

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

P. Agr. Ludwing Isaí Marroquín Jiménez

4 abr 2025

Tabla de contenidos

Introducción	7
Propósito del manual	7
Organización del manual	7
Pre requisitos	8
Colofón	8
 I Introducción a R y RStudio	 11
1 Conceptos básicos de R	12
1.1 ¿Qué es R?	12
1.1.1 Características principales de R	12
1.1.2 ¿Por qué es especial R?	13
1.2 ¿Qué es RStudio?	13
1.2.1 Características principales de RStudio	13
1.2.2 Beneficios de usar RStudio	14
1.3 Reproducibilidad y replicabilidad en la investigación científica	14
1.3.1 Reproducibilidad	15
1.3.2 Replicabilidad	16
1.3.3 Beneficios de la adopción de R para la Ciencia	16
 2 Instalación y configuración	 17
2.1 Descarga de R y RStudio	17
2.1.1 Descarga de R	17
2.1.2 Descarga de RStudio	17
2.2 Instalación de R y RStudio	18
2.2.1 Instalación de R	18
2.2.2 Instalación de RStudio	18
2.3 Configuración inicial	18
2.3.1 Seleccionar la versión de R	18
2.3.2 Configurar la apariencia de RStudio	19
2.4 Organización de proyectos	20
2.4.1 Crear un proyecto en RStudio	20
2.4.2 Establecer un directorio de trabajo	20
2.4.3 Uso de archivos <code>.Rproj</code>	21
 II Conceptos básicos de R	 22
3 Primeros pasos en R	23
3.1 Creación de scripts en RStudio	23

3.2	Guardado y organización de archivos	23
3.3	Organización de proyectos y directorios	24
3.3.1	Recomendaciones para la organización de archivos	24
3.4	Introducción a los objetos en R	25
3.4.1	Creación de objetos en R	25
3.5	Tipos de objetos en R	25
3.5.1	Objetos Numéricos	25
3.5.2	Objetos de Texto	26
3.5.3	Objetos de Tipo Factor	26
3.5.4	Objetos Lógicos	26
3.6	Conclusión	27
4	Estructura de datos en R	28
4.1	Vectores	28
4.1.1	Creación de vectores	28
4.1.2	Manipulación de vectores	29
4.2	Data frames	30
4.2.1	Creación de data frames	30
4.2.2	Ventajas de un data frame	31
4.2.3	Manipulación de data frames	31
4.3	Listas	32
4.3.1	Creación de listas	32
4.3.2	Acceso a elementos de una lista	32
4.4	Comparación y conclusiones: Data Frames y Listas	33
5	Importación de datos	34
5.1	Configuración inicial	34
5.2	Directorio de trabajo o <i>Working Directory</i>	34
5.2.1	Razones para establecer un directorio de trabajo	34
5.2.2	Configuración automática del directorio de trabajo	35
5.2.3	Beneficios de establecer un directorio de trabajo	35
5.2.4	Consecuencias de no establecer un directorio de trabajo	36
5.2.5	Recomendaciones prácticas	36
5.3	Importación de archivos CSV	36
5.4	Importación de archivos Excel	36
5.4.1	Pasos para importar un archivo Excel	37
5.5	Ejemplo práctico: Base de datos de estudiantes USAC	37
5.5.1	Verificación de datos importados	38
5.5.2	Notas importantes	38
6	Operadores en R	39
6.1	Tipos de operadores en R	39
6.2	Operadores de Asignación	39
6.2.1	Ejemplo práctico	39
6.3	Operadores aritméticos	40
6.3.1	Ejemplo práctico	40
6.4	Operadores lógicos	41
6.4.1	Ejemplo práctico	41

6.5	Operadores de Manipulación de Datos	42
6.5.1	Ejemplo práctico	42
7	Funciones en R	44
7.1	Definición y características de las funciones en R	44
7.1.1	¿Qué es una función?	44
7.1.2	Tipos de funciones	44
7.1.3	Funciones predefinidas en R	45
7.1.4	Diferencias entre funciones predefinidas y personalizadas	46
7.2	Usos y beneficios de las funciones en R	47
7.3	Cómo crear funciones en R: Sintaxis y ejemplos básicos	47
7.3.1	Sintaxis básica	47
7.3.2	Elementos clave de una función	48
7.3.3	Ejemplo básico	48
8	Paquetes en R	49
8.1	¿Qué es un paquete en R?	49
8.1.1	Características principales de los paquetes	49
8.1.2	¿Por qué usar paquetes en R?	49
8.2	Instalación y carga de paquetes	50
8.2.1	Ejemplo básico de instalación	50
8.2.2	Carga de paquetes	50
8.2.3	Automatización de la instalación y carga	50
8.3	Paquetes recomendados para tareas específicas	50
8.3.1	Manipulación de datos	51
8.3.2	Visualización de datos	51
8.3.3	Análisis estadístico básico	51
8.3.4	Ejemplo práctico: Instalación y carga de paquetes esenciales	51
8.4	Ejemplo práctico: Uso de paquetes en un flujo de trabajo	51
8.4.1	Preparación del área de trabajo	52
8.4.2	Importación de los datos	52
8.4.3	Análisis de la varianza	53
8.4.4	Visualización de resultados	53
8.4.5	Exportación de resultados	54
III	Manipulación de datos	55
9	Manipulación de datos con dplyr y tidyr	56
9.1	Introducción a los paquetes dplyr y tidyr	56
9.2	Operaciones básicas con dplyr	56
9.2.1	Filtrar filas con filter()	56
9.2.2	Seleccionar columnas	57
9.2.3	Crear nuevas columnas	57
9.2.4	Agrupar y resumir datos	58
9.3	Uso de pipes (%>%) para mejorar la legibilidad del código	58
9.3.1	Ventajas del uso de pipes	59
9.4	Transformación de datos con tidyr	59
9.4.1	Transformar datos con pivot_longer() y pivot_wider()	59

9.4.2	Ejemplo práctico	60
IV	Visualización de datos	62
10	Visualización de datos	63
10.1	Contexto de la base de datos utilizada	63
10.2	Introducción al paquete <code>ggplot2</code>	63
10.2.1	Estructura básica de un gráfico en <code>ggplot2</code>	63
10.3	Creación de gráficos básicos	65
10.3.1	Importación de la base de datos	65
10.3.2	Histogramas	66
10.3.3	Gráficos de barras	68
10.3.4	Gráficos de dispersión (scatterplots)	69
10.3.5	Boxplots	70
10.3.6	Gráfico de líneas	72
10.4	Personalización de gráficos	73
10.4.1	Personalización de colores	73
10.4.2	Etiquetas y títulos	77
10.4.3	Temas	80
10.4.4	Facetas	82
10.4.5	Ejemplo avanzado de personalización	83
V	Exportación de resultados	86
11	Exportación de resultados	87
11.1	Guardar gráficos con <code>ggsave()</code>	87
11.1.1	Sintaxis General de la Función <code>ggsave()</code>	87
11.1.2	Guardar en formatos PDF y PNG	88
11.2	Guardar Tablas en CSV y Excel	90
11.2.1	Crear un data frame de ejemplo	90
11.2.2	Guardar la tabla en formato CSV	90
11.2.3	Guardar en Formato Excel	91
11.3	Comparación de Formatos	91
VI	Material de apoyo y referencias	92
12	Material de apoyo y referencias	93
12.1	Material de apoyo	93
12.2	Referencias	93
VII	Ejemplos de Análisis Estadístico con R	95
13	Estadística descriptiva usando funciones en R	96
13.1	Medidas principales en estadística descriptiva	96
13.1.1	Medidas de tendencia central	96
13.1.2	Medidas de dispersión	97

13.1.3	Medidas de forma	99
13.2	Base de datos para los ejemplos	101
13.2.1	Preparación del área de trabajo	101
13.2.2	Establecer directorio de trabajo	101
13.2.3	Importar la base de datos	101
13.2.4	Revisar de la estructura de los datos	102
13.2.5	Limpieza de la base de datos	102
13.3	Funciones por defecto en R para estadística descriptiva	103
13.3.1	Medidas de tendencia central	103
13.3.2	Medidas de dispersión	103
13.3.3	Medidas de forma	104
13.3.4	Resumen general con <code>summary()</code>	104
13.4	Paquetes especializados para estadística descriptiva (el paquete <code>psych</code>) . . .	105
13.4.1	Instalación y carga del paquete	105
13.4.2	Análisis descriptivo general	105
13.4.3	Análisis descriptivo categorizado	106
13.4.4	Exportación de resultados	107
13.4.5	Ventajas del paquete <code>psych</code>	107
13.4.6	Limitaciones del paquete <code>psych</code>	108
13.5	Función personalizada para análisis descriptivo completo	108
13.5.1	Establecer funciones auxiliares	108
13.5.2	Función principal: análisis por categorías	110
13.5.3	Función para análisis de múltiples variables numéricas	110
13.5.4	Ejemplo de uso	111
13.6	Resumen Comparativo: Funciones Base de R, Paquete <code>psych</code> y Función Personalizada	113

Introducción

La estadística clásica es una disciplina fundamental en la investigación científica y en la toma de decisiones basada en datos. 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 manual, se abordan conceptos básicos acompañados de ejemplos prácticos que facilitan su comprensión y aplicación en diversos contextos. El objetivo es proporcionar una base sólida que permita aplicar R de manera efectiva en el análisis de datos, incluso sin experiencia previa.

Propósito del manual

El presente manual está diseñado para guiar a principiantes en el uso del lenguaje de programación R, desde su instalación y configuración hasta la aplicación de técnicas estadísticas clásicas. Se abordan temas como la manipulación de datos, la creación de gráficos y la realización de análisis estadísticos básicos. Además, se incluyen ejemplos prácticos que permiten aplicar los conceptos aprendidos en contextos reales.

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 (Ihaka & Gentleman, 1996; R Core Team, 2023). A lo largo de este texto, se exploran las principales características de R y su entorno de desarrollo integrado (IDE), RStudio, destacando su utilidad en proyectos académicos y profesionales. Este manual está dirigido a estudiantes, investigadores y profesionales interesados en adquirir habilidades en programación estadística, con un enfoque en la claridad, organización y reproducibilidad.

Organización del manual

El contenido del manual está estructurado de manera progresiva, comenzando con los conceptos más básicos y avanzando hacia herramientas y técnicas estadísticas 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 manual 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.
7. **Ejemplos de análisis estadístico con R:** Se desarrollan de manera detallada ejemplos del uso de R para desarrollar análisis estadístico de datos.

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.

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/RtmpUJRb09/file1daca064b89 -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      2025-04-04
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 visualización de datos y la investigación 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 (Ihaka & Gentleman, 1996). Desde su creación, R se ha consolidado como una de las herramientas más populares en las comunidades científica, académica y profesional.



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 y análisis multivariado.

