas del eje X 45 grados para mejorar la legibilidad, especialmente si las categorías tienen nombres largos.

   `panel.grid.major.x` y `panel.grid.minor`: Elimina las líneas de cuadrícula verticales y menores para un diseño más limpio.

   `plot.caption`: Ajusta el tamaño, posición y color de la nota al pie.

Parte V

Exportación de resultados

11 Exportación de resultados

La exportación de resultados es una etapa crucial en el análisis de datos, ya que permite guardar gráficos y tablas para su uso en informes, presentaciones o análisis posteriores. Este capítulo detalla cómo guardar gráficos en formatos PNG y PDF utilizando la función `ggsave()` del paquete `ggplot2`, y cómo exportar tablas en formatos CSV y Excel utilizando las funciones `write.csv()` y `write_xlsx()` del paquete `writexl`.

11.1 Guardar gráficos con `ggsave()`

11.1.1 Sintaxis General de la Función `ggsave()`

La función `ggsave()` pertenece al paquete `ggplot2` y se utiliza para guardar gráficos en diferentes formatos de archivo, como PNG, PDF, JPEG, entre otros. Es una herramienta versátil que permite personalizar aspectos como el tamaño, la resolución y el formato del archivo de salida.

La sintaxis general de la función es la siguiente:

```
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 explican los argumentos principales de la función:

1. **filename:** Este argumento es obligatorio y define el nombre del archivo de salida, incluyendo su extensión (por ejemplo, `"grafico.png"` o `"grafico.pdf"`). La extensión del archivo determina automáticamente el formato en el que se guardará el gráfico, a menos que se especifique explícitamente con el argumento `device`. Es importante asegurarse de que el nombre del archivo sea válido para el sistema operativo utilizado.

2. **plot:** Este argumento opcional permite especificar el gráfico que se desea guardar. Si no se proporciona, `ggsave()` guardará automáticamente el último gráfico creado en la sesión de R, utilizando la función `last_plot()`. Esto es útil para flujos de trabajo interactivos,

pero en proyectos más complejos se recomienda asignar los gráficos a objetos para evitar confusiones.

3. **device:** El argumento `device` define el tipo de dispositivo o formato del archivo de salida, como `"png"`, `"pdf"`, `"jpeg"`, entre otros. Si no se especifica, el formato se deduce automáticamente a partir de la extensión del archivo en `filename`. Este argumento es útil cuando se desea guardar un archivo con un formato específico, independientemente de la extensión.

4. **path:** Este argumento opcional permite especificar el directorio donde se guardará el archivo. Si no se proporciona, el archivo se guardará en el directorio de trabajo actual. Es especialmente útil para organizar los gráficos en carpetas específicas dentro de un proyecto.

5. **scale:** El argumento `scale` ajusta el tamaño del gráfico multiplicando las dimensiones especificadas en `width` y `height` por el valor proporcionado. Por defecto, su valor es `1`, lo que significa que no se aplica escalado. Un valor mayor que `1` aumenta el tamaño del gráfico, mientras que un valor menor lo reduce.

6. **width y height:** Estos argumentos definen el ancho y la altura del gráfico en las unidades especificadas por el argumento `units`. Si no se proporcionan, se utilizan las dimensiones predeterminadas del gráfico. Es importante ajustar estas dimensiones para garantizar que el gráfico se adapte correctamente al formato de salida.

7. **units:** El argumento `units` especifica las unidades de medida para `width` y `height`. Los valores posibles son `"in"` (pulgadas), `"cm"` (centímetros) y `"mm"` (milímetros). Por defecto, se utilizan pulgadas (`"in"`), pero se pueden cambiar según las necesidades del proyecto.

8. **dpi:** El argumento `dpi` (dots per inch) define la resolución del gráfico, siendo relevante para formatos rasterizados como PNG, JPEG o TIFF. Su valor predeterminado es `300`, adecuado para impresión. Para gráficos destinados a la web, se puede utilizar un valor menor, como `72`.

9. **limitsize:** Este argumento controla si se permite guardar gráficos con dimensiones excesivamente grandes (mayores a 50 pulgadas). Si se establece en `TRUE` (valor predeterminado), se genera un error al intentar guardar gráficos grandes. Para desactivar esta restricción, se debe establecer en `FALSE`.

11.1.2 Guardar en formatos PDF y PNG

Si no se especifica el argumento `plot`, la función `ggsave()` guardará automáticamente el último gráfico creado en la sesión de R. Esto es útil cuando se trabaja de manera interactiva y se desea guardar rápidamente un gráfico sin asignarlo a un objeto. A continuación, se describen los pasos para exportar gráficos en formatos PNG y PDF, junto con ejemplos prácticos.

Crear un gráfico con `ggplot2`

Antes de guardar un gráfico, es necesario crearlo. A continuación, se presenta el ultimo gráfico realizado en el capitulo anterior:

```r
# Instalación y carga de paquetes esenciales

# Paquete que incluye ggplot2, dplyr, tidyr
if (!require("tidyverse")) install.packages("tidyverse")

# Importar la base de datos
datos <- read_csv("datos_estudiantes.csv")

# Ejemplo avanzado de personalización de un gráfico
plot<- ggplot(data = datos, aes(x = FACULTAD)) +
  geom_bar(fill = "steelblue", color = "black", alpha = 0.8) +
  labs(
    title = "Distribución de estudiantes por facultad",
    subtitle = "Datos del estudio de 2002,
    Universidad de San Carlos de Guatemala",
    x = "Facultad",
    y = "Cantidad de estudiantes",
    caption = "Fuente: Estudio realizado en 2002"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 16, face = "bold", hjust = 0.5),
    plot.subtitle = element_text(size = 12, hjust = 0.5, color = "gray50"),
    axis.title = element_text(size = 12, face = "bold"),
    axis.text.x = element_text(angle = 45, hjust = 1, size = 10),
    panel.grid.major.x = element_blank(),
    panel.grid.minor = element_blank(),
    plot.caption = element_text(hjust = 0, size = 8, color = "gray50")
  )
```

Guardar el gráfico en formato PNG: Una vez creado el gráfico, se puede guardar utilizando `ggsave()`:

```r
# Guardar el gráfico en formato PNG
ggsave("grafico.png", width = 8, height = 6, dpi = 300)
```

Parámetros importantes:

`filename`: Nombre del archivo de salida (en este caso, `"grafico.png"`).

`plot`: Objeto del gráfico que se desea guardar.

`width` y `height`: Dimensiones del gráfico en pulgadas.

`dpi`: Resolución del archivo en puntos por pulgada (300 dpi es adecuado para impresión).

Resultado: El archivo `grafico.png` se guardará en el directorio de trabajo actual con las dimensiones y resolución especificadas.

Guardar en Formato PDF: El formato PDF es ideal para gráficos que requieren escalado sin pérdida de calidad, como en publicaciones científicas o informes.

```r
# Guardar el gráfico en formato PDF
ggsave("grafico.pdf", width = 8, height = 6)
```

Diferencias con PNG: No es necesario especificar la resolución (`dpi`), ya que el formato PDF es vectorial y no depende de la resolución.

Resultado: El archivo `grafico.pdf` se guardará en el directorio de trabajo actual, listo para ser escalado o utilizado en documentos de alta calidad.

11.2 Guardar Tablas en CSV y Excel

La exportación de tablas es esencial para compartir datos o realizar análisis adicionales en otras herramientas. A continuación, se detalla cómo guardar tablas en formatos CSV y Excel.

11.2.1 Crear un data frame de ejemplo

Se puede utilizar un data frame de ejemplo para ilustrar el proceso:

```r
# Crear un data frame de ejemplo
ejemplo <- data.frame(
  Nombre = c("Ana", "Luis", "María"),
  Edad = c(25, 30, 22),
  Ciudad = c("Madrid", "Barcelona", "Valencia")
)
```

11.2.2 Guardar la tabla en formato CSV

Utilizando la función `write.csv()`, se puede exportar el data frame:

```r
# Guardar la tabla en formato CSV
write.csv(ejemplo, "ejemplo.csv", row.names = FALSE)
```

Parámetros importantes:

`file`: Nombre del archivo de salida (en este caso, `"tabla.csv"`).

`row.names`: Si se establece en `FALSE`, no se incluirán los índices de las filas como una columna adicional.

Resultado: El archivo `ejemplo.csv` se guardará en el directorio de trabajo actual, listo para ser abierto en cualquier editor de texto o software como Excel.

11.2.3 Guardar en Formato Excel

El formato Excel es útil para compartir datos en un archivo más estructurado y compatible con herramientas como Microsoft Excel.

11.2.3.1 Instalar y cargar el paquete `writexl`

```
# Paquete para la exportación de datos a Excel
if (!require("writexl")) install.packages("writexl")
```

11.2.3.2 Guardar la tabla en formato Excel

Utilizando la función `write_xlsx()`, se puede exportar el data frame:

```
# Guardar la tabla en formato Excel
write_xlsx(ejemplo, "ejemplo.xlsx")
```

Parámetros importantes: `path`: Nombre del archivo de salida (en este caso, `"ejemplo.xlsx"`).

Resultado: El archivo `ejemplo.xlsx` se guardará en el directorio de trabajo actual, listo para ser abierto en Microsoft Excel o software similar.

11.3 Comparación de Formatos

Formato	Uso Principal	Ventajas	Desventajas
PNG	Presentaciones y documentos digitales	Alta calidad, ampliamente compatible	No escalable sin pérdida de calidad
PDF	Publicaciones científicas e informes	Escalable, ideal para impresión	Menos compatible con editores básicos
CSV	Análisis de datos en herramientas simples	Ligero, compatible con múltiples plataformas	No admite formatos complejos
Excel	Compartir datos estructurados	Compatible con herramientas avanzadas	Requiere software específico

Parte VI

Material de apoyo y referencias

12 Material de apoyo y referencias

12.1 Material de apoyo

1. Tutorial en YouTube "Cómo instalar R y RStudio en menos de 2 minutos - 2024". Elaborado por Herbert Lizama.
2. R para ciencia de datos por Handley Wickham & Garrett Grolemund

12.2 Referencias

Allaire, J. J., Xie, Y., & McPherson, J. (2022). *R Markdown: The Definitive Guide*. Chapman & Hall/CRC. https://www.taylorfrancis.com/books/mono/10.1201/9781138359444/markdown-yihui-xie-allaire-garrett-grolemund

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature, 533*(7604), 452–454. https://doi.org/10.1038/533452a

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

Hmelo-Silver, C. E., Duncan, R. G., & Chinn, C. A. (2007). Scaffolding and achievement in problem-based and inquiry learning: A response to Kirschner, Sweller, and Clark (2006). *Educational Psychologist, 42*(2), 99–107. https://doi.org/10.1080/00461520701263368

Ihaka, R., & Gentleman, R. (1996). R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics, 5*(3), 299–314.

Kolb, D. A. (1984). *Experiential learning: Experience as the source of learning and development*. Prentice Hall.

López E. & González, B. (2016). *Diseño y análisis de experimentos*. Internet Archive. https://archive.org/details/DiseoYAnlisisDeExperimentos2016

National Academies of Sciences, Engineering, and Medicine. (2019). *Reproducibility and replicability in science*. National Academies Press. https://doi.org/10.17226/25303

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. https://www.Rproject.org

The Turing Way Community. (2023). *The Turing Way: A handbook for reproducible, ethical and collaborative research*. https://the-turing-way.netlify.app

Wilkinson, M. D. et al. (2016). The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data, 3*(1), 160018. https://doi.org/10.1038/sdata.2016.18

Parte VII

Ejemplos de Análisis Estadístico con R

13 Estadística descriptiva usando funciones en R

La estadística descriptiva es una rama esencial de la estadística que se ocupa de resumir y describir las características principales de un conjunto de datos. Su propósito es proporcionar una visión clara y comprensible de los datos, permitiendo identificar patrones, tendencias y comportamientos generales sin realizar inferencias o predicciones. Este tipo de análisis es el primer paso en cualquier estudio estadístico, ya que organiza y presenta la información de manera que sea fácil de interpretar.

En R, la estadística descriptiva se puede realizar de manera eficiente gracias a una amplia variedad de herramientas y funciones predefinidas, así como paquetes especializados que amplían las capacidades del análisis. Estas herramientas permiten calcular medidas clave que se agrupan en tres categorías principales: medidas de tendencia central, medidas de dispersión y medidas de forma.

13.1 Medidas principales en estadística descriptiva

13.1.1 Medidas de tendencia central

Las medidas de tendencia central describen el valor típico o central de un conjunto de datos. Estas medidas son fundamentales para resumir los datos en un solo valor representativo.

Media aritmética: Es el promedio aritmético de los datos. Se calcula sumando todos los valores y dividiendo entre el número total de observaciones. Es sensible a valores extremos (outliers).

Fórmula:

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Ejemplo en R:

```r
datos <- c(10, 20, 30, 40, 50)
media <- mean(datos)
print(media)  # Resultado:
```

```
[1] 30
```

Mediana: Es el valor que divide el conjunto de datos en dos partes iguales, de modo que el 50% de los valores son menores o iguales a la mediana y el otro 50% son mayores o iguales. Es menos sensible a valores extremos que la media.

Ejemplo en R:

```r
mediana <- median(datos)
print(mediana)  # Resultado:
```

```
[1] 30
```

Moda: Es el valor o los valores que ocurren con mayor frecuencia en un conjunto de datos. En R base, no existe una función predefinida para calcular la moda, pero se puede implementar fácilmente.

Ejemplo de función para calcular la moda:

```r
moda_ej <- function(x) {
  tabla <- table(x)
  moda <- names(tabla[tabla == max(tabla)])
  return(moda)
}
datos_moda <- c(10, 20, 20, 30, 40)
print(moda_ej(datos_moda))  # Resultado:
```

```
[1] "20"
```

13.1.2 Medidas de dispersión

Las medidas de dispersión describen la variabilidad o el grado de dispersión de los datos en torno a la media. Estas medidas son esenciales para entender la distribución de los datos.

Varianza: Mide la dispersión de los datos respecto a la media. Es el promedio de las diferencias al cuadrado entre cada valor y la media. Una varianza alta indica que los datos están muy dispersos.

Fórmula de la varianza muestral:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \overline{x}\right)^2$$

```
varianza <- var(datos)
print(varianza)  # Resultado:
```

```
[1] 250
```

Desviación estándar: Es la raíz cuadrada de la varianza. Proporciona una medida de dispersión en las mismas unidades que los datos originales.

Fórmula:

$$\text{Desviación estándar} = \sqrt{\text{Varianza}}$$

Ejemplo en R:

```
desviacion <- sd(datos)
print(desviacion)  # Resultado:
```

```
[1] 15.81139
```

Rango: Es la diferencia entre el valor máximo y el valor mínimo de los datos. Es una medida simple pero útil para entender la amplitud de los datos.

Ejemplo en R:

```
rango <- max(datos)-min(datos)
print(rango)  # Resultado:
```

```
[1] 40
```

Rango intercuartílico (IQR): Es la diferencia entre el tercer cuartil (Q3) y el primer cuartil (Q1). Representa la dispersión de la mitad central de los datos y es menos sensible a valores extremos.

Fórmula:

$$\text{IQR} = Q3 - Q1$$

Ejemplo en R:

```
iqr <- IQR(datos)
print(iqr)  # Resultado:
```

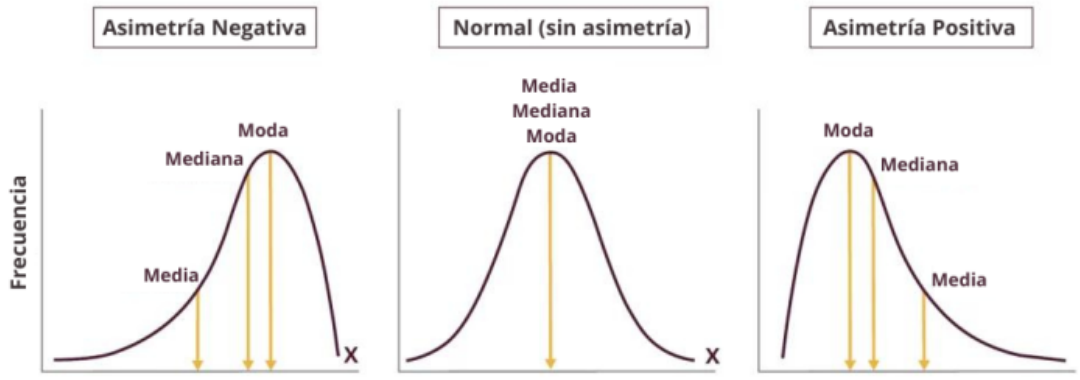
```
[1] 20
```

13.1.3 Medidas de forma

Las medidas de forma describen la distribución de los datos en términos de su simetría y concentración en torno a la media.

Asimetría (skewness): Mide el grado de simetría de la distribución de los datos. Una asimetría positiva indica que la cola derecha es más larga, mientras que una asimetría negativa indica que la cola izquierda es más larga.

Guía gráfica para interpretar asimetría:



Fórmula:

$$\text{Asimetría} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{n \cdot s^3}$$

Ejemplo en R (usando el paquete `psych`):

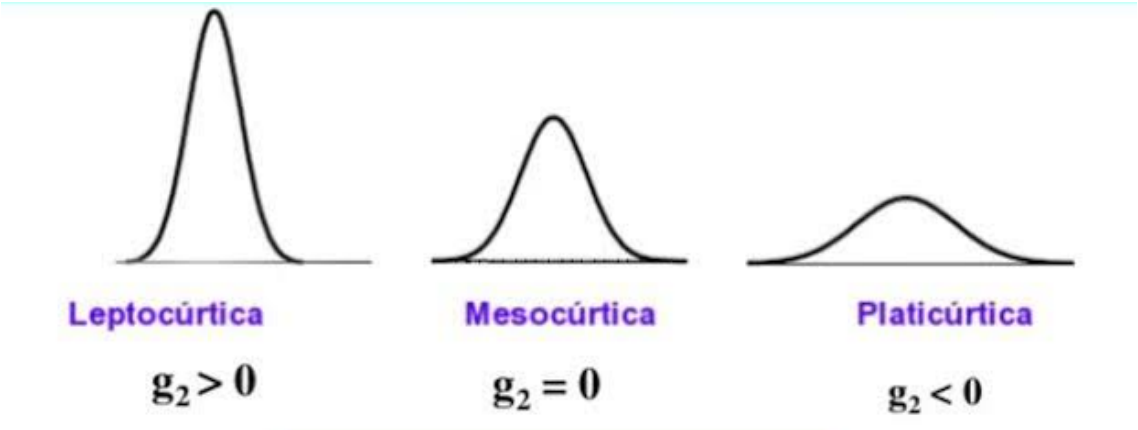
```r
# Instalación y carga del paquete psych
if (!require("psych")) install.packages("psych")

# Análisis de asimetria
datos <- c(10, 20, 30, 40, 50)
asimetria <- skew(datos)
print(asimetria)  # Resultado: 0 (distribución simétrica)
```

```
[1] 0
```

Curtosis (kurtosis): Mide la concentración de los datos en torno a la media. Una curtosis alta indica una distribución con colas más pesadas (leptocúrtica), mientras que una curtosis baja indica colas más ligeras (platicúrtica).

Guía gráfica para interpretar curtosis:



Fórmula:

$$\text{Curtosis} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^4}{n \cdot s^4} - 3$$

Ejemplo en R (usando el paquete `psych`):

```r
curtosis <- kurtosi(datos)
print(curtosis)  # Resultado: -1.912 (distribución platicúrtica)
```

```
[1] -1.912
```

13.2 Base de datos para los ejemplos

En 2002, se llevó a cabo un estudio en la Universidad de San Carlos de Guatemala, en el que se recopilaron datos de 460 estudiantes de diversas facultades, generando una base de datos que incluye una amplia variedad de variables como: *FACULTAD, EDAD, SEXO, EST_CIVIL, PESO_lbs, TALLA, entre otras*. Esta base de datos, disponible para su descarga en formato CSV, se utilizará a lo largo de esta sección del manual para ilustrar diferentes métodos de análisis descriptivo de datos, adaptando las herramientas y conceptos desarrollados a las características de las variables incluidas. Para seguir los ejemplos prácticos, se recomienda que el usuario descargue el archivo y lo guarde en la carpeta correspondiente al proyecto en curso.

13.2.1 Preparación del área de trabajo

Antes de comenzar con el análisis, es necesario preparar el entorno de trabajo instalando y cargando los paquetes necesarios, estableciendo el directorio de trabajo y revisando la estructura de los datos.

```
# Instalación y carga de paquetes
 # Incluye ggplot2, dplyr, tidyr
if (!require("tidyverse")) install.packages("tidyverse")

 # Exportación a Excel
if (!require("writexl")) install.packages("writexl")

 # Realiza analisis de estaística descriptiva completos
if (!require("psych")) install.packages("psych")

 # Se utiliza para establecer el directorio de trabajo
if (!require("rstudioapi")) install.packages("rstudioapi")
```

13.2.2 Establecer directorio de trabajo

Es importante asegurarse de que el archivo de datos esté en el directorio de trabajo correcto. Esto se puede hacer con el siguiente código una vez ya se ha guardado el script:

```
# Establecer y verificar directorio de trabajo
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
getwd()
```

13.2.3 Importar la base de datos

Una vez establecido el directorio de trabajo, se puede importar la base de datos en formato CSV:

```
# Importar la base de datos
estudiantes<- read_csv("datos_estudiantes.csv")
```

13.2.4 Revisar de la estructura de los datos

Es fundamental revisar la estructura de los datos para entender el tipo de variables y su formato:

```
# Revisar la estructura de los datos
sapply(estudiantes, class)

