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

P. Agr. Ludwing Isaí Marroquín Jiménez

6 may 2025

Tabla de contenidos

Introducción **9**
Propósito del manual . . . 9
Organización del manual . . . 9
Pre requisitos . . . 10
Software y convenciones . . . 11

I Introducción a R y RStudio **14**

1 Conceptos básicos de R **15**
1.1 ¿Qué es R? . . . 15
1.1.1 Características principales de R . . . 15
1.1.2 ¿Por qué es especial R? . . . 16
1.2 ¿Qué es RStudio? . . . 16
1.2.1 Características principales de RStudio . . . 17
1.2.2 Beneficios de usar RStudio . . . 17
1.3 Reproducibilidad y replicabilidad en la investigación científica . . . 17
1.3.1 El papel de R en la reproducibilidad . . . 18
1.3.2 Definición y características de la reproducibilidad . . . 18
1.3.3 Definición y características de la replicabilidad . . . 18
1.3.4 Beneficios de utilizar R para la ciencia reproducible . . . 19

2 Instalación y configuración **20**
2.1 Descarga de R y RStudio . . . 20
2.1.1 Descarga de R . . . 20
2.1.2 Descarga de RStudio . . . 20
2.2 Instalación de R y RStudio . . . 21
2.2.1 Instalación de R . . . 21
2.2.2 Instalación de RStudio . . . 21
2.3 Configuración inicial . . . 21
2.3.1 Seleccionar la versión de R . . . 21
2.3.2 Configurar la apariencia de RStudio . . . 22
2.3.3 Configurar el panel de trabajo . . . 22
2.3.4 Habilitar el número de líneas en el editor de scripts . . . 22
2.4 Organización de proyectos . . . 22
2.4.1 Crear un proyecto en RStudio . . . 23
2.4.2 Establecer un directorio de trabajo . . . 23
2.4.3 Uso de archivos .Rproj . . . 23
2.4.4 Beneficios de la organización de proyectos . . . 24

II Conceptos básicos de R **25**

3 Primeros pasos en R **26**
3.1 Creación de scripts en RStudio . . . 26
3.2 Guardado y organización de archivos . . . 26
3.2.1 Guardado de scripts y archivos . . . 26
3.2.2 Organización de directorios y proyectos . . . 27
3.2.3 Buenas prácticas para la organización de archivos . . . 27
3.3 Introducción a los objetos en R . . . 28
3.3.1 Creación de objetos en R . . . 28
3.3.2 Buenas prácticas y documentación . . . 28
3.4 Tipos principales de objetos en R . . . 28
3.4.1 Objetos Numéricos . . . 29
3.4.2 Objetos de Texto . . . 29
3.4.3 Objetos de Tipo Factor . . . 30
3.4.4 Objetos Lógicos . . . 31
3.5 Exploración de objetos: funciones útiles . . . 32

4 Estructuras de datos en R **34**
4.1 Vectores . . . 34
4.1.1 Tipos de Vectores y su creación . . . 34
4.1.2 Coerción de Tipos de Datos en Vectores . . . 35
4.1.3 Operaciones con Vectores . . . 35
4.2 Matrices . . . 38
4.2.1 Creación de Matrices y Argumentos de la Función matrix() . . . 38
4.2.2 Propiedades y Atributos de las Matrices . . . 40
4.2.3 Combinación de Matrices . . . 41
4.2.4 Operaciones y aplicaciones de las matrices . . . 42
4.3 Data frames . . . 43
4.3.1 Creación de data frames . . . 43
4.3.2 Ventajas de un data frame . . . 44
4.3.3 Manipulación de data frames . . . 44
4.4 Listas . . . 46
4.4.1 Creación de listas . . . 46
4.4.2 Acceso a elementos de una lista . . . 47
4.4.3 Aplicaciones prácticas . . . 48
4.5 Comparación entre Data Frames y Listas . . . 49

5 Importación de datos **50**
5.1 Configuración previa: el directorio de trabajo . . . 50
5.1.1 Automatización del directorio de trabajo en scripts independientes . . . 50
5.1.2 Verificación y buenas prácticas . . . 51
5.2 Importación de archivos CSV y Excel en R . . . 51
5.2.1 Importación de archivos CSV . . . 51
5.2.2 Importación de archivos Excel . . . 52
5.3 Ejemplo práctico: Base de datos de estudiantes USAC . . . 52
5.3.1 Importación de la base de datos . . . 53
5.3.2 Verificación de la importación de datos . . . 53
5.3.3 Consideraciones adicionales . . . 54

6 Operadores en R **55**
6.1 Operadores de Asignación 55
6.2 Operadores aritméticos 56
6.3 Operadores lógicos 56
6.3.1 Ejemplo práctico 57
6.4 Operadores de Manipulación de Datos 58

7 Funciones en R **60**
7.1 Definición y características de las funciones en R 60
7.1.1 Tipos de funciones 60
7.1.2 Funciones predefinidas en R 61
7.1.3 Diferencias entre funciones predefinidas y personalizadas 62
7.2 Usos y beneficios de las funciones en R 63
7.3 Cómo crear funciones en R: Sintaxis y ejemplos básicos 63
7.3.1 Sintaxis básica 63
7.3.2 Elementos clave de una función 64
7.3.3 Ejemplo básico 64

8 Paquetes en R **65**
8.1 ¿Qué es un paquete en R? 65
8.1.1 Características principales de los paquetes 65
8.1.2 Importancia del uso de paquetes en R 65
8.2 Instalación y carga de paquetes 66
8.2.1 **Proceso de instalación** 66
8.2.2 Carga de paquetes 66
8.2.3 Automatización de la instalación y carga 66
8.3 Paquetes recomendados para tareas específicas 67
8.3.1 Instalación y carga de paquetes esenciales 67
8.4 Ejemplo práctico: Integración de paquetes en un análisis estadístico 68
8.4.1 Contexto del estudio 68
8.4.2 Implementación del análisis 68

III Manipulación de datos **71**

9 Introducción a la manipulación de datos en R **72**
9.1 Principales tareas de manipulación de datos 72
9.1.1 Filtrado de datos 72
9.1.2 Selección de variables 72
9.1.3 Transformación de datos 73
9.1.4 Agregación de información 73
9.1.5 Reestructuración de datos 73
9.2 Enfoques disponibles en R para la manipulación de datos 73
9.2.1 Herramientas base de R 73
9.2.2 El enfoque tidyverse 74

10 Manipulación de Datos con Herramientas Base de R **75**
10.1 Datos de ejemplo 75

10.2 Selección y filtrado de datos en data frames . . . 76
10.2.1 Selección de columnas . . . 76
10.2.2 Filtrado de filas por condiciones lógicas . . . 77
10.3 Modificación de variables . . . 78
10.3.1 Creación de nuevas columnas . . . 78
10.3.2 Recodificación de variables categóricas . . . 79
10.4 Ordenamiento y agrupamiento de datos . . . 79
10.4.1 Ordenamiento de datos . . . 80
10.4.2 Cálculos estadísticos por grupo . . . 81
10.5 Manejo de valores faltantes y duplicados . . . 81
10.5.1 Identificación y manejo de valores faltantes . . . 81
10.5.2 Manejo de duplicados . . . 82
10.5.3 Ejemplo práctico: Preparación de datos para análisis de varianza (ANOVA) . . . 82