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, su comunidad activa desarrolla constantemente nuevos paquetes y recursos, manteniéndolo en constante evolución.

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 (Allaire et al., 2022).



1.2.1 Características principales de RStudio

1. **Interfaz intuitiva y organizada:** RStudio divide su interfaz en paneles que permiten acceder fácilmente a diferentes herramientas y funciones.
2. **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.

3. **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`.
4. **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.
5. **Soporte multiplataforma:** RStudio está disponible para sistemas operativos Windows, macOS y Linux, lo que lo hace accesible para una amplia variedad de usuarios.
6. **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

1. **Eficiencia:** Su diseño permite realizar tareas de análisis de datos de manera más rápida y organizada.
2. **Reproducibilidad:** Las herramientas integradas, como R Markdown y el sistema de proyectos, garantizan que los análisis puedan ser replicados fácilmente.
3. **Accesibilidad:** Su interfaz gráfica es ideal tanto para principiantes como para usuarios avanzados.
4. **Flexibilidad:** Permite trabajar con datos, gráficos, modelos estadísticos y aplicaciones interactivas en un solo entorno.

1.3 Reproducibilidad y replicabilidad en la investigación científica

La reproducibilidad y replicabilidad son pilares fundamentales de la investigación científica. Según Baker (2016), el 64% de los investigadores enfrenta dificultades para replicar estudios previos debido a una documentación insuficiente. Herramientas como Excel o Infostat suelen realizar cálculos en celdas ocultas y gráficos ajustados manualmente, lo que dificulta la replicación exacta de los análisis.

R permite documentar cada paso del análisis mediante scripts, asegurando que los procedimientos puedan ser replicados y reinterpretados en el futuro. Esto no solo mejora la transparencia y la credibilidad científica, sino que también optimiza el tiempo y los recursos al facilitar la reutilización de métodos en nuevos estudios.

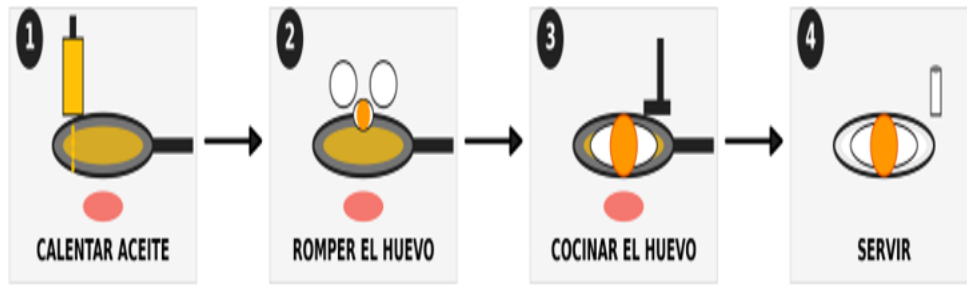


Figura 1.1: Un script en R es comparable a una receta, ya que permite seguir cada paso de manera estructurada para reproducir un análisis específico. Al igual que una receta puede ajustarse al cambiar los ingredientes, un script puede adaptarse para llevar a cabo un análisis distinto utilizando nuevos conjuntos de datos.

1.3.1 Reproducibilidad

La reproducibilidad se define como la capacidad de obtener los mismos resultados al emplear los mismos datos y métodos utilizados en un análisis original. Este concepto constituye un pilar fundamental en la investigación científica, ya que permite verificar y validar los hallazgos de un estudio. Según el informe de las National Academies of Sciences, Engineering, and Medicine (2019), la reproducibilidad asegura que los resultados puedan ser replicados por otros investigadores o incluso por el mismo autor en el futuro, siempre que se disponga de los datos y métodos originales.

Entre las principales características de la reproducibilidad se encuentran:

1. **Disponibilidad de los datos originales:** Es imprescindible que los datos empleados en el análisis estén accesibles para que otros puedan replicar el estudio.
2. **Documentación detallada de los métodos:** Los pasos, herramientas y configuraciones utilizadas deben estar claramente descritos para garantizar la repetición exacta del análisis.
3. **Resultados consistentes:** Al repetir el análisis con los mismos datos y procedimientos, los resultados obtenidos deben ser idénticos.

La reproducibilidad es esencial por varias razones:

1. **Fomenta la transparencia:** Permite que otros investigadores comprendan con precisión cómo se obtuvieron los resultados.
2. **Facilita la verificación:** Ayuda a identificar posibles errores o inconsistencias en el análisis original.
3. **Promueve la colaboración científica:** Proporciona una base sólida para que otros investigadores puedan construir sobre el trabajo existente, ampliando su alcance y aplicabilidad.

1.3.2 Replicabilidad

Por otro lado, la replicabilidad 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 crucial para determinar la robustez y la aplicabilidad de los resultados científicos en nuevos contextos.

Las características clave de la replicabilidad incluyen:

1. **Uso de nuevos datos:** Los datos empleados en el nuevo estudio deben ser diferentes a los del análisis original, pero representar un contexto similar.
2. **Adaptación de los métodos:** Los procedimientos pueden ser ajustados o modificados para adaptarse a las características de los nuevos datos.
3. **Coherencia en los hallazgos:** Aunque los resultados no sean idénticos, deben ser consistentes con los obtenidos en el estudio original.

La replicabilidad aporta beneficios significativos, tales como:

1. **Evaluación de la generalización:** Permite determinar si los resultados del estudio original son aplicables en otros contextos o poblaciones.
2. **Refuerzo de la credibilidad científica:** Incrementa la confianza en los hallazgos al demostrar que no son producto de circunstancias específicas.
3. **Impulso al avance del conocimiento:** Facilita la exploración de nuevas aplicaciones o extensiones de los hallazgos originales, ampliando su impacto en la comunidad científica.

1.3.3 Beneficios de la adopción de R para la Ciencia

El uso de R en la investigación científica ofrece múltiples ventajas que contribuyen a la reproducibilidad y replicabilidad de los estudios:

1. **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).
2. **Eficiencia:** Los métodos desarrollados en R pueden ser reutilizados en nuevos estudios, optimizando recursos y tiempo (Gentleman & Temple Lang, 2007).
3. **Cumplimiento de principios FAIR:** El uso de R facilita la adherencia a los principios FAIR (Findable, Accessible, Interoperable, Reusable), promoviendo una gestión adecuada de los datos científicos (Wilkinson et al., 2016).

2 Instalación y configuración

Antes de comenzar a trabajar con R y RStudio, es fundamental realizar la instalación y configuración de ambos programas. R es un lenguaje de programación y entorno computacional ampliamente utilizado en el análisis estadístico, la visualización de datos y la investigación reproducible (Ihaka & Gentleman, 1996). Por su parte, RStudio es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) diseñado específicamente para trabajar con R, proporcionando una interfaz amigable y herramientas avanzadas que optimizan el flujo de trabajo (Allaire et al., 2022). Esta sección detalla los pasos necesarios para descargar, instalar y configurar ambos programas, asegurando un entorno de trabajo funcional y eficiente.

2.1 Descarga de R y RStudio

Para utilizar R y RStudio, es necesario descargar ambos programas. Mientras que R proporciona el núcleo del lenguaje y las herramientas computacionales, RStudio actúa como una interfaz que simplifica su uso y mejora la experiencia del usuario.

2.1.1 Descarga de R

Es recomendable 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 manual se utiliza la versión **R 4.4.2**, conocida por su estabilidad y amplio soporte (R Core Team, 2023).

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, como macOS y Linux (Allaire et al., 2022).

2.2 Instalación de R y RStudio

La instalación de R y RStudio debe realizarse en un orden específico para evitar conflictos y errores. A continuación, se describen los pasos detallados para cada programa:

2.2.1 Instalación de R

1. Una vez descargado el instalador de R, se debe ejecutar el archivo **.exe**.
2. Seguir las instrucciones proporcionadas por el asistente de instalación.
3. Durante el proceso, se pueden aceptar las configuraciones predeterminadas, a menos que se requiera una configuración personalizada.

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, lo que resulta útil para trabajar con proyectos que requieren versiones específicas del lenguaje (R Core Team, 2023).

2.3 Configuración inicial

Una vez instalados R y RStudio, es recomendable realizar una configuración inicial para optimizar el entorno de trabajo. Estas configuraciones permiten personalizar la experiencia del usuario, mejorar la organización y facilitar el desarrollo de análisis estadísticos. A continuación, se describen los pasos esenciales para configurar RStudio de manera eficiente.

2.3.1 Seleccionar la versión de R

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

1. Ir a **Tools > Global Options > General**.
2. En el apartado **R version**, seleccionar la versión deseada de R.

2.3.2 Configurar la apariencia de RStudio

RStudio ofrece opciones de personalización para adaptar su apariencia a las preferencias del usuario, mejorando la experiencia de trabajo y reduciendo la fatiga visual durante sesiones prolongadas (Allaire et al., 2022). A continuación, se detallan las configuraciones principales:

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.
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 para optimizar el flujo de trabajo. Los pasos para configurar los paneles son:

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. Guardar 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. Para habilitar esta opción:

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.

2.4 Organización de proyectos

La organización de proyectos en RStudio es un aspecto clave para garantizar un flujo de trabajo eficiente, reproducible y bien estructurado. Una adecuada organización no solo facilita la gestión de archivos y scripts, sino que también mejora la colaboración y la reproducibilidad de los análisis. A continuación, se describen las mejores prácticas para organizar proyectos en RStudio.

2.4.1 Crear un proyecto en RStudio

RStudio permite crear proyectos para organizar de manera eficiente los archivos, datos y scripts relacionados con un análisis específico. Los pasos para crear un proyecto son los siguientes:

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. Configurar el nombre y la ubicación del proyecto.
4. Hacer clic en **Create Project** para finalizar la configuración.

El uso de proyectos en RStudio permite mantener una estructura clara y organizada, lo que facilita la gestión de los recursos necesarios para el análisis (Allaire et al., 2022).

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 generados durante el análisis. Para establecer el directorio de trabajo, se puede utilizar la función `setwd()` de la siguiente manera:

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

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

2.4.3 Uso de archivos .Rproj

El archivo **.Rproj** es el núcleo del proyecto en RStudio. Este archivo contiene las configuraciones específicas del proyecto, como el directorio de trabajo, las opciones de visualización y otros ajustes personalizados. Al abrir un archivo **.Rproj**, se cargará automáticamente el entorno de trabajo asociado, lo que facilita la continuidad en el desarrollo del análisis.

2.4.3.1 Beneficios de la organización de proyectos

La organización de proyectos en RStudio ofrece múltiples beneficios que impactan positivamente en la calidad y eficiencia del trabajo:

1. **Reproducibilidad:** Facilita que otros usuarios (o el propio usuario en el futuro) comprendan y reproduzcan el análisis, asegurando que los resultados sean consistentes.
2. **Eficiencia:** Reduce el tiempo perdido buscando archivos o configurando rutas manualmente, permitiendo un enfoque más directo en el análisis.
3. **Colaboración:** Mejora la comunicación y el trabajo en equipo al mantener una estructura clara y consistente, especialmente en proyectos compartidos.
4. **Optimización del flujo de trabajo:** La combinación de una apariencia personalizada y una estructura organizada permite al usuario enfocarse en el análisis de datos de manera más eficiente y profesional (Allaire et al., 2022).