# Convertir todos los nombres de las columnas a minúsculas
names(estudiantes)<- tolower(names(estudiantes))

# Tener todas las variables en minúsculas facilita su manipulación

# Revisar los valores de las variables categoricas
categoricas<-list(facultad = c(unique(estudiantes$facultad)),
           sexo = c(unique(estudiantes$sexo)),
           est_civil = c(unique(estudiantes$est_civil)),
           trabaja = c(unique(estudiantes$trabaja)),
           jornada = c(unique(estudiantes$jornada)),
           fuma = c(unique(estudiantes$fuma)),
           alcohol = c(unique(estudiantes$alcohol)))
```

13.2.5 Limpieza de la base de datos

Antes de realizar el análisis, es necesario limpiar los datos para corregir valores inconsistentes y asegurarse de que las variables estén en el formato adecuado:

```
# Corregir los valores incorrectos de la variable "fuma"
estudiantes$fuma <- ifelse(tolower(estudiantes$fuma) == "sí",
                           1, estudiantes$fuma)
unique(estudiantes$fuma)
```

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

```
# Establecer como varibales tipo factor a las variables categoricas
estudiantes <- estudiantes %>%
  mutate(across(
    c(facultad, sexo, est_civil, trabaja, jornada, fuma, alcohol),
    as.factor))
```

13.3 Funciones por defecto en R para estadística descriptiva

R base proporciona varias funciones útiles para realizar análisis descriptivos básicos. A continuación, se presentan ejemplos utilizando la base de datos de estudiantes.

13.3.1 Medidas de tendencia central

Las medidas de tendencia central describen el valor típico o central de un conjunto de datos.

Media: Promedio de los valores.

Mediana: Valor que divide los datos en dos partes iguales.

Moda: Valor más frecuente (no existe una función por defecto en R para calcular la moda).

```
# Calcular medidas de tendencia central para la variable "edad"
media_edad <- mean(estudiantes$edad, na.rm = TRUE)
mediana_edad <- median(estudiantes$edad, na.rm = TRUE)

# Resultados
print(paste("Media:", media_edad))
```

```
[1] "Media: 24.0195652173913"
```

```
print(paste("Mediana:", mediana_edad))
```

```
[1] "Mediana: 22"
```

13.3.2 Medidas de dispersión

Las medidas de dispersión describen la variabilidad de los datos.

Varianza: Dispersión respecto a la media.

Desviación estándar: Raíz cuadrada de la varianza.

Rango: Diferencia entre el valor máximo y mínimo.

Rango intercuartílico (IQR): Diferencia entre el tercer y primer cuartil.

```
# Calcular medidas de dispersión para la variable "peso_lbs"
varianza_peso <- var(estudiantes$peso_lbs, na.rm = TRUE)
desviacion_peso <- sd(estudiantes$peso_lbs, na.rm = TRUE)
rango_peso <- max(estudiantes$peso_lbs)-min(estudiantes$peso_lbs)
iqr_peso <- IQR(estudiantes$peso_lbs, na.rm = TRUE)

# Resultados
print(paste("Varianza:", varianza_peso))
```

```
[1] "Varianza: 872.415330870512"
```

```
print(paste("Desviación estándar:", desviacion_peso))
```

```
[1] "Desviación estándar: 29.5366777222915"
```

```
print(paste("Rango:", paste(rango_peso)))
```

```
[1] "Rango: 170"
```

```
print(paste("IQR:", iqr_peso))
```

```
[1] "IQR: 40"
```

13.3.3 Medidas de forma

Las medidas de forma describen la distribución de los datos en términos de simetría y concentración.

Asimetría: Grado de simetría de la distribución.

Curtosis: Concentración de los datos en torno a la media.

La asimetría y curtosis no se pueden calcular con la funciones base de R para ello se debe emplear el paquete `psych`, con este las medidas de forma se calculan fácilmente:

```
# Calcular asimetría y curtosis para la variable "talla"
asimetria_talla <- skew(estudiantes$talla, na.rm = TRUE)
curtosis_talla <- kurtosi(estudiantes$talla, na.rm = TRUE)

# Resultados
print(paste("Asimetría:", asimetria_talla))
```

```
[1] "Asimetría: 0.0362232638780007"
```

```
print(paste("Curtosis:", curtosis_talla))
```

```
[1] "Curtosis: -0.177190677008652"
```

13.3.4 Resumen general con `summary()`

La función `summary()` proporciona un resumen estadístico básico para variables numéricas y categóricas:

```r
# Resumen general de la variable talla
resumen_general <- summary(estudiantes$talla)
print(resumen_general)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.280   1.570   1.630   1.639   1.700   1.900
```

13.4 Paquetes especializados para estadística descriptiva (el paquete `psych`)

El paquete `psych` es una herramienta poderosa y versátil para realizar análisis estadísticos descriptivos avanzados en R. Este paquete es especialmente útil cuando se trabaja con variables categóricas y numéricas simultáneamente, ya que permite calcular estadísticas detalladas, realizar análisis por grupos y obtener medidas de forma como asimetría y curtosis. Además, incluye opciones para calcular errores estándar e intervalos de confianza, lo que lo convierte en una excelente opción para análisis más completos.

13.4.1 Instalación y carga del paquete

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

13.4.2 Análisis descriptivo general

A continuación, se utilizará la base de datos de estudiantes para realizar un análisis descriptivo detallado. Este análisis incluirá medidas de tendencia central, dispersión y forma.

La función `describe()` del paquete `psych` permite calcular estadísticas descriptivas detalladas para variables numéricas. Estas estadísticas incluyen: Media, Desviación estándar, Mediana, Rango, Asimetría, Curtosis, Errores estándar.