11 Manipulación de datos con dplyr y tidyr 84
11.1 Introducción a los paquetes dplyr y tidyr . . . 84
11.2 Datos ejemplo . . . 84
11.3 Operaciones básicas con dplyr . . . 85
11.3.1 Filtrado de datos con filter() . . . 85
11.3.2 Selección de columnas con select() . . . 86
11.3.3 Creación y transformación de variables con mutate() . . . 87
11.3.4 Agrupamiento y resumen con group_by() y summarize() . . . 87
11.3.5 Ordenamiento de datos con arrange() . . . 88
11.4 Introducción a los pipes (%>%) . . . 89
11.4.1 Ventajas del uso de pipes . . . 89
11.4.2 Ejemplo práctico . . . 90
11.5 Transformaciones de datos con tidyr . . . 90
11.5.1 Transformación de formato ancho a largo con pivot_longer() . . . 91
11.5.2 Transformación de formato largo a ancho con pivot_wider() . . . 92
11.5.3 Manipulación de variables compuestas con separate() y unite() . . . 93
11.6 Ejemplo integrado: Preparación de datos para análisis estadístico clásico . . . 94
11.7 Comparación entre la manipulación de datos con R base y tidyverse . . . 96

IV Visualización de datos 98

12 Introducción a la visualización de datos 99
12.1 Historia y evolución de la visualización en estadística . . . 99
12.2 Principios básicos de la visualización efectiva . . . 99
12.2.1 Claridad . . . 100
12.2.2 Precisión . . . 100
12.2.3 Eficiencia . . . 101
12.2.4 Errores comunes a evitar en la visualización de datos . . . 101
12.2.5 Recomendaciones para una visualización efectiva . . . 101
12.3 Tipos de gráficos y su utilidad en estadística clásica . . . 102
12.3.1 Gráficos de barras . . . 102
12.3.2 Histogramas . . . 103
12.3.3 Diagramas de caja (boxplots) . . . 103

12.3.4 Gráficos de dispersión (scatterplots) . . . 104
12.3.5 Gráficos QQ (quantile-quantile) . . . 105
12.3.6 Gráficos de residuos . . . 105

13 Visualizaciones Base de R **107**
13.1 Funciones gráficas básicas de R . . . 107
13.2 Creación de gráficos exploratorios . . . 108
13.2.1 Histogramas . . . 108
13.2.2 Diagramas de caja (boxplots) . . . 109
13.2.3 Gráficos de dispersión . . . 110
13.2.4 Gráficos de líneas . . . 112
13.3 Visualización para la comprobación de supuestos estadísticos . . . 113
13.3.1 Gráficos Q-Q para evaluar normalidad . . . 113
13.3.2 Gráficos de diagnóstico para modelos de regresión . . . 115
13.4 Personalización de gráficos en R base . . . 116
13.4.1 **Principales argumentos y funciones:** . . . 116
13.4.2 Ejemplo integral . . . 117

14 Visualización de datos con ggplot2 **119**
14.1 Contexto y origen de la base de datos utilizada . . . 119
14.2 Introducción al paquete ggplot2 . . . 120
14.2.1 Ventajas principales de ggplot2 . . . 120
14.2.2 ¿Cómo funciona la gramática de los gráficos? . . . 121
14.3 Estructura básica de un gráfico en ggplot2 . . . 121
14.4 Creación de gráficos exploratorios y descriptivos . . . 122
14.4.1 Gráficos de barras para variables categóricas . . . 122
14.4.2 Histogramas para variables continuas . . . 123
14.4.3 Gráficos de dispersión para relaciones entre variables numéricas . . . 124
14.4.4 Boxplots para comparación de grupos . . . 125
14.5 Personalización de gráficos en ggplot2 . . . 126
14.5.1 Modificación de colores y escalas . . . 127
14.5.2 Etiquetas, títulos y leyendas . . . 128
14.5.3 Aplicación y personalización de temas . . . 129
14.6 Uso de facetas para comparación de grupos . . . 131
14.6.1 Facet_wrap y facet_grid: sintaxis y aplicaciones . . . 131
14.6.2 **Cuadro resumen de diferencias de las funciones face_wrap y face_grid** . . . 133
14.7 Comparación entre ggplot2 y el sistema gráfico base de R . . . 133

V Gestión y Exportación de resultados **135**

15 Introducción a la Gestión de Proyectos en R **136**
15.1 Importancia de la gestión de proyectos en análisis estadístico . . . 136
15.2 Organización de archivos en proyectos de R . . . 136
15.3 Uso de RStudio Projects para la gestión eficiente . . . 137
15.4 Principios de reproducibilidad y documentación en proyectos de R . . . 137

16 Exportación de Resultados de Análisis en R **139**
16.1 Exportación de gráficos: formatos PNG y PDF . . . 139
16.1.1 Sintaxis general de ggsave() . . . 139
16.1.2 Ejemplo práctico: creación y exportación de un gráfico . . . 140
16.2 Exportación de tablas de datos: formatos CSV y Excel . . . 142
16.2.1 Exportar a CSV con write.csv() . . . 142
16.2.2 Exportar a Excel con write_xlsx() del paquete writexl . . . 143
16.3 Comparación de formatos y recomendaciones de uso . . . 144

17 Uso de Git y GitHub en Proyectos de R **146**
17.1 Introducción al control de versiones y colaboración . . . 146
17.2 Subida de un proyecto de R a GitHub . . . 147
17.3 Modificación y seguimiento de proyectos en GitHub . . . 148
17.4 Importación de repositorios de GitHub para trabajo local o personal . . . 149

VI Referencias **151**

18 Referencias **152**

VII Ejemplos de Análisis Estadístico con R **154**

19 Estadística descriptiva usando funciones en R **155**
19.1 Medidas principales en estadística descriptiva . . . 155
19.1.1 Medidas de tendencia central . . . 155
19.1.2 Medidas de dispersión . . . 156
19.1.3 Medidas de forma . . . 158
19.2 Base de datos para los ejemplos . . . 160
19.2.1 Preparación del área de trabajo . . . 160
19.2.2 Establecer directorio de trabajo . . . 160
19.2.3 Importar la base de datos . . . 160
19.2.4 Revisar de la estructura de los datos . . . 161
19.2.5 Limpieza de la base de datos . . . 161
19.3 Funciones por defecto en R para estadística descriptiva . . . 162
19.3.1 Medidas de tendencia central . . . 162
19.3.2 Medidas de dispersión . . . 162
19.3.3 Medidas de forma . . . 163
19.3.4 Resumen general con `summary()` . . . 163
19.4 Paquetes especializados para estadística descriptiva (el paquete `psych`) . . . 164
19.4.1 Instalación y carga del paquete . . . 164
19.4.2 Análisis descriptivo general . . . 164
19.4.3 Análisis descriptivo categorizado . . . 165
19.4.4 Exportación de resultados . . . 166
19.4.5 Ventajas del paquete `psych` . . . 166
19.4.6 Limitaciones del paquete `psych` . . . 167
19.5 Función personalizada para análisis descriptivo completo . . . 167
19.5.1 Establecer funciones auxiliares . . . 167
19.5.2 Función principal: análisis por categorías . . . 169

19.5.3 Función para análisis de múltiples variables numéricas 169
19.5.4 Ejemplo de uso 170
19.6 Resumen Comparativo: Funciones Base de R, Paquete `psych` y Función Personalizada 172

20 Regresión lineal simple usando R **174**
20.1 Conceptos fundamentales 174
20.2 Modelo de Regresión Lineal Simple 174
20.3 Supuestos de la Regresión Lineal 175
20.4 Evaluación del Modelo 175
20.5 Ejemplo Práctico 175
20.5.1 Instalación y Carga de Paquetes 176
20.5.2 Importación de Datos 176
20.5.3 Visualización de Datos 177
20.5.4 Ajuste del Modelo de Regresión 177
20.5.5 Gráficos de Diagnóstico para evaluar los supuestos del modelo 178
20.5.6 Prueba de Normalidad de los Residuos 179
20.5.7 Prueba de Homocedasticidad de la varianza 179

21 Regresión múltiple usando R **181**
21.1 Notas: 181

Introducción

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

Propósito del manual

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

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

Organización del manual

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

El manual se organiza en las siguientes secciones:

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

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

Pre requisitos

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

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

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

Software y convenciones

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

Durante la elaboración del manual se emplearon diversos paquetes del ecosistema de R, entre los que destacan knitr y bookdown, los cuales permiten combinar las ventajas de LaTeX y R para la generación de documentos dinámicos y reproducibles (Xie, 2015; Xie, 2024). Esta integración posibilita que los ejemplos de código y los resultados presentados sean fácilmente replicables por el lector.