Parte II

Conceptos básicos de R

3 Primeros pasos en R

Comenzar a trabajar con R y RStudio puede parecer un desafío 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 las mejores prácticas para organizar archivos y proyectos. Estas bases son esenciales para garantizar un flujo de trabajo eficiente, reproducible y bien estructurado.

Un script en RStudio no solo es un espacio para escribir código, sino también una herramienta clave para documentar y reproducir análisis de datos. Además, 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 para realizar análisis estadísticos y visualizaciones de datos de manera efectiva.

3.1 Creación de scripts en RStudio

El primer paso para trabajar en RStudio es crear un script, que es un archivo donde se escribe y guarda el código que se ejecutará en R. Existen dos métodos principales para crear un script en RStudio:

1. **Manualmente:** En la barra de menú, seleccionar **File > New File > R Script**.
2. **Utilizando atajos de teclado:** Presionar la combinación de teclas **Ctrl + Shift + N** para crear un nuevo script de manera rápida.

Una vez creado, el script se convierte en el espacio de trabajo principal donde se desarrollarán los análisis y se documentarán los pasos realizados. Es importante guardar el script desde el inicio para evitar la pérdida de trabajo.

3.2 Guardado y organización de archivos

Guardar y organizar los archivos de manera adecuada es fundamental para mantener un flujo de trabajo eficiente y reproducible. Para guardar un script, se debe seleccionar la opción **“Save As...”** en la pestaña **“File”**, lo que permitirá elegir la ubicación y el nombre del archivo. Se recomienda seguir estas prácticas:

1. **Nombres descriptivos y consistentes:** Utilizar nombres que describan claramente el contenido del archivo. Por ejemplo:

Para scripts: `analisis_rendimiento.R` o `2023-10-15_analisis_maiz.R`.

Para datos: `datos_suelo_2023.csv` o `resultados_finales.xlsx`.

2. **Evitar espacios y caracteres especiales:** Los nombres de los archivos no deben incluir espacios ni caracteres especiales, ya que esto puede generar errores 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. **Incluir fechas en un formato estandarizado:** Incorporar fechas en el formato `YYYY-MM-DD` ayuda a identificar rápidamente versiones o actualizaciones. Por ejemplo: `2023-10-15_importacion_datos.R`.

3.3 Organización de proyectos y directorios

La organización adecuada de los archivos en carpetas específicas para cada proyecto es esencial para garantizar un flujo de trabajo ordenado. Si se utiliza un archivo de proyecto **.Rproj**, este configurará automáticamente el directorio de trabajo, lo que facilita la gestión de los archivos relacionados. Se recomienda:

Crear una carpeta específica para cada proyecto: Agrupar todos los elementos relacionados con un proyecto (scripts, datos, resultados) en una sola carpeta.

3.3.1 Recomendaciones para la organización de archivos

La organización de archivos en RStudio es comparable a mantener una cocina bien ordenada: separar los utensilios, ingredientes y especias en lugares específicos facilita el trabajo y evita errores. Para lograrlo, se sugieren las siguientes prácticas:

1. **Estandarizar nombres de archivos:** Usar nombres descriptivos, evitar caracteres especiales y seguir un formato uniforme.
2. **Documentar los pasos del análisis:** Incluir comentarios en los scripts para explicar cada paso del análisis.
3. **Utilizar proyectos de RStudio:** Crear un archivo **.Rproj** para cada proyecto, lo que asegura que el entorno de trabajo esté configurado correctamente.
4. **Realizar copias de seguridad:** Guardar versiones importantes de los archivos en un repositorio de control de versiones (como Git) o en una ubicación segura.

3.4 Introducción a los objetos en R

En R, todo se maneja como un objeto, lo que significa que cualquier dato o estructura que se utilice en este lenguaje se representa como un objeto. Un objeto es una entidad que almacena información y tiene atributos como nombre, tipo y, en algunos casos, dimensiones. Los objetos son fundamentales para trabajar en R, ya que permiten almacenar, manipular y analizar datos de manera eficiente (R Core Team, 2023).

3.4.1 Creación de objetos en R

Para crear un objeto en R, se utiliza un operador de asignación, que puede ser `=` o `<-`. Sin embargo, el uso de `<-` es el estándar recomendado en la comunidad de R, ya que mejora la legibilidad del código y sigue las convenciones establecidas (Ihaka & Gentleman, 1996). Por ejemplo, para asignar el valor 10 a un objeto llamado `x`, se puede escribir:

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

El operador `<-` indica que el valor a la derecha se asigna al nombre del objeto a la izquierda. Este enfoque es especialmente útil para mantener un código claro y organizado.

3.5 Tipos de objetos en R

En R, los objetos pueden clasificarse en diferentes tipos según el tipo de datos que almacenan. Los principales tipos de objetos son: numéricos, texto, factores y lógicos. A continuación, se describen en detalle cada uno de ellos.

3.5.1 Objetos Numéricos

Los objetos numéricos almacenan datos como números enteros o decimales. Son esenciales para representar variables cuantitativas, como edad, altura o peso. En R, los números se manejan como valores de punto flotante por defecto, lo que significa que incluso los enteros se tratan como números decimales.

Ejemplo de creación de objetos numéricos:

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

Nota importante: 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. Los comentarios son útiles para documentar el código, explicar pasos específicos y facilitar su comprensión tanto para otros usuarios como para el propio autor en el futuro.

3.5.2 Objetos de Texto

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

Ejemplo de creación de objetos de texto:

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

Los objetos de texto son fundamentales para trabajar con datos categóricos o descriptivos, como nombres de variables, etiquetas de gráficos o valores de texto en bases de datos.

3.5.3 Objetos de Tipo Factor

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

Ejemplo de creación de objetos tipo factor:

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

En este ejemplo, `estado_civil` y `sexo` son factores con un único nivel. Los factores también pueden tener múltiples niveles, que se definen explícitamente al crearlos. Por ejemplo:

```
# Creación de un factor con múltiples niveles
estado_civil <- factor("soltero", levels = c("soltero", "casado", "divorciado"))
sexo <- factor("masculino", levels = c("masculino", "femenino"))
```

3.5.4 Objetos Lógicos

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

Ejemplo de creación de objetos lógicos:

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

```
[1] TRUE
```

En este caso, la comparación `edad >= 18` evalúa si el valor de `edad` es mayor o igual a 18. El resultado (`TRUE` o `FALSE`) se almacena en el objeto `mayoria_de_edad`.

Los objetos lógicos son ampliamente utilizados en tareas como la selección de datos, la creación de condiciones en bucles y la evaluación de reglas en análisis estadísticos.

3.6 Conclusión

Los objetos son la base del trabajo en R, ya que permiten almacenar y manipular datos de manera estructurada y eficiente. Comprender los diferentes tipos de objetos y cómo crearlos es un paso esencial para aprovechar al máximo las capacidades de este lenguaje. Desde datos numéricos hasta factores y valores lógicos, cada tipo de objeto tiene aplicaciones específicas que facilitan el análisis estadístico y la visualización de datos (R Core Team, 2023; Ihaka & Gentleman, 1996).

4 Estructura de datos en R

En R, las estructuras de datos son esenciales 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 de datos. 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 (R Core Team, 2023).

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 (Ihaka & Gentleman, 1996).

Los vectores son esenciales en R porque muchas otras estructuras de datos, como los *data frames* y las *matrices*, están construidas a partir de ellos. Una característica clave de los vectores es que todos sus elementos 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 en R, se utiliza la función `c()`, que permite combinar elementos separados por comas. A continuación, se presentan ejemplos de creación de vectores de diferentes tipos:

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

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

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

En estos ejemplos:

1. El vector `edades` almacena valores numéricos.

2. El vector `nombres` contiene cadenas de texto.
3. El vector `mayores_de_edad` almacena valores lógicos (`TRUE` o `FALSE`) que resultan de una comparación lógica.

4.1.2 Manipulación de vectores

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

4.1.2.1 Acceso a elementos específicos

Se pueden acceder a elementos individuales de un vector utilizando índices entre corchetes (`[]`). Los índices en R comienzan en 1.

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

```
[1] 17
```

En este ejemplo, `edades[1]` devuelve el primer elemento del vector `edades`, que es 17.

4.1.2.2 Filtrado de elementos

Es posible filtrar elementos de un vector aplicando condiciones lógicas. Esto resulta útil para seleccionar subconjuntos de datos.

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

```
[1] 25
```

En este caso, la condición `edades > 20` devuelve un vector con los valores que cumplen la condición, es decir, las edades mayores a 20.

4.1.2.3 Combinación de vectores

Los vectores pueden combinarse para crear nuevos vectores utilizando la función `c()`.

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

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

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

4.2 Data frames

Un *data frame* es una estructura de datos tabular en R, similar a una hoja de cálculo o una tabla en una base de datos. Organiza los datos en filas y columnas, donde cada columna es un vector que puede contener un tipo de dato diferente (por ejemplo, números, texto o factores). Cada fila representa una observación, mientras que cada columna corresponde a una variable (R Core Team, 2023).

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 una amplia variedad de funciones y paquetes en R, lo que los convierte en una de las estructuras más utilizadas en este lenguaje (Wickham & Grolemund, 2017).