```r
# Análisis descriptivo general para variables numéricas
resultado_general <- describe(estudiantes[, c("edad", "peso_lbs", "talla")])

# Mostrar resultados
print(resultado_general)
```

```
         vars   n   mean    sd median trimmed   mad   min   max  range skew
edad        1 460  24.02  5.74  22.00   22.99  2.97 17.00  55.0  38.00 1.94
peso_lbs    2 460 139.44 29.54 134.00  137.46 29.65 79.00 249.0 170.00 0.68
talla       3 460   1.64  0.09   1.63    1.64  0.10  1.28   1.9   0.62 0.04
         kurtosis   se
edad         4.39 0.27
```

```
peso_lbs     0.42 1.38
talla       -0.18 0.00
```

Salida esperada: El resultado incluye un resumen detallado de cada variable numérica, con estadísticas como la media, desviación estándar, asimetría y curtosis.

13.4.3 Análisis descriptivo categorizado

La función `describeBy()` permite realizar un análisis descriptivo agrupado por una o más variables categóricas. Esto es útil para comparar estadísticas entre diferentes grupos.

```r
# Análisis descriptivo agrupado por sexo y trabaja
resultado_agrupado <- describeBy(
  estudiantes[, c("edad", "peso_lbs", "talla")],
  group = list(estudiantes$sexo, estudiantes$trabaja)
)

# Mostrar resultados
print(resultado_agrupado)
```

```
 Descriptive statistics by group
: F
: 1
         vars  n   mean    sd median trimmed   mad   min    max  range  skew
edad        1 76  26.04  5.69  25.00   25.37  5.93 18.00  42.00  24.00  0.91
peso_lbs    2 76 126.67 25.32 122.50  124.05 20.76 79.00 199.00 120.00  0.97
talla       3 76   1.57  0.08   1.57    1.57  0.07  1.28   1.75   0.47 -0.42
         kurtosis   se
edad         0.14 0.65
peso_lbs     0.86 2.90
talla        1.73 0.01
------------------------------------------------------------
: M
: 1
         vars   n   mean    sd median trimmed   mad    min    max  range  skew
edad        1 105  27.58  7.49   25.0   26.59  5.93  18.00  55.00  37.00  1.26
peso_lbs    2 105 156.31 26.94  152.0  154.75 25.20 102.00 238.00 136.00  0.61
talla       3 105   1.70  0.07    1.7    1.70  0.07   1.49   1.85   0.36 -0.09
         kurtosis   se
edad         1.34 0.73
peso_lbs     0.41 2.63
talla       -0.15 0.01
------------------------------------------------------------
: F
: 2
         vars   n   mean    sd median trimmed   mad  min    max  range skew
```

```
edad        1 154  22.57  4.51  21.00   21.77  2.97 17.0  44.00  27.00 2.53
peso_lbs    2 154 125.66 24.56 120.00  123.27 17.79 84.0 227.00 143.00 1.07
talla       3 154   1.59  0.07   1.58    1.58  0.06  1.4   1.78   0.38 0.44
         kurtosis   se
edad         7.79 0.36
peso_lbs     1.37 1.98
talla       -0.07 0.01
------------------------------------------------------------
: M
: 2
         vars   n   mean    sd median trimmed   mad   min   max  range skew
edad        1 125  21.58  2.87   21.0   21.24  1.48 17.00  34.0  17.00 1.66
peso_lbs    2 125 149.99 28.27  147.0  148.49 26.69 88.00 249.0 161.00 0.64
talla       3 125   1.69  0.08    1.7    1.69  0.07  1.52   1.9   0.38 0.10
         kurtosis   se
edad         4.34 0.26
peso_lbs     0.92 2.53
talla       -0.05 0.01
```

Nota importante: Cuando se utiliza `describeBy()`, el paquete `psych` genera una tabla separada para cada combinación de las variables categóricas. Esto puede ser útil para análisis simples, pero puede volverse limitante en casos donde se necesite consolidar los resultados en un solo dataframe o realizar análisis más complejos.

13.4.4 Exportación de resultados

Los resultados del análisis descriptivo pueden exportarse fácilmente a un archivo Excel para su revisión o presentación:

```
# Exportar resultados agrupados a Excel
write_xlsx(resultado_agrupado, "analisis_descriptivo_psych.xlsx")
```

13.4.5 Ventajas del paquete `psych`

El paquete `psych` ofrece varias ventajas para el análisis descriptivo:

1. **Estadísticas más detalladas**: Incluye medidas avanzadas como asimetría, curtosis, errores estándar e intervalos de confianza.
2. **Análisis por grupos**: Permite calcular estadísticas descriptivas para diferentes combinaciones de variables categóricas.
3. **Flexibilidad**: Facilita la categorización de variables numéricas en rangos personalizados.
4. **Exportación de resultados**: Los resultados pueden exportarse fácilmente a formatos como Excel para su análisis posterior.

13.4.6 Limitaciones del paquete `psych`

Aunque el paquete `psych` es muy útil, presenta algunas limitaciones:

1. **Resultados separados por combinación de categorías**: La función `describeBy()` genera una tabla separada para cada combinación de las variables categóricas, lo que puede dificultar la consolidación de los resultados en un solo archivo o dataframe.
2. **Falta de personalización**: No permite agregar estadísticas personalizadas, como percentiles específicos o medidas adicionales que no estén incluidas en las funciones predefinidas.
3. **Manejo de valores faltantes**: Aunque maneja valores faltantes de manera básica, no ofrece opciones avanzadas para imputación o análisis detallado de datos incompletos.
4. **Exportación limitada**: Los resultados no están listos para exportarse directamente en un formato tabular consolidado, lo que requiere pasos adicionales para su preparación.

13.5 Función personalizada para análisis descriptivo completo

Para superar las limitaciones del paquete `psych`, se puede utilizar una función personalizada que ofrezca mayor flexibilidad y personalización. A continuación, se presenta una solución que incluye funciones auxiliares para calcular medidas avanzadas como la moda, asimetría y curtosis.

13.5.1 Establecer funciones auxiliares

13.5.1.1 Moda