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

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

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

```
[1] 10
```

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

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

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

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

```
Warning in system2("quarto", "-V", stdout = TRUE, env = paste0("TMPDIR=", : el
comando ejecutado '"quarto"
TMPDIR=C:/Users/Usuario/AppData/Local/Temp/Rtmp2Rm5fP/file37785d047f7e -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     2025-04-30
 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 especializado en el análisis estadístico, la visualización de datos y la investigación científica. Su desarrollo fue iniciado en 1996 por Ross Ihaka y Robert Gentleman, quienes lo concibieron como una herramienta flexible y robusta para realizar análisis reproducibles y generar visualizaciones de alta calidad (Ihaka & Gentleman, 1996). Desde su creación, R ha evolucionado hasta convertirse en una de las plataformas más utilizadas en los ámbitos científico, académico y profesional, gracias a su capacidad de adaptación, su naturaleza de código abierto y el respaldo de una comunidad global activa.

1.1.1 Características principales de R

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

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

Otra característica fundamental de R es su contribución a la reproducibilidad científica. El uso de scripts y cuadernos de trabajo permite documentar cada paso del análisis, facilitando la replicación y verificación de resultados por parte de otros investigadores (Peng, 2011). Además, R es altamente interoperable, permitiendo la integración con otros lenguajes de programación como Python, C++ y SQL, así como la importación y exportación de datos en múltiples formatos, incluyendo CSV, Excel, JSON y bases de datos relacionales (R Core Team, 2023).

1.1.2 ¿Por qué es especial R?

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

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

1.2 ¿Qué es RStudio?

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

1.2.1 Características principales de RStudio

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

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

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

1.2.2 Beneficios de usar RStudio

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

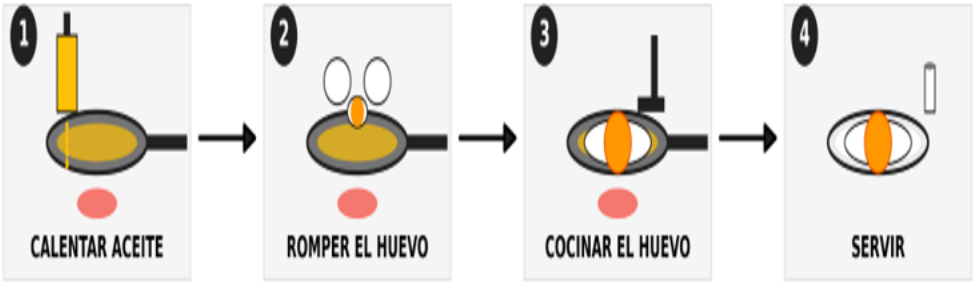
1.3 Reproducibilidad y replicabilidad en la investigación científica

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

modificaciones manuales, lo que dificulta la reproducción exacta de los resultados y la verificación independiente de los análisis.

1.3.1 El papel de R en la reproducibilidad

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



1.3.2 Definición y características de la reproducibilidad

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

1.3.3 Definición y características de la replicabilidad

Por otro lado, la replicabilidad se refiere a la obtención de resultados consistentes al realizar un estudio similar en un contexto diferente, utilizando nuevos datos o métodos ajustados. Este concepto evalúa la capacidad de generalización de los hallazgos y su aplicabilidad en distintos escenarios, lo que resulta fundamental para validar la robustez de los resultados

científicos (National Academies of Sciences, Engineering, and Medicine, 2019). La replicabilidad implica el uso de datos diferentes, la adaptación de los métodos y la obtención de resultados coherentes con los del estudio original, aunque no necesariamente idénticos. Este proceso permite evaluar la generalización de los resultados, refuerza la credibilidad científica y facilita la extensión del conocimiento a nuevas aplicaciones o contextos (The Turing Way Community, 2023).

1.3.4 Beneficios de utilizar R para la ciencia reproducible

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

2 Instalación y configuración

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

2.1 Descarga de R y RStudio

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

2.1.1 Descarga de R

Se recomienda descargar una versión estable de R para evitar posibles incompatibilidades con paquetes que aún no han sido actualizados para las versiones más recientes. Por ejemplo, la versión R 4.4.2 es reconocida por su estabilidad y amplio soporte dentro de la comunidad de usuarios (R Core Team, 2023). El repositorio oficial de R se encuentra disponible en https://cran.r-project.org/bin/windows/base/old/, donde es posible acceder a todas las versiones publicadas. Para descargar una versión específica, se debe seleccionar el nombre de la versión deseada y hacer clic en el archivo con terminación -win.exe, lo que iniciará la descarga del instalador correspondiente (R Core Team, 2023).

2.1.2 Descarga de RStudio

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

2.2 Instalación de R y RStudio

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

2.2.1 Instalación de R

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

2.2.2 Instalación de RStudio

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

2.3 Configuración inicial

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

2.3.1 Seleccionar la versión de R

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

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

2.3.2 Configurar la apariencia de RStudio

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

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

2.3.3 Configurar el panel de trabajo

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

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

2.3.4 Habilitar el número de líneas en el editor de scripts

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

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

2.4 Organización de proyectos

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

2.4.1 Crear un proyecto en RStudio

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

1. En la barra de menú, seleccionar **File > New Project**.
2. Elegir una de las siguientes opciones:
    1. **New Directory**: para crear un proyecto desde cero en una nueva carpeta.
    2. **Existing Directory**: para convertir una carpeta existente en un proyecto de RStudio.
    3. **Version Control**: para clonar un repositorio de Git y trabajar en un proyecto con control de versiones.
3. Configurar el nombre y la ubicación del proyecto según las necesidades del análisis.
4. Hacer clic en **Create Project** para finalizar la configuración.

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

2.4.2 Establecer un directorio de trabajo

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

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

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

2.4.3 Uso de archivos .Rproj

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

2.4.4 Beneficios de la organización de proyectos

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

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

Parte II

Conceptos básicos de R

3 Primeros pasos en R

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

3.1 Creación de scripts en RStudio

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

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

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

3.2 Guardado y organización de archivos

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

3.2.1 Guardado de scripts y archivos

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

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

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

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

3.2.2 Organización de directorios y proyectos

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

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

3.2.3 Buenas prácticas para la organización de archivos

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

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

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

3.3 Introducción a los objetos en R

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

3.3.1 Creación de objetos en R

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

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

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

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

3.3.2 Buenas prácticas y documentación

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

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

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

3.4 Tipos principales de objetos en R

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

3.4.1 Objetos Numéricos

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

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

Ejemplo de creación de objetos numéricos:

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

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

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

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

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

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

```
 num 145
```

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

```
[1] 22.75833
```

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

3.4.2 Objetos de Texto

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

Ejemplo de creación de objetos de texto:

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

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

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

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

```
 chr "Juan García"
```

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

```
[1] 4
```

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

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

3.4.3 Objetos de Tipo Factor

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

Ejemplo de creación de objetos tipo factor:

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

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

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

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

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

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

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

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

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

```
[1] TRUE
```

Los factores son especialmente útiles en análisis estadísticos porque:

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

3.4.4 Objetos Lógicos

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

Ejemplo de creación de objetos lógicos:

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


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

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

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

```
 logi TRUE
```

Los objetos lógicos son esenciales para:

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

3.5 Exploración de objetos: funciones útiles

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

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

Ejemplo de uso de estas funciones:

```
# Exploración de un objeto numérico
class(x)      # Devuelve "numeric"
```

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

```
typeof(x)     # Devuelve "double"
```

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