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. A continuación, se presenta un ejemplo usando los vectores creados en la sección anterior:

```
# 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

En este ejemplo:

1. La columna `nombres` contiene texto.
2. La columna `edades` almacena valores numéricos.
3. La columna `mayores_de_edad` contiene valores lógicos (`TRUE` o `FALSE`).

4.2.2 Ventajas de un data frame

Los *data frames* ofrecen múltiples ventajas que los hacen indispensables para el análisis de datos en R:

1. **Estructura clara:** Cada fila representa una observación y cada columna una variable, lo que facilita la interpretación de los datos.
2. **Compatibilidad:** Son compatibles con funciones estadísticas y de visualización, así como con paquetes populares como `ggplot2` y `dplyr`.
3. **Flexibilidad:** Permiten almacenar diferentes tipos de datos en columnas, como números, texto y factores.

4.2.3 Manipulación de data frames

Los *data frames* pueden manipularse fácilmente utilizando funciones básicas de R o herramientas avanzadas de paquetes como `dplyr`. A continuación, se describen algunas operaciones comunes:

4.2.3.1 Acceso a columnas

Se puede acceder a una columna específica utilizando el operador `$`:

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

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

4.2.3.2 Filtrado de filas

Es posible filtrar filas aplicando condiciones lógicas. Por ejemplo, para seleccionar las filas donde la edad sea mayor a 20:

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

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

4.3 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. A diferencia de los *data frames*, las listas no requieren que los elementos tengan la misma longitud o tipo de dato, lo que las hace ideales para almacenar resultados complejos o datos heterogéneos (R Core Team, 2023).

4.3.1 Creación de listas

Para crear una lista, se utiliza la función `list()`. A continuación, se presenta un ejemplo:

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

En este ejemplo:

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

4.3.2 Acceso a elementos de una lista

Los elementos de una lista pueden accederse mediante su nombre o índice:

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

```
# Acceso al elemento 'nombres' por nombre
mi_lista$nombres
```

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

Por índice: Utilizando corchetes dobles `[[]]`:

```
# Acceso al primer elemento de la lista
mi_lista[[1]]
```

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


4.4 Comparación y conclusiones: Data Frames y Listas

Los *data frames* y las listas son estructuras de datos fundamentales en R, cada una con características específicas que las hacen adecuadas para diferentes tareas. Mientras que los *data frames* son ideales para trabajar con datos tabulares estructurados, las listas ofrecen una mayor flexibilidad para almacenar y manipular datos heterogéneos. A continuación, se presenta una comparación detallada entre ambas estructuras:

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 esenciales en R, y su elección depende del tipo de datos y del análisis que se desee realizar. En el contexto del análisis estadístico, los *data frames* se utilizan habitualmente para almacenar datos tabulares, como encuestas o resultados experimentales, mientras que las listas son más adecuadas para almacenar resultados complejos, como salidas de modelos estadísticos, pruebas o análisis que combinan diferentes tipos de datos.

Por ejemplo:

1. Un *data frame* puede contener las respuestas de una encuesta, con columnas que representan variables como edad, género y puntuaciones.
2. Una lista puede almacenar los resultados de un análisis estadístico, como coeficientes de regresión, valores p y gráficos generados.

Comprender las diferencias y aplicaciones de estas estructuras permite aprovechar al máximo las capacidades de R en el análisis de datos. La elección adecuada entre un *data frame* y una lista no solo facilita el manejo de los datos, sino que también optimiza el flujo de trabajo y la reproducibilidad del análisis (R Core Team, 2023; Wickham & Grolemund, 2017).

5 Importación de datos

La importación de datos es uno de los primeros pasos esenciales en cualquier análisis estadístico. R ofrece diversas herramientas y funciones para importar datos desde diferentes fuentes, como archivos CSV, Excel o páginas web en formato HTML. La capacidad de importar datos de manera eficiente y reproducible permite trabajar con grandes volúmenes de información de forma sistemática (R Core Team, 2023).

5.1 Configuración inicial

Antes de importar datos, es importante asegurarse de que el directorio de trabajo esté correctamente configurado. Cuando se trabaja con proyectos de RStudio (`.Rproj`), el directorio se configura automáticamente. Sin embargo, para scripts independientes, se puede establecer el directorio usando la función `setwd()`:

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

5.2 Directorio de trabajo o *Working Directory*

El directorio de trabajo en R es el “punto de partida” desde donde el software buscará automáticamente los archivos necesarios (como datos y scripts) y guardará los resultados generados. Establecer correctamente el directorio de trabajo es esencial para garantizar la organización, la reproducibilidad y la eficiencia del análisis, especialmente cuando se trabaja con scripts independientes que no forman parte de un proyecto en RStudio. Si no se configura adecuadamente, se corre el riesgo de enfrentar errores como “archivo no encontrado” o de que el código deje de funcionar al mover el proyecto a otra ubicación o computadora (R Core Team, 2023).

5.2.1 Razones para establecer un directorio de trabajo

Configurar un directorio de trabajo adecuado ofrece múltiples beneficios:

1. **Reproducibilidad:** Permite que el código funcione en cualquier computadora sin depender de rutas absolutas como `C:/Usuario/MiPC/...`. Esto asegura que el análisis sea portable y replicable.
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.

5.2.2 Configuración automática del directorio de trabajo

Cuando se trabaja con scripts independientes, se puede establecer automáticamente el directorio de trabajo como la carpeta donde está guardado el script. Esto se logra utilizando el paquete `rstudioapi`:

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

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

5.2.2.1 Explicación del código

1. `if (!require("rstudioapi")) install.packages("rstudioapi")`: Esta estructura condicional verifica si el paquete `rstudioapi` está instalado. Si no lo está, lo instala automáticamente desde CRAN.
2. `rstudioapi::getActiveDocumentContext()$path`: Obtiene la ruta completa del script actual. Por ejemplo, si el script está ubicado en `C:/proyecto/scripts/analisis.R`, esta función devuelve esa ruta completa.
3. `dirname()`: Extrae la carpeta que contiene el script. Siguiendo el ejemplo anterior, convierte `C:/proyecto/scripts/analisis.R` en `C:/proyecto/scripts/`.
4. `setwd()`: Establece la carpeta extraída como el directorio de trabajo. Esto asegura que cualquier archivo en esa carpeta pueda ser accedido directamente sin necesidad de especificar rutas completas.

5.2.3 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 adicionales.
2. **Automatización:** No es necesario modificar manualmente las rutas al compartir el script con otros usuarios, lo que facilita la colaboración.
3. **Reproducibilidad:** Garantiza que el análisis pueda ser replicado en diferentes entornos sin problemas de rutas.

5.2.4 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.

5.2.5 Recomendaciones prácticas

1. **Guardar el script antes de configurar el directorio:** Es importante guardar el script antes de ejecutar la línea de configuración automática, ya que R necesita conocer su ubicación para establecer el directorio de trabajo correctamente.
2. **Verificar el directorio actual:** Antes de importar datos o guardar resultados, se puede verificar el directorio de trabajo actual utilizando la función `getwd()`:

```
# Verificar el directorio de trabajo actual  
getwd()
```

3. **Usar proyectos en RStudio:** Siempre que sea posible, se recomienda trabajar dentro de un proyecto de RStudio (`.Rproj`), ya que este configura automáticamente el directorio de trabajo y facilita la organización.

5.3 Importación de archivos CSV

Los archivos CSV (Comma-Separated Values) son uno de los formatos más comunes para almacenar datos tabulares. Según Wickham (2016), estos archivos son preferidos por su simplicidad y portabilidad. Para importar un archivo CSV, se utiliza la función `read.csv()`:

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

Parámetros importantes:

1. **header = :** Indica si el archivo tiene una fila de encabezado (TRUE) o no (FALSE).
2. **sep = :** Especifica el separador entre valores (, , ; , \t, etc.).

5.4 Importación de archivos Excel

Para importar archivos Excel (`.xlsx`), se utiliza el paquete `readxl` (Wickham & Bryan, 2023). Este paquete ofrece funciones especializadas para leer datos de hojas de cálculo de Excel sin necesidad de conversiones previas.

5.4.1 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")
```

2. Importar el archivo Excel

```
# Importar un archivo Excel
datos_excel <- read_excel("ruta/del/archivo/datos.xlsx",
                          sheet = "Hoja1", #Nombre de la hoja
                          col_names = TRUE/FALSE) #Primera fila como encabezado
```

Parámetros importantes:

1. **sheet** = : Especifica la hoja del archivo que se desea importar.
2. **col_names** = : Indica si el archivo tiene una fila de encabezado (TRUE) o no (FALSE).

5.5 Ejemplo práctico: Base de datos de estudiantes USAC

Para ilustrar estos conceptos, utilizaremos una base de datos recopilada en 2002 en la Universidad de San Carlos de Guatemala, que contiene información de 460 estudiantes de diversas facultades. Esta base de datos está disponible en ambos formatos ([CSV](#) y [Excel](#)) y será utilizada a lo largo del manual.

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

# Importar datos en formato CSV
datos_csv <- read_csv("datos_estudiantes.csv",
                     header = TRUE,
                     sep = ",")

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

# Visualizar los primeros registros
head(datos_csv) # Muestra las primeras 6 filas
```

5.5.1 Verificación de datos importados

Después de importar los datos, es importante verificar que la importación se realizó correctamente. R ofrece varias funciones para este propósito:

```
# Estructura del data frame
str(datos_csv)

# Resumen estadístico básico
summary(datos_csv)

# Dimensiones del data frame
dim(datos_csv)
```

5.5.2 Notas importantes

1. **Ubicación de archivos:** Cuando se trabaja con proyectos de RStudio, los archivos deben estar en el directorio del proyecto para facilitar su acceso.
2. **Codificación de caracteres:** En caso de problemas con caracteres especiales (ñ, tildes), se puede especificar la codificación:

```
datos <- read.csv("archivo.csv", encoding = "UTF-8")
```

6 Operadores en R

En R, los operadores son herramientas fundamentales que permiten realizar cálculos, comparaciones, asignaciones y manipulaciones de datos. Son el equivalente a las herramientas básicas de un taller, que se combinan para construir soluciones más complejas. Comprender su funcionamiento es esencial para aprovechar al máximo las capacidades del lenguaje en el análisis estadístico y la programación (R Core Team, 2023).

Los operadores en R se clasifican en diferentes categorías según su función. A continuación, se describen los principales tipos de operadores disponibles en el lenguaje, junto con ejemplos prácticos para ilustrar su uso.

6.1 Tipos de operadores en R

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

6.2 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 (Ihaka & Gentleman, 1996).

6.2.1 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)

# 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 (`==`).

6.3 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.

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.

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

6.3.1 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.

```
# 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

```

6.4 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.

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

6.4.1 Ejemplo práctico

Los operadores lógicos son especialmente útiles en tareas como la selección de datos, la creación de condiciones en bucles y la evaluación de reglas en análisis estadísticos (R Core Team, 2023).

```

# 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.

6.5 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.

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

6.5.1 Ejemplo práctico

```
# 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. Comprender cómo funcionan las funciones y cómo crearlas es esencial para aprovechar al máximo las capacidades de R en el análisis estadístico y la programación (R Core Team, 2023; Wickham & Golemund, 2017).

En esta sección, 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 (Chambers, 2008).

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, las funciones se dividen en dos categorías principales:

1. **Funciones predefinidas (built-in):** Estas funciones vienen incluidas en R o en paquetes adicionales y están diseñadas para realizar tareas comunes, como cálculos matemáticos, operaciones estadísticas, manipulación de datos y visualización.
2. **Funciones personalizadas (user-defined):** Estas son creadas por el usuario para realizar tareas específicas que no están cubiertas por las funciones predefinidas. Son útiles para automatizar procesos repetitivos o realizar cálculos complejos.

7.1.3 Funciones predefinidas en R

Las funciones predefinidas son herramientas esenciales en R, ya que permiten realizar operaciones comunes de manera rápida y eficiente. A continuación, se presentan algunos ejemplos de funciones predefinidas ampliamente utilizadas:

Ejemplos de funciones predefinidas:

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

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

```
[1] 3
```

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

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

```
[1] 15
```

3. `sd()`: Calcula la desviación estándar.

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

```
[1] 1.581139
```

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

```
# 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 en R (R Core Team, 2023).