```
# Función de 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 = ", "))
}
```

13.5.1.2 Asimetría

```
# Función Asimentría
calcular_asimetria <- function(x) {
  # Eliminar valores NA
  x <- na.omit(x)

  # Verificar que haya suficientes datos
  if (length(x) < 3) return(NA_real_)

  # Calcular media y desviación estándar
  m <- mean(x)
  s <- sd(x)

  # Manejar el caso de desviación estándar cero
  if (s == 0) return(0)

  # Calcular asimetría
  asimetria <- sum((x - m)^3) / (length(x) * s^3)
  return(asimetria)
}
```

13.5.1.3 Curtosis

```
# Función Curtosis
calcular_curtosis <- function(x) {
  # Eliminar valores NA
  x <- na.omit(x)

  # Verificar que haya suficientes datos
  if (length(x) < 4) return(NA_real_)

  # Calcular media y desviación estándar
  m <- mean(x)
  s <- sd(x)

  # Manejar el caso de desviación estándar cero
```

```r
  if (s == 0) return(0)

  # Calcular curtosis
  curtosis <- sum((x - m)^4) / (length(x) * s^4) - 3
  return(curtosis)
}
```

13.5.2 Función principal: análisis por categorías

La función principal realiza el análisis descriptivo agrupando los datos según las variables categóricas especificadas. Calcula estadísticas clave como la media, mediana, moda, desviación estándar, varianza, rango, cuartiles, asimetría y curtosis.

```r
# Función Análisis por Categoría
analisis_por_categoria <- function(datos,
                                   columna_numerica,
                                   columnas_categoricas) {
  datos %>%
    group_by(across(all_of(columnas_categoricas))) %>%
    summarise(
      Variable = columna_numerica,
      N_validos = sum(!is.na(.data[[columna_numerica]])),
      N_missing = sum(is.na(.data[[columna_numerica]])),
      Media = mean(.data[[columna_numerica]], na.rm = TRUE),
      Mediana = median(.data[[columna_numerica]], na.rm = TRUE),
      Moda = moda(.data[[columna_numerica]]),
      Desviacion_estandar = sd(.data[[columna_numerica]], na.rm = TRUE),
      Varianza = var(.data[[columna_numerica]], na.rm = TRUE),
      Rango_min = min(.data[[columna_numerica]], na.rm = TRUE),
      Rango_max = max(.data[[columna_numerica]], na.rm = TRUE),
      Rango = Rango_max - Rango_min,
      IQR = IQR(.data[[columna_numerica]], na.rm = TRUE),
      Q1 = quantile(.data[[columna_numerica]], probs = 0.25, na.rm = TRUE),
      Q2 = quantile(.data[[columna_numerica]], probs = 0.50, na.rm = TRUE),
      Q3 = quantile(.data[[columna_numerica]], probs = 0.75, na.rm = TRUE),
      Asimetria = calcular_asimetria(.data[[columna_numerica]]),
      Curtosis = calcular_curtosis(.data[[columna_numerica]]),
      .groups = 'drop'
    )
}
```

13.5.3 Función para análisis de múltiples variables numéricas

Para analizar varias columnas numéricas simultáneamente, se puede usar la siguiente función, que aplica `analisis_por_categoria` a cada columna numérica especificada:

```r
# Función para anlisar multiples variables numericas simultaneamente
analisis_multiple <- function(datos,
                              columnas_numericas,
                              columnas_categoricas) {
  resultados <- list()
  for (col in columnas_numericas) {
    resultados[[col]] <- analisis_por_categoria(
      datos = datos,
      columna_numerica = col,
      columnas_categoricas = columnas_categoricas
    )
  }
  bind_rows(resultados)  # Combina todos los resultados en un dataframe
}
```

13.5.4 Ejemplo de uso

Para ilustrar el uso de la función personalizada, se realizará un análisis descriptivo completo utilizando la base de datos de estudiantes de la Universidad de San Carlos de Guatemala. Este ejemplo mostrará cómo analizar múltiples variables numéricas categorizadas por diferentes variables cualitativas.

13.5.4.1 Preparación del análisis

Antes de ejecutar el análisis, es necesario definir las columnas numéricas y categóricas que se incluirán en el estudio. Las columnas numéricas representan las variables cuantitativas que se analizarán, mientras que las columnas categóricas se utilizarán para agrupar los datos.

```r
# Definir las columnas numéricas y categóricas
# Variables cuantitativas
columnas_numericas <- c("talla", "peso_lbs", "edad")

# Variables cualitativas
columnas_categoricas <- c("sexo", "trabaja")
```

A continuación, se ejecuta la función `analisis_multiple`, que aplica el análisis descriptivo a cada variable numérica, agrupando los resultados según las categorías especificadas.

```r
# Ejecutar el análisis descriptivo completo
resultados_finales <- analisis_multiple(
  datos = estudiantes,
  columnas_numericas = columnas_numericas,
  columnas_categoricas = columnas_categoricas
)
```

13.5.4.2 Visualización de resultados

Los resultados del análisis se almacenan en un objeto llamado `resultados_finales`, que contiene un resumen estadístico detallado para cada combinación de las variables categóricas. Este resumen incluye medidas como la media, mediana, moda, desviación estándar, varianza, rango, asimetría y curtosis, entre otras.

Para visualizar los resultados directamente en la consola, se utiliza la función `print()`. El resultado es una tabla organizada que muestra las estadísticas descriptivas para cada combinación de las categorías `sexo` y `trabaja`.