```
str(x)        # Muestra la estructura del objeto
```

```
 num 10
```

```
# Exploración de un objeto de texto
class(nombre) # Devuelve "character"
```

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

```
str(nombre)   # Muestra la estructura del objeto
```

```
 chr "Juan"
```

```r
# Exploración de un factor
estado_civil <- factor("soltero", levels = c("soltero", "casado", "divorciado"))
class(estado_civil)   # Devuelve "factor"
```

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

```r
typeof(estado_civil)  # Devuelve "integer"
```

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

```r
str(estado_civil)     # Muestra la estructura del factor
```

```
 Factor w/ 3 levels "soltero","casado",..: 1
```

```r
levels(estado_civil)  # Devuelve los niveles posibles
```

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

4 Estructuras de datos en R

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

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

4.1 Vectores

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

4.1.1 Tipos de Vectores y su creación

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

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

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

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

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

4.1.2 Coerción de Tipos de Datos en Vectores

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

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

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

4.1.3 Operaciones con Vectores

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

4.1.3.1 Acceso a elementos específicos

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

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

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

4.1.3.2 Filtrado de elementos

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

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

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

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

4.1.3.3 Combinación de vectores

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

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

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

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

4.1.3.4 Funciones Útiles para Vectores

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

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

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

4.1.3.5 Aplicaciones Prácticas

Los vectores son esenciales en análisis estadísticos básicos y exploratorios:

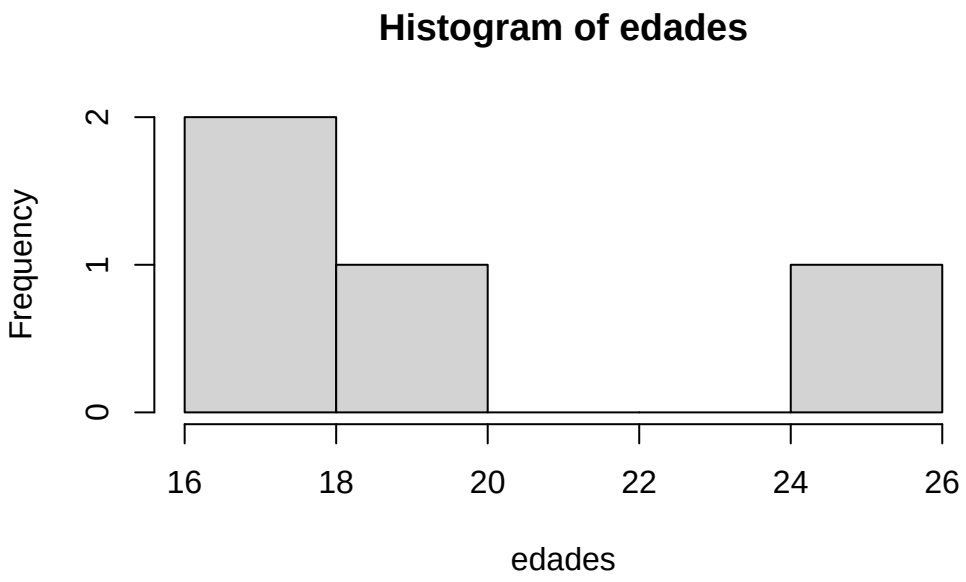
```r
# Análisis descriptivo
summary(edades)  # Resumen estadístico
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  17.00   17.75   19.00   20.00   21.25   25.00 
```

```r
table(mayores_de_edad)  # Tabla de frecuencias
```

```
mayores_de_edad
FALSE  TRUE 
    1     3 
```

```r
hist(edades)  # Histograma de edades
```



La utilización adecuada de los vectores en R resulta indispensable para la gestión eficiente de datos y la ejecución de análisis estadísticos rigurosos. El dominio de esta estructura permite no solo realizar procedimientos exploratorios y descriptivos, sino también sienta las bases metodológicas para la implementación de técnicas analíticas avanzadas y el desarrollo de modelos estadísticos complejos en entornos de investigación y aplicación profesional (Grolemund & Wickham, 2017; Tukey, 1977).

4.2 Matrices

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.2.2 Propiedades y Atributos de las Matrices

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

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

```
[1] 2 3
```

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

```
[1] 2
```

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

```
[1] 3
```

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

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

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

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

```
[1] 2
```

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

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

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

```
Fila1 Fila2
    3     4
```

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

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

4.2.3 Combinación de Matrices

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

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

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

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

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

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

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

4.2.4 Operaciones y aplicaciones de las matrices

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

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

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

# Multiplicación matricial
producto_matricial <- matriz_C %*% matriz_D

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

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

4.3 Data frames

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

Una característica distintiva de los data frames es la posibilidad de que cada columna almacene un tipo de dato diferente, como valores numéricos, cadenas de texto, valores lógicos o factores. Esta flexibilidad resulta fundamental para el manejo de datos heterogéneos, permitiendo la integración y el análisis eficiente de información proveniente de encuestas, experimentos científicos, registros administrativos y otros contextos donde la diversidad de variables es común. Además, los data frames son ampliamente compatibles con funciones y paquetes especializados en R, como `ggplot2` y `dplyr`, lo que los convierte en la estructura de datos más utilizada en el análisis estadístico y científico con este lenguaje (Wickham & Grolemund, 2017; R Core Team, 2023).

4.3.1 Creación de data frames

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

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

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

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

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

```r
# 3. Visualizar el data frame
datos
```

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

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

4.3.2 Ventajas de un data frame

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

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

4.3.3 Manipulación de data frames

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

4.3.3.1 Acceso a columnas

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

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

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

4.3.3.2 Filtrado de filas

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

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

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

4.3.3.3 Agregar nuevas columnas

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

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

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

4.3.3.4 Seleccionar varias columnas

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

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

4.3.3.5 Resumir información

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

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

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

4.4 Listas

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

4.4.1 Creación de listas

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

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

Por ejemplo:

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

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

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

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

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

4.4.2 Acceso a elementos de una lista

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

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

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

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

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

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

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

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