7.1.3.1 Funciones personalizadas en R

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:

```
# 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:

1. El usuario define la lógica de la función.
2. Especifica los argumentos necesarios (**radio**).
3. Utiliza la función para realizar cálculos.

7.1.4 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	<code>mean()</code> , <code>sum()</code> , <code>sd()</code> , <code>summary()</code>	<code>calcular_area_circulo()</code>
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 usuario. 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:

```
# Función 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 amplía 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 automáticamente
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
```

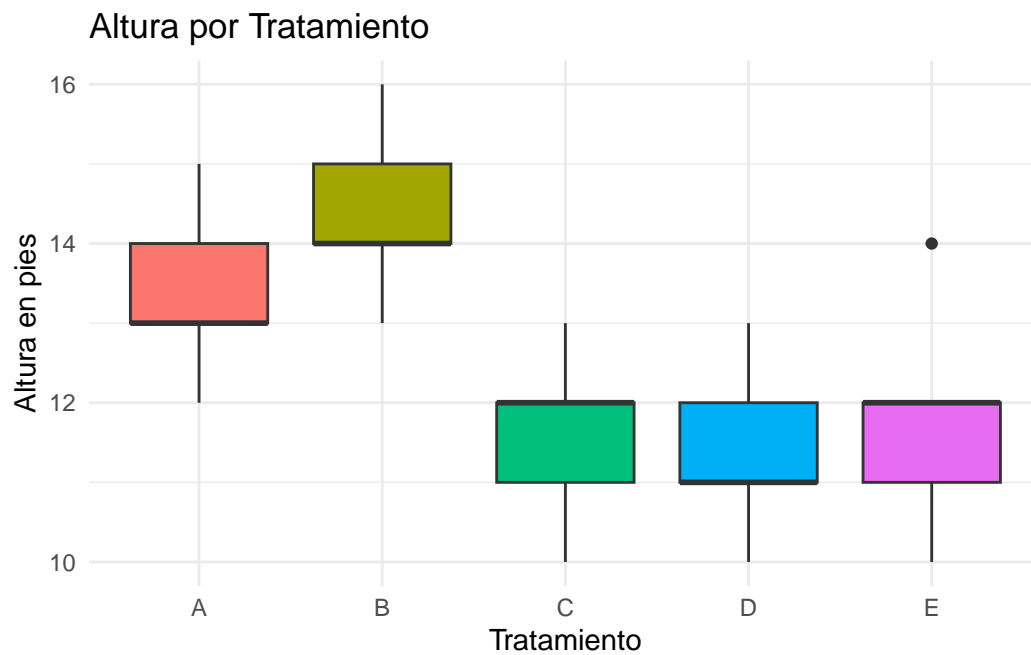
```
# Prueba de Tukey
comparacion_tukey <- HSD.test(modelo_anova, "tratamiento")
print(comparacion_tukey$groups)
```

```
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.


```
# 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:

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

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

```
# 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:

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

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

```
# 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:

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

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

```
# 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:

```
# 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 recopilaban 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**.

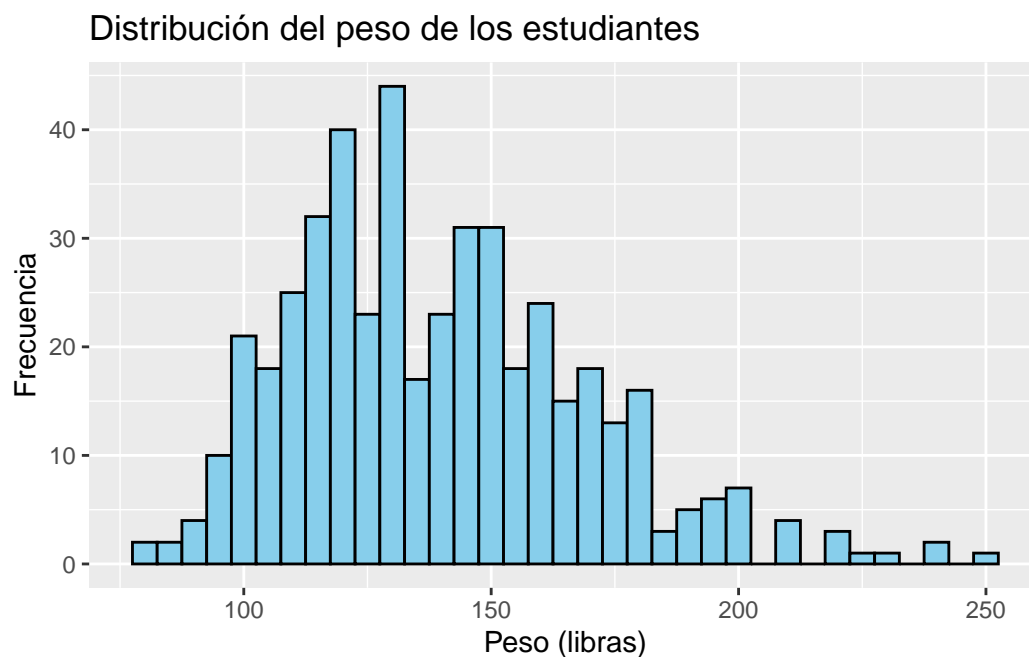
10.3.2 Histogramas

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.

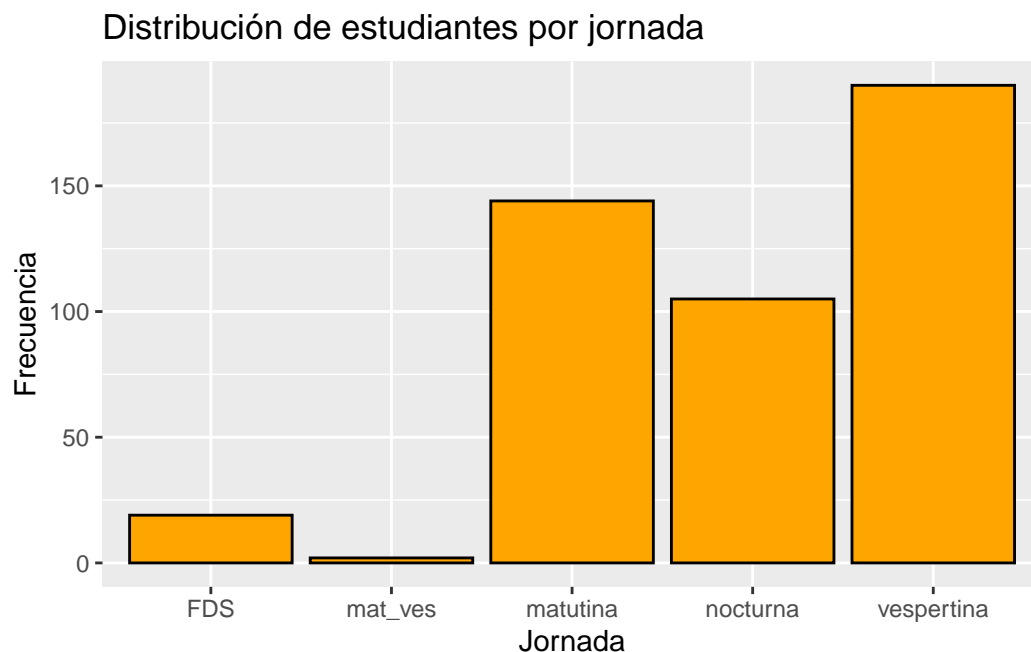
10.3.3 Gráficos de barras

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.

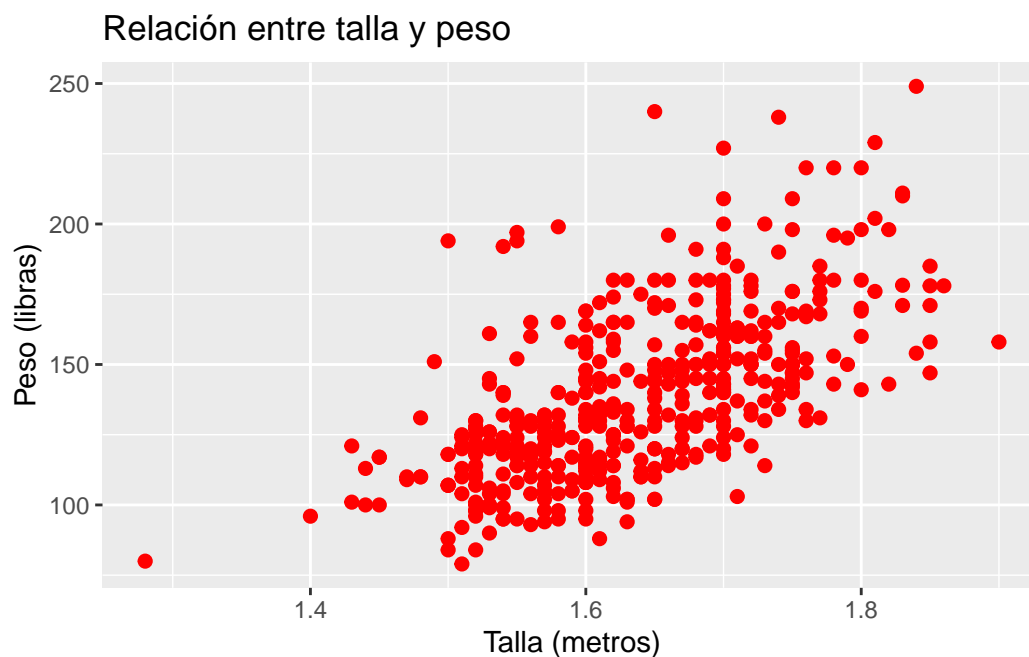
10.3.4 Gráficos de dispersión (scatterplots)

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.

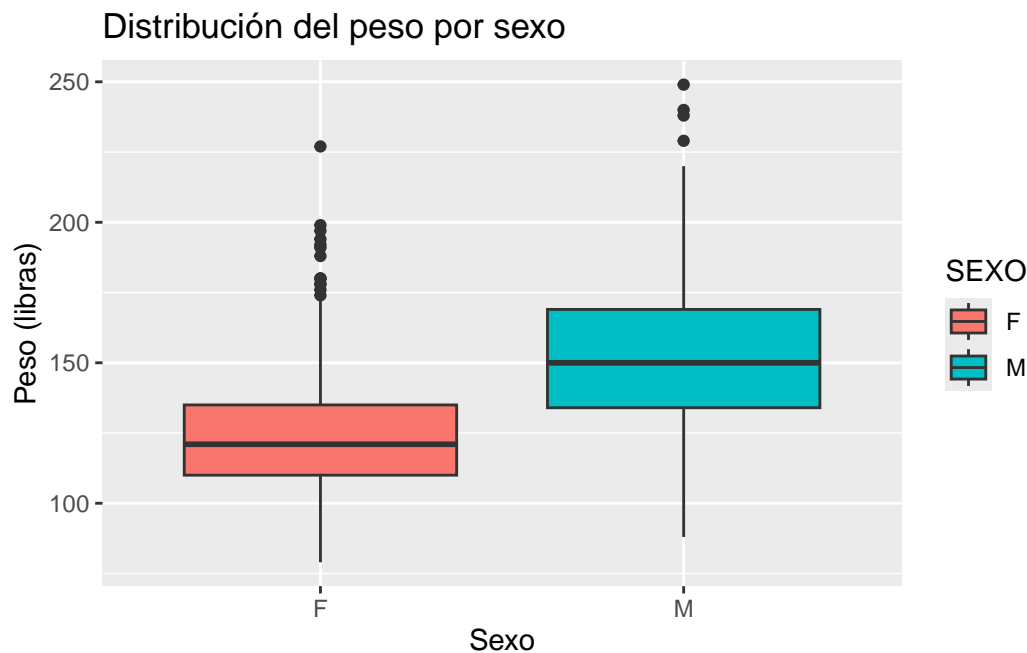
10.3.5 Boxplots

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:

```
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.