```
# Mostrar resultados
print(resultados_finales)
```

```
# A tibble: 12 x 19
   sexo  trabaja Variable N_validos N_missing  Media Mediana Moda
   <fct> <fct>   <chr>        <int>     <int>  <dbl>   <dbl> <chr>
 1 F     1       talla           76         0   1.57    1.57 1.57
 2 F     2       talla          154         0   1.59    1.58 1.6
 3 M     1       talla          105         0   1.70    1.7  1.7
 4 M     2       talla          125         0   1.69    1.7  1.7
 5 F     1       peso_lbs        76         0 127.    122.   130
 6 F     2       peso_lbs       154         0 126.    120    114
 7 M     1       peso_lbs       105         0 156.    152    150
 8 M     2       peso_lbs       125         0 150.    147    150
 9 F     1       edad            76         0  26.0    25    21
10 F     2       edad           154         0  22.6    21    21
11 M     1       edad           105         0  27.6    25    22
12 M     2       edad           125         0  21.6    21    21
# i 11 more variables: Desviacion_estandar <dbl>, Varianza <dbl>,
#   Rango_min <dbl>, Rango_max <dbl>, Rango <dbl>, IQR <dbl>, Q1 <dbl>,
#   Q2 <dbl>, Q3 <dbl>, Asimetria <dbl>, Curtosis <dbl>
```

Además de visualizar los resultados en la consola, es posible exportarlos a un archivo Excel para facilitar su revisión o presentación:

```
# Exportar resultados a Excel para mejor visualización
write_xlsx(resultados_finales, "analisis_descriptivo_estudiantes.xlsx")
```

Este ejemplo demuestra cómo utilizar la función personalizada para realizar un análisis descriptivo completo y categorizado. La tabla resultante proporciona una visión detallada de las características de las variables numéricas, agrupadas por categorías cualitativas. Además, la posibilidad de exportar los resultados a Excel permite compartir y analizar los datos de manera más eficiente.

13.6 Resumen Comparativo: Funciones Base de R, Paquete `psych` y Función Personalizada

A continuación, se presenta una comparación entre el paquete `psych` y la función personalizada para realizar análisis descriptivos, destacando las fortalezas y limitaciones de cada enfoque. Esta comparación permite identificar cuál es la mejor opción según las necesidades específicas del análisis.

Característica	**Funciones Base de R**	**Paquete `psych`**	**Función Personalizada**
Estadísticas avanzadas	Calcula medidas básicas como media, mediana, desviación estándar, varianza y rango.	Incluye medidas como asimetría y curtosis.	Incluye asimetría, curtosis, moda y más estadísticas avanzadas.
Análisis por grupos	Requiere pasos adicionales para agrupar y calcular estadísticas por categorías.	Genera tablas separadas para cada combinación de categorías, lo que puede dificultar la consolidación.	Consolida todos los resultados en un único dataframe, facilitando su manejo y análisis.
Flexibilidad	Limitada a las funciones predefinidas, sin opciones para personalización avanzada.	Limitada a las funciones predefinidas del paquete.	Totalmente personalizable, permitiendo agregar o modificar estadísticas según las necesidades.
Exportación	Requiere pasos adicionales para preparar los resultados antes de exportarlos.	Requiere pasos adicionales para preparar los resultados antes de exportarlos.	Los resultados están listos para exportarse directamente en un formato tabular.
Facilidad de uso	Muy fácil de usar para cálculos básicos, pero limitada en análisis avanzados.	Fácil de usar para análisis avanzados estándar.	Requiere más configuración inicial, pero ofrece mayor control y personalización.
Manejo de valores faltantes	Manejo básico con argumentos como `na.rm = TRUE`.	Manejo básico de valores faltantes.	Permite un manejo avanzado y personalizado de valores faltantes.

Las **funciones base de R** son ideales para cálculos rápidos y sencillos, como la media (`mean()`), mediana (`median()`), desviación estándar (`sd()`), varianza (`var()`), rango (`range()`), y el resumen general (`summary()`). Estas funciones son fáciles de usar y están disponibles de forma predeterminada, lo que las convierte en una excelente opción para análisis básicos. Sin embargo, su alcance es limitado cuando se requiere un análisis más

detallado o agrupado, ya que no incluyen medidas avanzadas como asimetría o curtosis, ni permiten un análisis categorizado sin pasos adicionales.

El **paquete `psych`** es una herramienta poderosa y fácil de usar para realizar análisis descriptivos avanzados, especialmente cuando se busca rapidez y simplicidad en el cálculo de estadísticas estándar. Ofrece medidas avanzadas como asimetría y curtosis, y permite realizar análisis agrupados con la función `describeBy()`. Sin embargo, su enfoque en generar tablas separadas para cada combinación de categorías puede ser una limitación en proyectos que requieren consolidar resultados o realizar análisis más complejos. Además, la personalización de las estadísticas calculadas es limitada, ya que depende de las funciones predefinidas del paquete.

Por otro lado, la **función personalizada** destaca por su flexibilidad y capacidad de adaptación. Permite incluir estadísticas adicionales, como la moda, y consolidar resultados en un único dataframe, lo que facilita su manejo y exportación en un formato listo para su uso. Además, ofrece un control total sobre el análisis, permitiendo adaptarlo a necesidades específicas, como el manejo avanzado de valores faltantes o la categorización de variables numéricas. Esto la convierte en una opción ideal para proyectos que demandan un mayor nivel de personalización y control.