```r
# Acceder al primer elemento de la lista (en este caso, el vector de nombres)
mi_lista[[1]]
```

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

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

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

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

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

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

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

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

4.4.3 Aplicaciones prácticas

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

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

4.5 Comparación entre Data Frames y Listas

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

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

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

5 Importación de datos

La importación de datos es un paso fundamental en cualquier análisis estadístico, ya que permite trabajar con información proveniente de diversas fuentes, como archivos CSV, Excel o páginas web. R proporciona funciones y paquetes especializados para importar datos de manera eficiente y reproducible, lo que facilita el manejo de grandes volúmenes de información (R Core Team, 2023).

5.1 Configuración previa: el directorio de trabajo

Antes de importar datos, es esencial asegurarse de que el directorio de trabajo esté correctamente configurado. El directorio de trabajo, o *working directory*, es la carpeta desde la cual R buscará archivos y guardará los resultados generados. Una configuración adecuada del directorio de trabajo contribuye a la organización, reproducibilidad y eficiencia del análisis.

Cuando se utiliza un proyecto de RStudio (**.Rproj**), el directorio de trabajo se establece automáticamente al abrir el proyecto, lo que simplifica la gestión de archivos. Sin embargo, si se trabaja con scripts independientes, es necesario definir manualmente el directorio utilizando la función `setwd()`. Por ejemplo:

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

Configurar el directorio de trabajo correctamente evita errores como "archivo no encontrado" y asegura que el código sea portable y replicable en diferentes computadoras o ubicaciones.

5.1.1 Automatización del directorio de trabajo en scripts independientes

Para scripts que no forman parte de un proyecto, se puede automatizar la configuración del directorio de trabajo utilizando el paquete `rstudioapi`. Este método establece como directorio de trabajo la carpeta donde está guardado el script, facilitando la portabilidad y colaboración:

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

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

Este código verifica si el paquete `rstudioapi` está instalado y, de no ser así, lo instala automáticamente. Luego, obtiene la ruta del script actual y la utiliza para definir el directorio de trabajo, permitiendo acceder a los archivos de la carpeta sin necesidad de rutas completas.

5.1.2 Verificación y buenas prácticas

Antes de importar datos o guardar resultados, es recomendable verificar el directorio de trabajo actual con la función `getwd()`:

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

Además, se sugiere guardar el script antes de ejecutar la configuración automática del directorio, ya que R necesita conocer la ubicación del archivo para establecer correctamente el entorno de trabajo.

Siempre que sea posible, se recomienda trabajar dentro de un proyecto de RStudio, ya que esto automatiza la gestión del directorio de trabajo y mejora la organización de los archivos relacionados con el análisis.

5.2 Importación de archivos CSV y Excel en R

La importación de datos tabulares es una tarea fundamental en el análisis estadístico. R facilita este proceso mediante funciones y paquetes que permiten trabajar con archivos en formatos ampliamente utilizados, como CSV y Excel. Comprender cómo importar estos archivos correctamente es esencial para garantizar la integridad y reproducibilidad del análisis (Wickham, 2016; Wickham & Bryan, 2023).

5.2.1 Importación de archivos CSV

Los archivos CSV (Comma-Separated Values) son ampliamente utilizados debido a su simplicidad y compatibilidad con diferentes plataformas. Para importar un archivo CSV en R, se emplea la función `read.csv()`, que permite leer datos tabulares de manera eficiente. Un ejemplo básico de uso es el siguiente:

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

Los parámetros principales de esta función son:

1. **header:** Indica si la primera fila del archivo contiene los nombres de las columnas (`TRUE`) o si los datos comienzan desde la primera fila (`FALSE`).

2. **sep:** Especifica el carácter separador de los valores. Por defecto, es la coma (`,`), pero puede ajustarse a otros separadores como punto y coma (`;`) o tabulación (`\t`) según el formato del archivo.

5.2.2 Importación de archivos Excel

Para trabajar con archivos de Excel (.xlsx), R dispone del paquete `readxl`, que permite importar datos directamente desde hojas de cálculo sin necesidad de convertir previamente el archivo a otro formato (Wickham & Bryan, 2023). El proceso recomendado es el siguiente:

1. Instalar y cargar el paquete `readxl`.

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

2. Importar el archivo Excel utilizando la función `read_excel()`:

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

Los parámetros más relevantes son:

1. **sheet**: Permite especificar el nombre o el número de la hoja que se desea importar.
2. **col_names**: Indica si la primera fila debe ser utilizada como nombres de las columnas (`TRUE`) o si los datos comienzan desde la primera fila (`FALSE`).

5.3 Ejemplo práctico: Base de datos de estudiantes USAC

Para consolidar los conceptos de importación de datos, se utilizará como ejemplo una base de datos recopilada en 2002 en la Universidad de San Carlos de Guatemala. Esta base contiene información de 460 estudiantes de distintas facultades y está disponible tanto en formato CSV como Excel. A lo largo del manual, esta base de datos servirá como referencia para los ejercicios y ejemplos prácticos.

5.3.1 Importación de la base de datos

A continuación, se muestra cómo importar la base de datos en ambos formatos:

1. Importar archivo CSV:

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

2. Importar archivo Excel: Primero, el usuario se debe asegurar de tener instalado y cargado el paquete `readxl`.

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


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

5.3.2 Verificación de la importación de datos

Una vez importados los datos, es fundamental comprobar que la importación se realizó correctamente. R ofrece varias funciones útiles para este propósito:

1. Visualizar las primeras filas del conjunto de datos:

```
# Visualizar las primeras filas de la base de datos
head(datos_csv)
```

2. Examinar la estructura del data frame:

```
# Examinar la estructura de la base de datos
str(datos_csv)
```

3. Obtener un resumen estadístico básico:

```
# Resumen estadístico básico de la base de datos
summary(datos_csv)
```

4. Conocer las dimensiones del data frame (número de filas y columnas):

```
# Conocer las dimensiones del data frame
dim(datos_csv)
```

5.3.3 Consideraciones adicionales

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:

```r
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.

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

6.1 Operadores de Asignación

En R, los operadores de asignación permiten crear objetos y almacenar valores en ellos, lo que constituye una de las bases para la manipulación de datos y la programación en este lenguaje. Los dos operadores de asignación más utilizados son `<-` y `=`. Ambos operadores cumplen la función de asignar un valor a un objeto, sin embargo, la convención en la comunidad de R es emplear `<-`, ya que este evita posibles ambigüedades con el operador lógico de igualdad (`==`) y mantiene la claridad en el código (Ihaka & Gentleman, 1996).

Por ejemplo, al ejecutar la instrucción `x <- 10`, se asigna el valor 10 al objeto `x`. De manera similar, `y = 20` asigna el valor 20 al objeto `y`, aunque esta forma es menos recomendada en contextos profesionales. Posteriormente, es posible utilizar estos objetos en operaciones, como se muestra a continuación:

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

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

```
[1] 30
```

Es importante destacar que, aunque el operador = puede emplearse para asignar valores, su uso puede generar confusiones, especialmente cuando se trabaja con expresiones lógicas, ya que = también se utiliza en otros contextos dentro del lenguaje. Por esta razón, se recomienda preferir el uso de <- para la asignación de valores en la mayoría de los casos (Ihaka & Gentleman, 1996).

6.2 Operadores aritméticos

Los operadores aritméticos permiten realizar operaciones matemáticas básicas y avanzadas. Son fundamentales para trabajar con datos numéricos y realizar cálculos en análisis estadísticos. Estos operadores operan sobre valores numéricos y devuelven resultados numéricos. 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 (R Core Team, 2023).

A continuación se presenta un cuadro con los principales operadores aritméticos disponibles en R y las operaciones que ejecutan:

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 Operadores lógicos

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

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

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

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

6.3.1 Ejemplo práctico

A continuación, se muestra un ejemplo práctico, donde se emplean operadores lógicos para realizar comparaciones y evaluaciones condicionales:

```
# Comparaciones simples
edad <- 25
es_mayor <- edad > 18          # TRUE, porque 25 es mayor que 18
es_menor <- edad < 30          # TRUE, porque 25 es menor que 30
es_igual <- edad == 25         # TRUE, porque 25 es igual a 25
es_diferente <- edad != 20     # TRUE, porque 25 es diferente de 20

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

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

```
[1] FALSE
```

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

```
[1] TRUE
```

En este ejemplo, se observa cómo los operadores lógicos permiten evaluar condiciones tanto simples como compuestas, facilitando la clasificación de datos y la toma de decisiones dentro del análisis estadístico en R (R Core Team, 2023).

6.4 Operadores de Manipulación de Datos

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

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

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

A continuación, se presenta un ejemplo práctico utilizando fragmentos de código en R para ilustrar el uso de estos operadores:

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

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

```
[1] 10
```

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

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

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

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