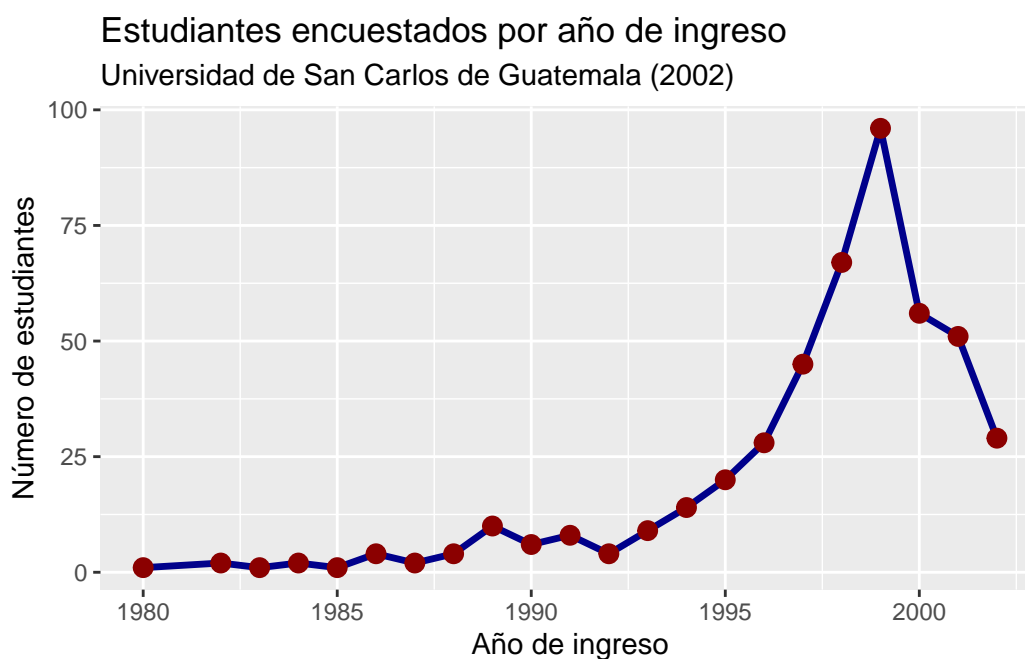
10.3.6 Gráfico de líneas

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.

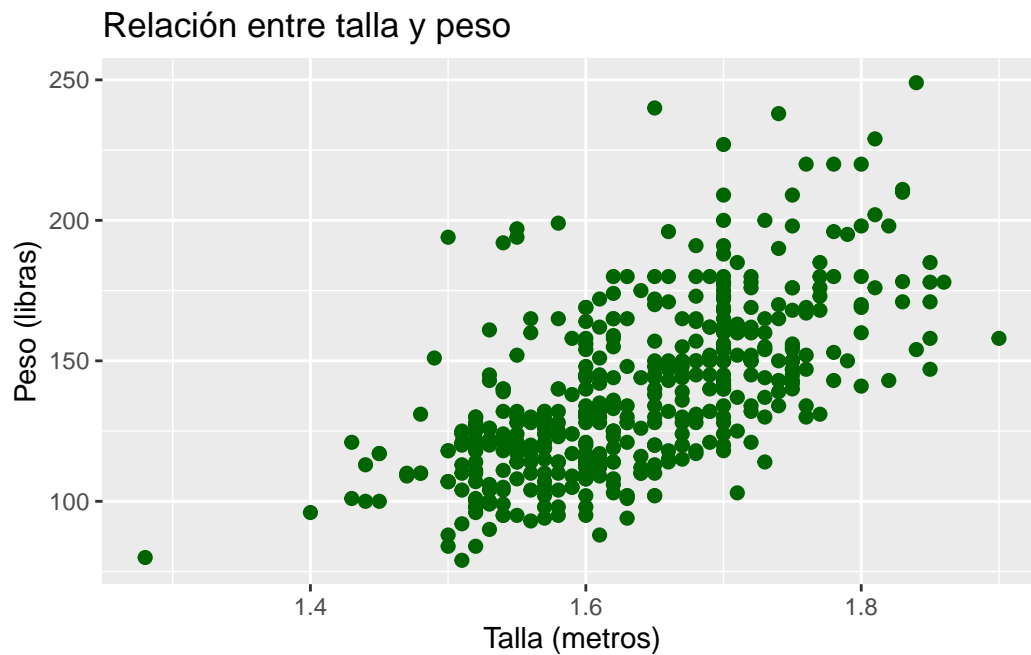
10.4.1 Personalización de colores

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:

```
# 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)")
```



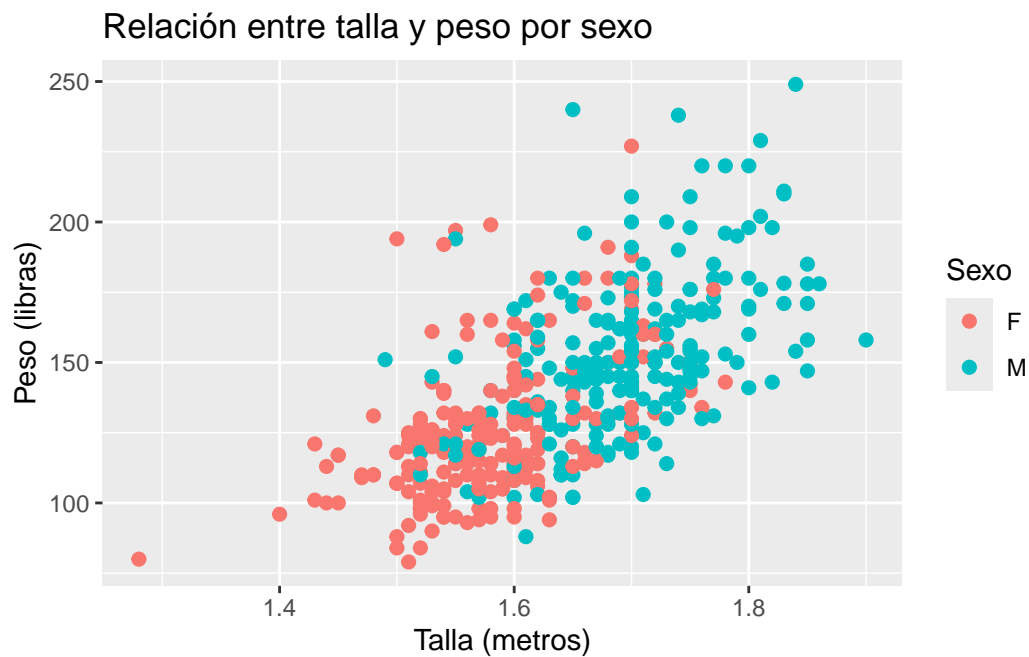
En este ejemplo:

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()`:

```
# 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")
```



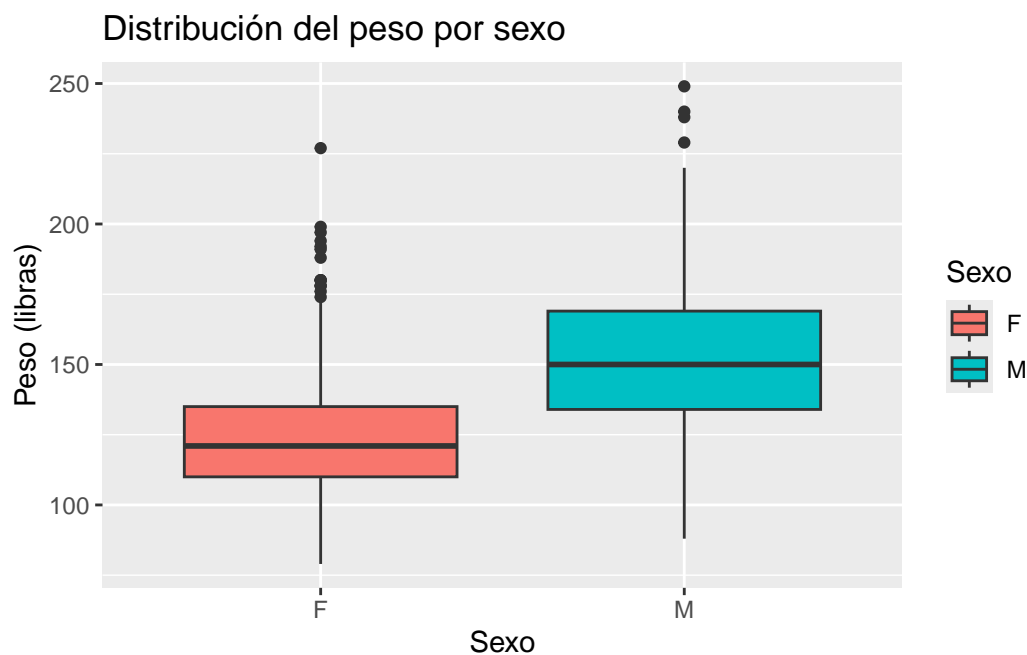
En este ejemplo:

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")
```

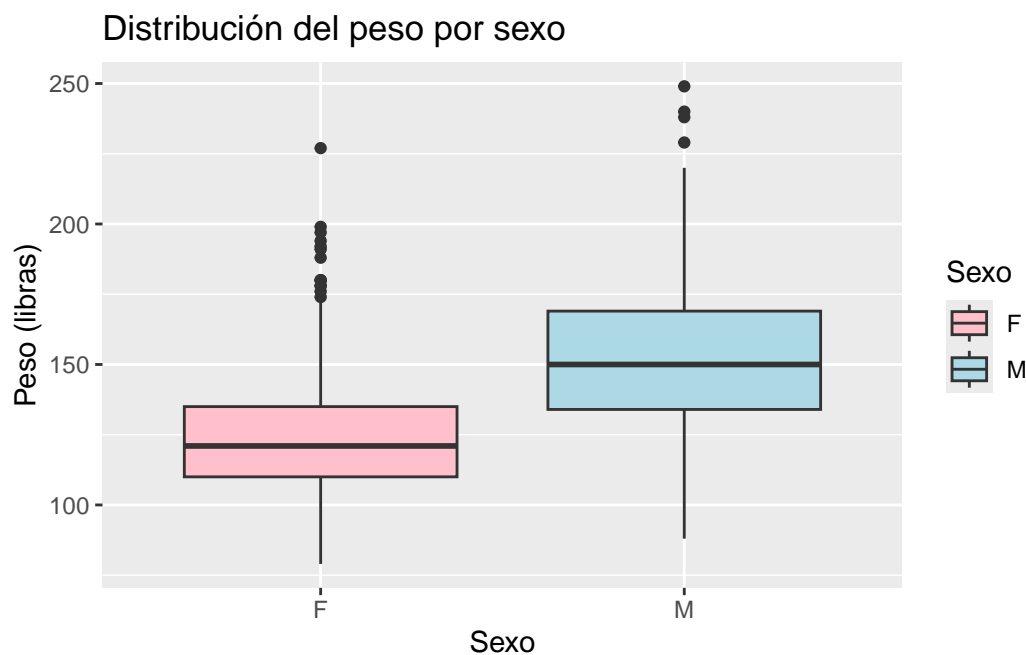


En este ejemplo, las cajas del *boxplot* se rellenan con colores diferentes según la variable SEXO.

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 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

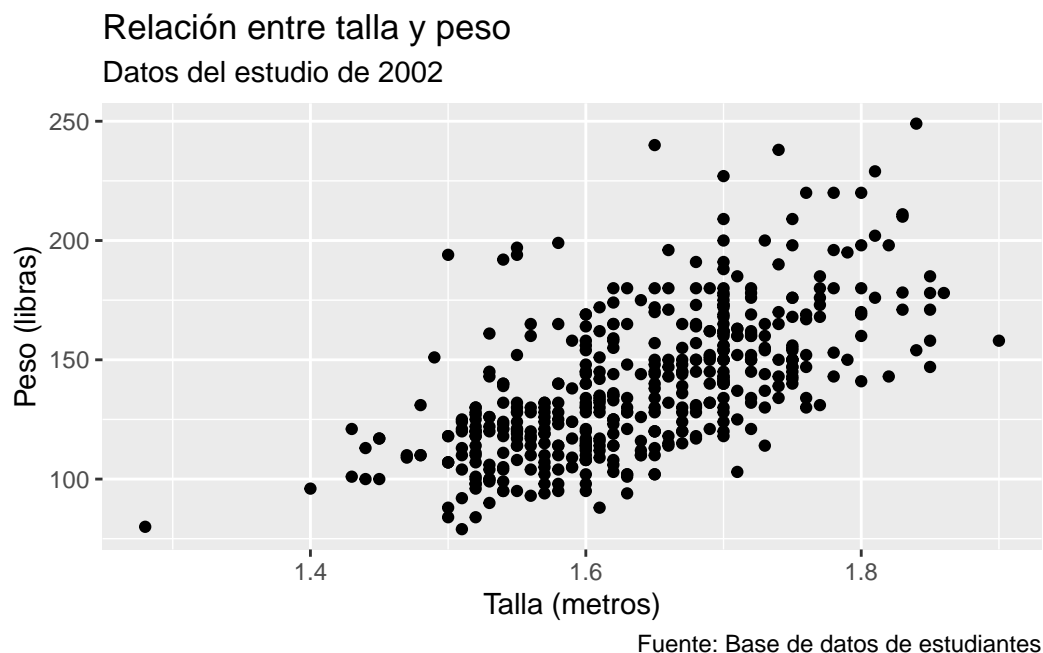
La función `labs()` se utiliza para añadir y personalizar los siguientes elementos:

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")
```



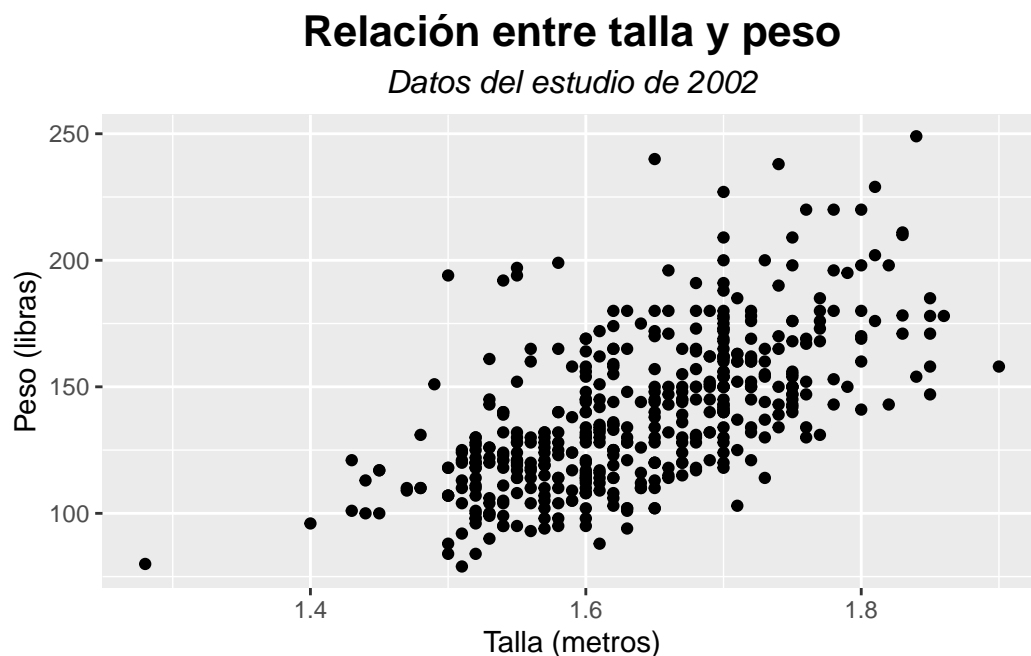
En este ejemplo:

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()`:

```
# 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"))
```



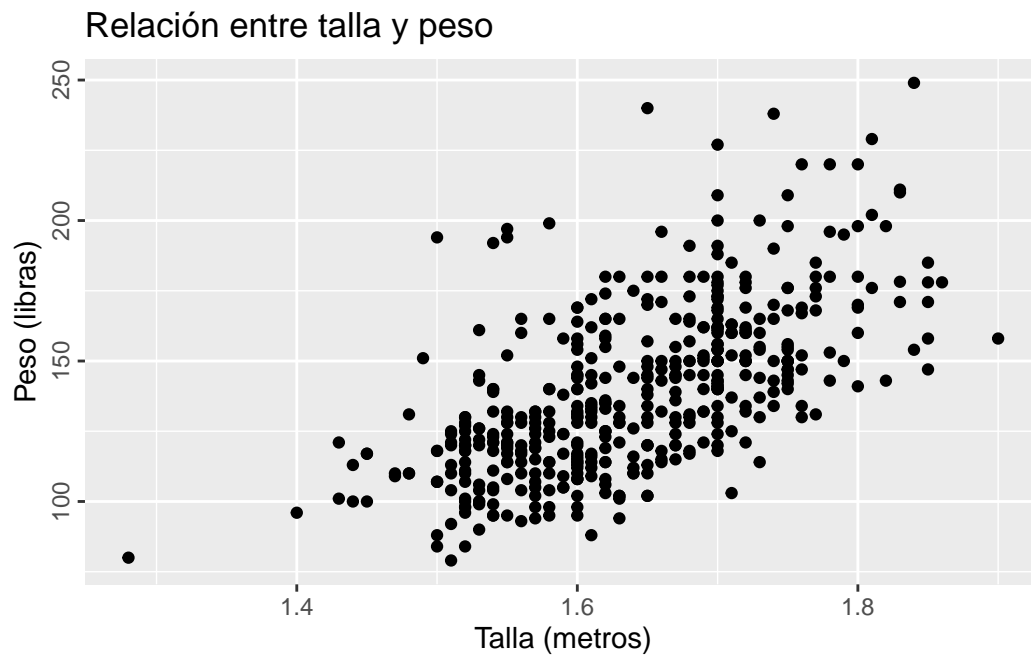
En este ejemplo:

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:

```
# 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:

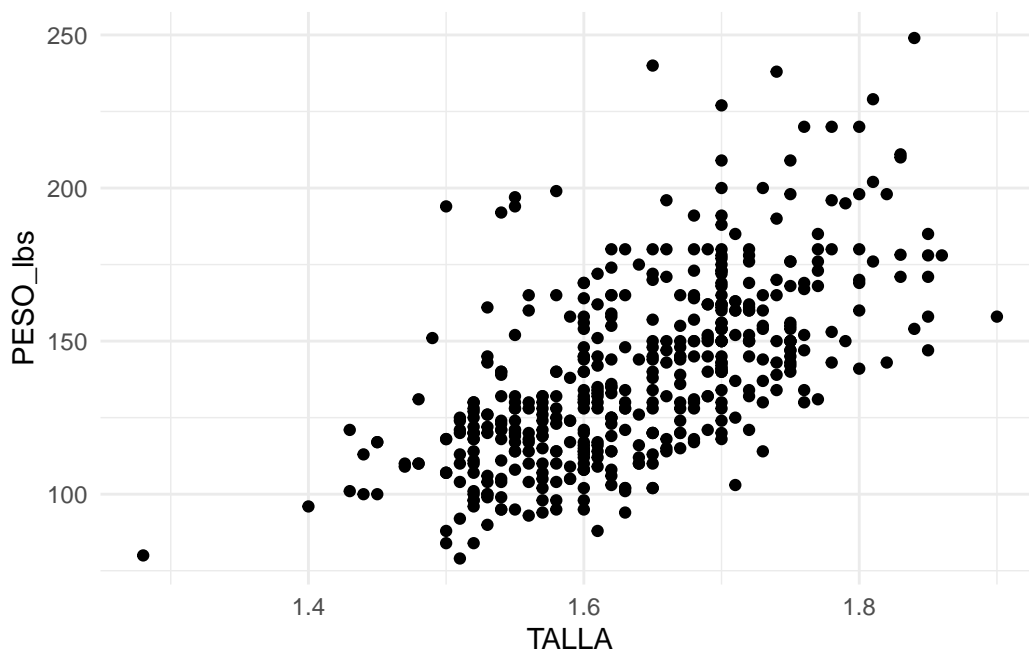
1. **theme_minimal()**: Un diseño limpio y moderno, con un fondo blanco y líneas simples.
2. **theme_classic()**: Un estilo clásico con líneas de ejes visibles y sin cuadrícula.
3. **theme_light()**: Similar a **theme_minimal()**, pero con cuadrículas más visibles.

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()
```