```
[1] 65
```

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

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

Este ejemplo muestra cómo los operadores de manipulación de datos permiten seleccionar elementos individuales, columnas completas o secuencias de valores dentro de las estructuras de datos más utilizadas en R. Estas operaciones son fundamentales para filtrar, transformar y analizar información de manera precisa y eficiente en el entorno estadístico (R Core Team, 2023).

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 & Grolemund, 2017).

Esta sección aborda el concepto de función en R, su utilización y el proceso para crear funciones personalizadas que resuelvan necesidades particulares.

7.1 Definición y características de las funciones en R

Una función en R se define como un objeto capaz de recibir uno o más valores de entrada, denominados argumentos, ejecutar una serie de operaciones sobre ellos y devolver un resultado. La utilización de funciones permite estructurar el código de manera eficiente y reutilizable, lo que es fundamental para el desarrollo de proyectos complejos (Chambers, 2008).

Toda función en R se compone de tres elementos principales:

1. El **nombre**, que sirve como identificador para invocar la función.
2. Los **argumentos**, que corresponden a los valores de entrada requeridos para ejecutar las operaciones internas.
3. El **cuerpo**, que contiene el conjunto de instrucciones que determinan el comportamiento de la función.

7.1.1 Tipos de funciones

En R, las funciones se dividen en dos categorías principales:

1. Las **funciones predefinidas,** también conocidas como **built-in**, están incluidas en el propio lenguaje o en paquetes adicionales. Estas funciones cubren una amplia gama de tareas, como cálculos matemáticos, operaciones estadísticas, manipulación de datos y generación de gráficos.
2. Las **funciones personalizadas,** o **user-defined**, son creadas por el usuario para abordar necesidades específicas que no se resuelven con las funciones predefinidas. Este tipo de funciones resulta especialmente útil para automatizar procesos repetitivos o implementar cálculos complejos adaptados a un contexto particular.

El dominio de la creación y el uso de funciones, tanto predefinidas como personalizadas, es clave para desarrollar análisis eficientes y escalables en R (R Core Team, 2023; Wickham & Grolemund, 2017; Chambers, 2008).

7.1.2 Funciones predefinidas en R

Las funciones predefinidas en R constituyen herramientas fundamentales para la realización de operaciones habituales de manera eficiente y directa. Estas funciones, incluidas en el núcleo del lenguaje, permiten efectuar cálculos estadísticos, manipular datos y obtener resúmenes sin necesidad de definir procedimientos adicionales. Su uso facilita la resolución de tareas comunes y contribuye a la claridad del código (R Core Team, 2023).

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

```
# Definición de un vector de datos
datos <- c(1, 2, 3, 4, 5)

# Cálculo de la media aritmética
mean(datos)           # Resultado: 3
```

```
[1] 3
```

```
# Suma de los elementos del vector
sum(datos)             # Resultado: 15
```

```
[1] 15
```

```
# Cálculo de la desviación estándar
sd(datos)        # Resultado: 1.581139
```

```
[1] 1.581139
```

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

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

Estas funciones están disponibles de forma predeterminada en R y permiten realizar análisis estadísticos básicos de manera inmediata, sin requerir definiciones adicionales por parte del usuario. Su dominio es esencial para el trabajo cotidiano con datos en R (R Core Team, 2023).

7.1.2.1 Funciones personalizadas en R

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

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

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

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

En este ejemplo, el usuario define la lógica de la función, especifica el argumento necesario (radio) y utiliza la función para obtener el resultado deseado.

7.1.3 Diferencias entre funciones predefinidas y personalizadas

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

Característica	Funciones predefinidas	Funciones personalizadas
Disponibilidad	Incluidas en R o en paquetes	Creadas por el usuario
Flexibilidad	Limitada a las tareas para las que fueron diseñadas	Totalmente adaptables a las necesidades del usuario
Ejemplos	`mean()`, `sum()`, `sd()`, `summary()`	`calcular_area_circulo()`
Reutilización	Reutilizables en cualquier script	Reutilizables si se definen en el entorno o importan

Esta distinción permite seleccionar el tipo de función más adecuado según el contexto y los objetivos del análisis (R Core Team, 2023).

7.2 Usos y beneficios de las funciones en R

El empleo de funciones en R aporta ventajas significativas que optimizan el desarrollo y la gestión de proyectos de análisis de datos. La reutilización de código es uno de los principales beneficios, ya que una función definida puede emplearse en distintas partes de un mismo proyecto o en proyectos diferentes, lo que reduce la duplicación y facilita el mantenimiento. Además, las funciones promueven la modularidad, permitiendo descomponer problemas complejos en componentes más simples y manejables, lo que mejora la organización y la estructura del código.

Otro aspecto relevante es la legibilidad, ya que encapsular operaciones dentro de funciones contribuye a que el código sea más claro y comprensible, facilitando su revisión y colaboración entre usuarios. Asimismo, las funciones permiten automatizar tareas repetitivas, lo que incrementa la eficiencia y ahorra tiempo en la ejecución de procesos rutinarios (R Core Team, 2023; Wickham & Grolemund, 2017).

A continuación, se presenta un ejemplo de automatización mediante una función personalizada. Si se requiere calcular el área de varios círculos con diferentes radios, basta con aplicar la función previamente definida a un vector de radios, evitando así la repetición manual del cálculo:

```
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
```

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

7.3 Cómo crear funciones en R: Sintaxis y ejemplos básicos

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

7.3.1 Sintaxis básica

La sintaxis general para crear una función en R es la siguiente:

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

7.3.2 Elementos clave de una función

Cada función en R se compone de los siguientes elementos:

1. **Nombre de la función**: que debe ser descriptivo y reflejar claramente la tarea que realiza.
2. **Argumentos**: representan los valores de entrada requeridos por la función. Es posible asignar valores por defecto a estos argumentos para hacer la función más flexible.
3. **Cuerpo de la función**: donde se incluyen las operaciones y cálculos que se ejecutan al llamar a la función.
4. **Valor de retorno**: que se especifica mediante la instrucción return(). Si no se utiliza `return()`, la función devolverá automáticamente el último valor evaluado en su cuerpo.

7.3.3 Ejemplo básico

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

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

# Uso de la función
celsius_a_fahrenheit(25)   # Resultado: 77
```

```
[1] 77
```

8 Paquetes en R

Los paquetes en R constituyen una de las herramientas más potentes del lenguaje, ya que permiten ampliar sus funcionalidades básicas y abordar tareas especializadas de manera eficiente. Gracias a los paquetes, es posible acceder a funciones, datos y documentación desarrollados por expertos, lo que facilita la realización de análisis estadísticos, manipulación de datos y visualización avanzada, entre otras aplicaciones (R Core Team, 2023).

8.1 ¿Qué es un paquete en R?

Un paquete en R se define como una colección estructurada de funciones, conjuntos de datos y documentación que extiende las capacidades del entorno base. Estos paquetes, desarrollados tanto por la comunidad como por equipos especializados, están orientados a resolver problemas concretos en diversas áreas, desde la manipulación de datos hasta el análisis estadístico avanzado o la generación de gráficos complejos (Wickham & Bryan, 2023).

8.1.1 Características principales de los paquetes

Las características más relevantes de los paquetes en R se resumen a continuación:

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 Importancia del uso de paquetes en R

El uso de paquetes resulta fundamental para aprovechar plenamente el potencial de R, ya que ofrecen extensibilidad, eficiencia y especialización. Los paquetes permiten realizar tareas que no están disponibles en el entorno base, simplifican procesos complejos y proporcionan soluciones adaptadas a áreas específicas, como la agronomía, la biología o la economía. Además, la comunidad activa de R garantiza la actualización y el soporte continuo de una amplia variedad de paquetes, lo que contribuye a mantener el lenguaje a la vanguardia en el análisis de datos (Wickham & Grolemund, 2017).

8.2 Instalación y carga de paquetes

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

8.2.1 Proceso de instalación

Para instalar un paquete desde CRAN, se utiliza la función `install.packages()`. Por ejemplo, para instalar el paquete `ggplot2`, ampliamente utilizado para la visualización de datos, se emplea la siguiente instrucción:

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

La instalación de un paquete es un proceso que solo debe realizarse una vez en el sistema.

8.2.2 Carga de paquetes

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

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

La carga de paquetes debe repetirse cada vez que se inicia una nueva sesión en R, ya que los paquetes no se cargan automáticamente al abrir el entorno.

8.2.3 Automatización de la instalación y carga

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

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

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

8.3 Paquetes recomendados para tareas específicas

En el ámbito del análisis estadístico y la manipulación de datos, existen diversos paquetes que facilitan tareas específicas y permiten realizar análisis complejos de manera eficiente. A continuación, se presenta una clasificación de los paquetes más relevantes según su área de aplicación:

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

8.3.1 Instalación y carga de paquetes esenciales

El siguiente código muestra cómo instalar y cargar un conjunto básico de paquetes para análisis estadísticos:

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

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

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

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

Estos paquetes proporcionan un conjunto robusto de herramientas para realizar análisis estadísticos completos, desde la exploración inicial de datos hasta análisis especializados en áreas específicas como la agronomía o la psicometría (R Core Team, 2023).

8.4 Ejemplo práctico: Integración de paquetes en un análisis estadístico

Este ejemplo ilustra la aplicación práctica de diversos paquetes de R en un análisis estadístico, basado en el estudio presentado por López y González (2016) sobre el crecimiento de plántulas de pino maximinoii. El código completo está disponible en: https://github.com/Ludwing-MJ/Paquetes_Ej.

8.4.1 Contexto del estudio

Se evaluó el efecto de cinco tratamientos de preparación del terreno sobre el crecimiento en altura de plántulas de pino maximinoii. El experimento incluyó 25 parcelas, con cinco repeticiones por tratamiento, asignadas aleatoriamente. Las mediciones de altura se realizaron después de cinco años de crecimiento.

8.4.2 Implementación del análisis

```r
# Instalación y carga de paquetes necesarios
## Incluye ggplot2, dplyr, tidyr
if (!require("tidyverse")) install.packages("tidyverse")
## Diseños experimentales agrícolas
if (!require("agricolae")) install.packages("agricolae")
```

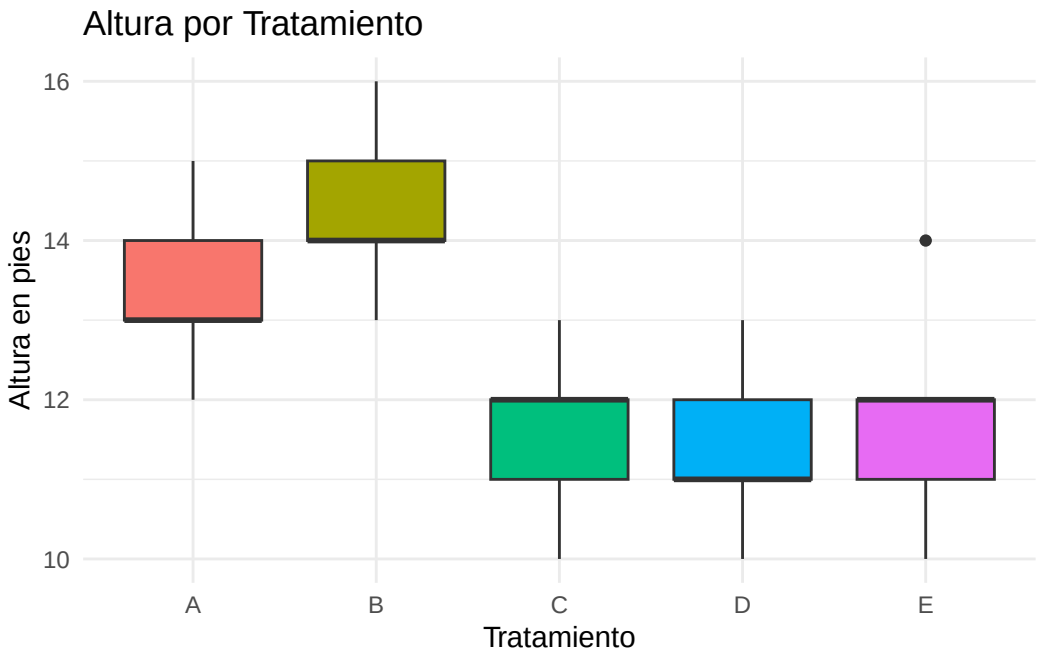
```r
## Importación de archivos Excel
if (!require("readxl")) install.packages("readxl")
## Exportación a Excel
if (!require("writexl")) install.packages("writexl")
## Establecer directorio de trabajo automaticamente
if (!require("rstudioapi")) install.packages("rstudioapi")


# Carga de datos
altura_pino <- read_excel("datos_arboles.xlsx")

# Análisis de varianza y comparaciones múltiples
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
```

```r
comparacion_tukey <- HSD.test(modelo_anova, "tratamiento")
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
```

```r
# 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() +
    theme(legend.position = "none")
```




```
# Exportación de resultados
write_xlsx(comparacion_tukey$groups, "resultados_tukey.xlsx")
ggsave("ggplot_pino.png")
```

Este ejemplo demuestra cómo los diferentes paquetes se integran en un flujo de trabajo coherente, desde la preparación de datos hasta la visualización y exportación de resultados. El uso de paquetes especializados como `agricolae` para el análisis estadístico, `ggplot2` para la visualización y `writexl` para la exportación de resultados, simplifica significativamente el proceso de análisis y presentación de datos (López & González, 2016; R Core Team, 2023).

La estructura modular del código y el uso de paquetes especializados facilitan la reproducibilidad del análisis y permiten adaptarlo fácilmente a otros conjuntos de datos similares.

Parte III

Manipulación de datos

9 Introducción a la manipulación de datos en R

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

9.1 Principales tareas de manipulación de datos

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

9.1.1 Filtrado de datos

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

9.1.2 Selección de variables

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

9.1.3 Transformación de datos

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

9.1.4 Agregación de información

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

9.1.5 Reestructuración de datos

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

9.2 Enfoques disponibles en R para la manipulación de datos

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

9.2.1 Herramientas base de R

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

9.2.2 El enfoque tidyverse

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

10 Manipulación de Datos con Herramientas Base de R

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

10.1 Datos de ejemplo

Para ilustrar las técnicas de manipulación, se emplea un conjunto de datos simulado que representa un experimento agrícola. Este conjunto contiene variables numéricas y categóricas, así como algunos valores faltantes, lo que permite demostrar operaciones comunes en la estadística clásica.

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

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

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

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

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

10.2 Selección y filtrado de datos en data frames

La selección y el filtrado permiten trabajar con subconjuntos de datos relevantes para el análisis estadístico. Estas operaciones se realizan mediante la indexación y el uso de condiciones lógicas.

10.2.1 Selección de columnas

Para seleccionar columnas específicas, se utiliza la notación de corchetes `[, ]`, donde el primer argumento indica las filas y el segundo las columnas.

```r
# Selección de columnas por nombre
datos_mediciones <- datos_cultivo[, c("altura_cm", "peso_gr")]
head(datos_mediciones) # Muestra las primeras filas del resultado
```

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

```r
# Exclusión de una columna
datos_sin_fecha <- datos_cultivo[, !names(datos_cultivo) %in% "fecha_siembra"]
head(datos_sin_fecha) # Muestra las primeras filas del resultado
```

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

```r
# Selección por posición
primeras_tres_columnas <- datos_cultivo[, 1:3]
head(primeras_tres_columnas) # Muestra las primeras filas del resultado
```

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

El uso de nombres de columnas es recomendable para evitar errores si el orden de las variables cambia.

10.2.2 Filtrado de filas por condiciones lógicas

El filtrado de filas se realiza especificando una condición lógica en la parte de filas de la notación de corchetes.

```r
# Filtrar por tratamiento
datos_control <- datos_cultivo[datos_cultivo$tratamiento == "Control", ]
head(datos_control) # Muestra las primeras filas del resultado
```

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

```r
# Filtrar por condición múltiple
datos_altos <- datos_cultivo[datos_cultivo$altura_cm > 65 &
                             datos_cultivo$tratamiento != "Control", ]
head(datos_altos) # Muestra las primeras filas del resultado
```

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

```r
# Uso de subset para mayor legibilidad
datos_fertilizante_A <- subset(datos_cultivo, tratamiento == "Fertilizante A" &
                                bloque %in% c(2,3,4))
head(datos_fertilizante_A) # Muestra las primeras filas del resultado
```

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

La función `subset()` permite expresar condiciones de manera más clara y evita la repetición del nombre del data frame.