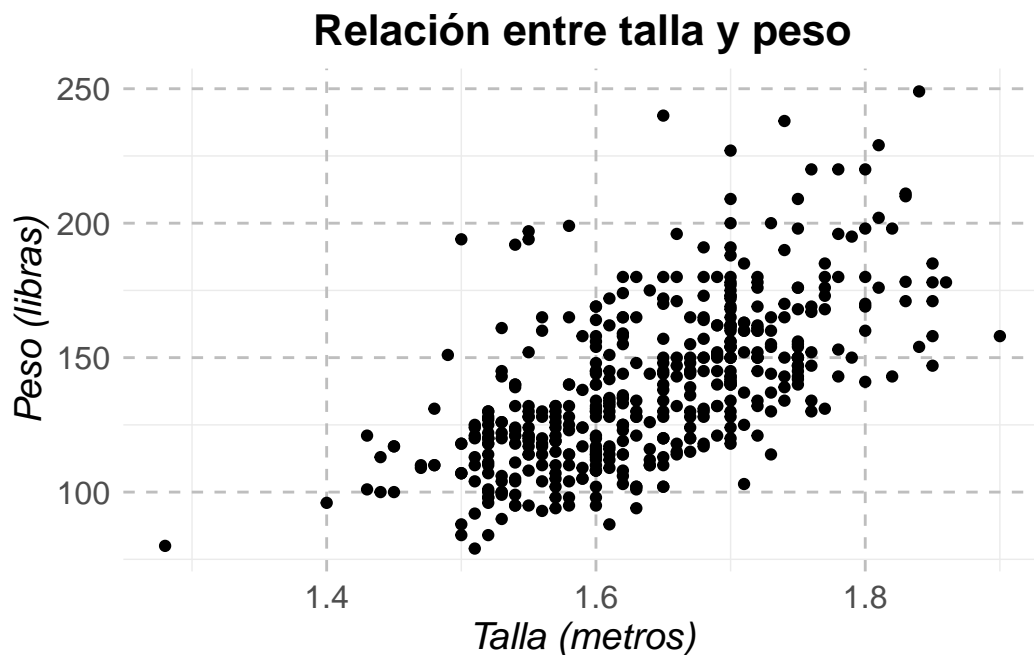
En este ejemplo: `theme_minimal()` elimina elementos innecesarios, como bordes y fondos grises, dejando un diseño limpio y profesional.

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.

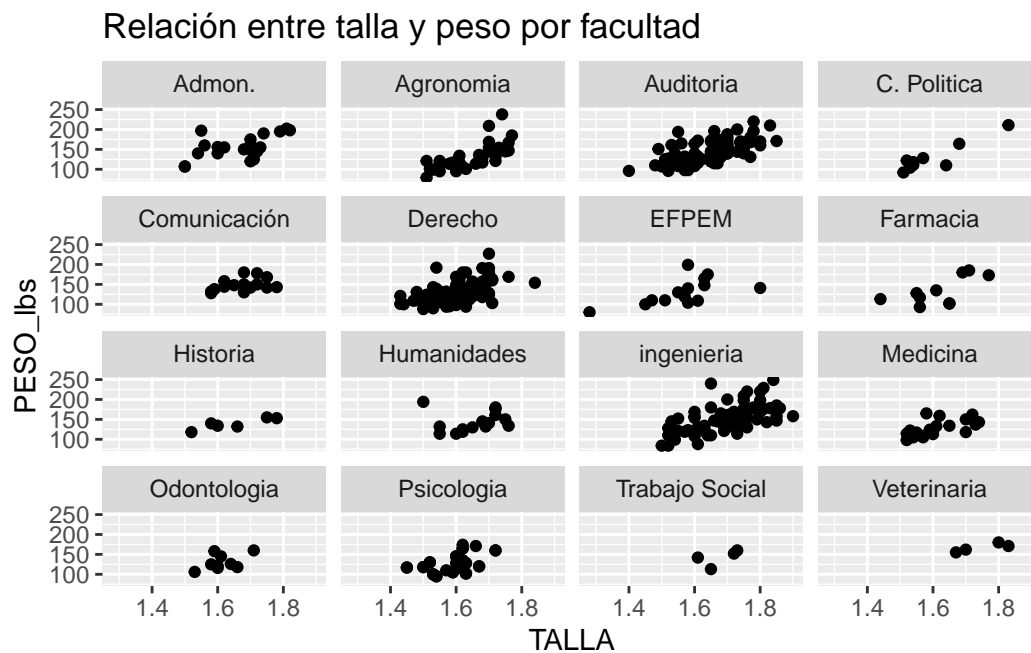
10.4.4 Facetas

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:

```
# 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

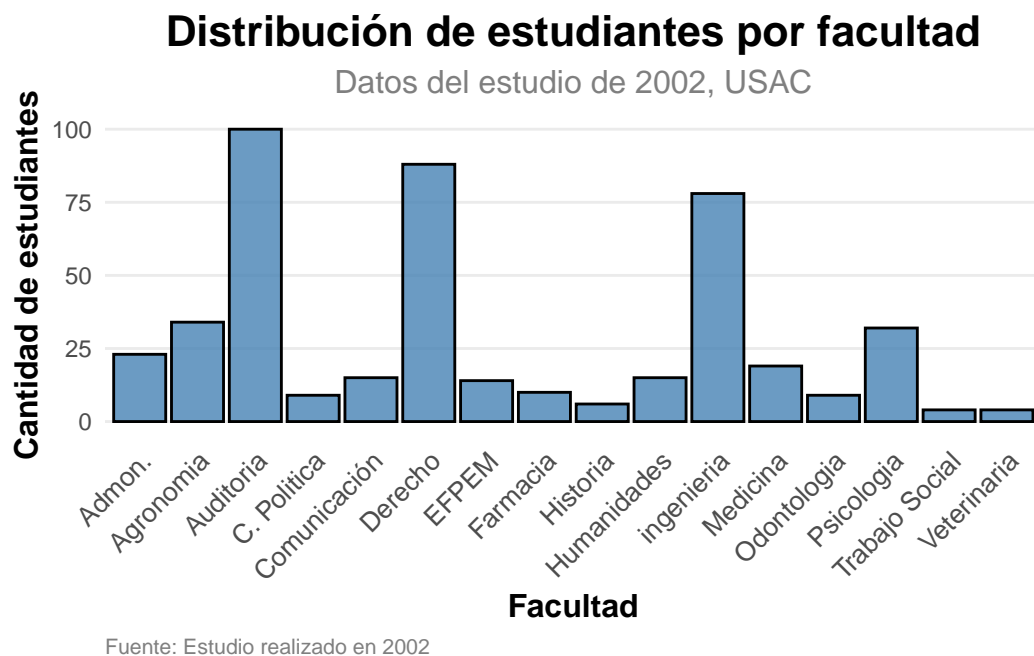
La flexibilidad y la lógica de capas de **ggplot2** permiten crear gráficos con un alto grado de personalización, adaptados a necesidades específicas y con un diseño profesional. A continuación, se presenta un ejemplo de un gráfico de barras con personalización detallada:

```
# 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 último gráfico realizado en el capítulo anterior:


```
# 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()`:

```
# 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.

```
# 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:

```
# 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:

```
# 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 Golemund

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-golemund>

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:

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

Ejemplo en 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:

```
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:

```
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:

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

```
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:
```

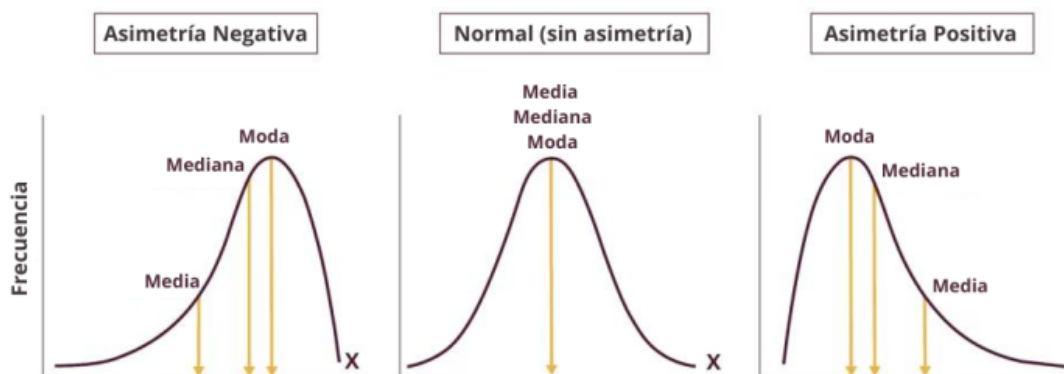
```
[1] 20
```

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):

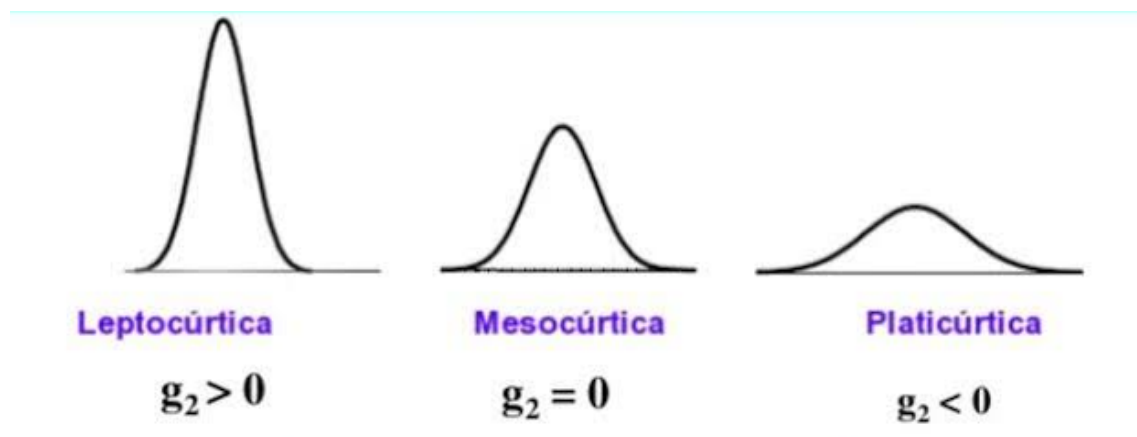
```
# Instalación y carga del paquete psych
if (!require("psych")) install.packages("psych")

# Análisis de asimetría
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):

```
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:


```
# 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

```
# 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.

```
# 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.

```
# 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 Asimetría
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

```

```

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.

```

# 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:

```
# Función para analizar 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.

```
# 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.

```
# 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 <code>psych</code>	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 <code>na.rm = TRUE</code> .	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.