10.3 Modificación de variables

La modificación de variables es fundamental para adaptar los datos a los requerimientos de los métodos estadísticos clásicos.

10.3.1 Creación de nuevas columnas

Nuevas variables pueden crearse mediante operaciones aritméticas o lógicas sobre las columnas existentes.

```r
# Índice de crecimiento: relación entre altura y peso
datos_cultivo$indice_crecimiento <- datos_cultivo$altura_cm/datos_cultivo$peso_gr

# Clasificación de peso en alto o bajo
datos_cultivo$categoria_peso <- ifelse(datos_cultivo$peso_gr > 120,
                                       "Alto", "Bajo")

# Transformación logarítmica para normalizar la variable peso
datos_cultivo$log_peso <- log(datos_cultivo$peso_gr)
head(datos_cultivo) # Muestra las primeras filas del resultado
```

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

```
3                NA           Bajo 4.546481
4         0.6453831           Bajo 4.623010
5         0.6350575           Bajo 4.648230
6         1.0565553           Bajo 4.354141
```

La función `ifelse(condición, valor_si_verdadero, valor_si_falso)` permite crear variables categóricas a partir de condiciones lógicas.

10.3.2 Recodificación de variables categóricas

La recodificación es útil para adaptar los niveles de factores a los requerimientos del análisis.

```
# Recodificación de niveles de daño por plaga
datos_cultivo$nivel_daño <- factor(datos_cultivo$daño_plaga,
                                   levels = c("Bajo", "Medio", "Alto"),
                                   labels = c("1", "2", "3"))

# Variable indicadora para tratamiento control
datos_cultivo$es_control <- ifelse(datos_cultivo$tratamiento == "Control", 1, 0)
head(datos_cultivo)  # Muestra las primeras filas del resultado
```

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

La función `factor()` permite definir el orden y las etiquetas de los niveles de una variable categórica, lo cual es relevante en análisis como ANOVA.

10.4 Ordenamiento y agrupamiento de datos

El ordenamiento y el agrupamiento facilitan la exploración y el cálculo de estadísticas descriptivas por grupos, pasos previos a los análisis clásicos.

10.4.1 Ordenamiento de datos

El ordenamiento se realiza con la función `order()`, que devuelve el índice de ordenación.

```
# Ordenar por altura
datos_ordenados <- datos_cultivo[order(datos_cultivo$altura_cm), ]
head(datos_ordenados) # Muestra las primeras filas del resultado
```

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

```
# Ordenar por tratamiento y peso descendente
datos_ordenados_multi <- datos_cultivo[order(datos_cultivo$tratamiento,
                                             -datos_cultivo$peso_gr), ]
head(datos_ordenados_multi) # Muestra las primeras filas del resultado
```

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

El signo negativo delante de una variable numérica indica orden descendente.

10.4.2 Cálculos estadísticos por grupo

El cálculo de medias, desviaciones estándar y otros estadísticos por grupo es esencial en la estadística clásica. Las funciones `tapply()`, `aggregate()` y `by()` permiten realizar estos cálculos.

```
# Media de altura por tratamiento usando tapply
medias_altura <- tapply(datos_cultivo$altura_cm, datos_cultivo$tratamiento,
                        mean, na.rm = TRUE)
medias_altura
```

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

```
# Estadísticas descriptivas por tratamiento usando aggregate
estadisticas_grupo <- aggregate(cbind(altura_cm, peso_gr) ~ tratamiento,
                                data = datos_cultivo,
                                FUN = function(x)
                                  c(media = mean(x, na.rm = TRUE),
                                    sd = sd(x, na.rm = TRUE)))
estadisticas_grupo
```

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

En `tapply()`, el primer argumento es el vector de datos, el segundo es el factor de agrupamiento y el tercero es la función estadística. El argumento `na.rm = TRUE` indica que se deben omitir los valores faltantes.

10.5 Manejo de valores faltantes y duplicados

El tratamiento de valores faltantes y duplicados es crucial para evitar sesgos y errores en los análisis estadísticos clásicos (Wickham & Grolemund, 2017).

10.5.1 Identificación y manejo de valores faltantes

```
# Conteo de valores faltantes por columna
na_por_columna <- colSums(is.na(datos_cultivo))
na_por_columna # Numero de valores faltantes por columna
```

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

```
# Eliminación de filas con valores faltantes
datos_completos <- na.omit(datos_cultivo)

# Imputación de valores faltantes con la media
datos_cultivo$altura_cm[is.na(datos_cultivo$altura_cm)] <-
  mean(datos_cultivo$altura_cm, na.rm = TRUE)
```

La función `is.na()` identifica los valores faltantes, `colSums()` suma por columna y `na.omit()` elimina filas con al menos un NA.

10.5.2 Manejo de duplicados

```
# Identificación de duplicados
duplicados <- duplicated(datos_cultivo[, c("tratamiento", "bloque")])
summary(duplicados) # Resume el número de duplicados encontrados
```

```
   Mode   FALSE
logical      20
```

```
# Eliminación de duplicados
datos_sin_duplicados <- unique(datos_cultivo)
```

La función `duplicated()` devuelve un vector lógico que indica si una fila es duplicada respecto a las variables seleccionadas. `unique()` elimina las filas duplicadas.

10.5.3 Ejemplo práctico: Preparación de datos para análisis de varianza (ANOVA)

A continuación se muestra cómo preparar un conjunto de datos para un análisis de varianza, integrando las técnicas presentadas.

```
# 1. Eliminar valores faltantes
datos_anova <- na.omit(datos_cultivo)

# 2. Seleccionar variables relevantes
datos_anova <- datos_anova[, c("tratamiento", "bloque", "altura_cm")]
```

```r
# 3. Recodificar variables como factores
datos_anova$tratamiento <- factor(datos_anova$tratamiento)
datos_anova$bloque <- factor(datos_anova$bloque)

# 4. Estadísticas descriptivas previas
estadisticas_previas <- aggregate(altura_cm ~ tratamiento,
                                  data = datos_anova,
                                  FUN = function(x)
                                    c(media = mean(x),
                                      sd = sd(x),
                                      n = length(x)))
estadisticas_previas
```

```
     tratamiento altura_cm.media altura_cm.sd altura_cm.n
1        Control       63.525000     3.168990    4.000000
2 Fertilizante A       63.275000    13.077812    4.000000
3 Fertilizante B       70.225000     4.823812    4.000000
4 Fertilizante C       71.300000     9.268945    4.000000
```

```r
# 5. Realizar ANOVA
modelo_anova <- aov(altura_cm ~ tratamiento + bloque, data = datos_anova)
summary(modelo_anova)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
tratamiento  3  219.3   73.09   1.277  0.346
bloque       4  413.0  103.25   1.805  0.221
Residuals    8  457.8   57.22
```

En este ejemplo, se eliminan los valores faltantes, se seleccionan las variables necesarias, se recodifican como factores y se calculan estadísticas descriptivas antes de aplicar el modelo ANOVA, una técnica central en la estadística clásica para comparar medias entre grupos (Montgomery, 2017; R Core Team, 2023).

11 Manipulación de datos con dplyr y tidyr

11.1 Introducción a los paquetes dplyr y tidyr

Los paquetes dplyr y tidyr forman parte del ecosistema tidyverse y han sido diseñados específicamente para la manipulación y transformación de datos en R. En el contexto del análisis estadístico clásico, estas herramientas facilitan la preparación de datos para técnicas como ANOVA, regresión y pruebas de hipótesis, ofreciendo una sintaxis clara y consistente (Wickham & Grolemund, 2017).

El paquete dplyr se especializa en la manipulación de datos tabulares, proporcionando funciones específicas para operaciones comunes como filtrado, selección y agregación. Por su parte, tidyr se centra en la reorganización de datos, permitiendo transformaciones entre diferentes formatos según los requerimientos del análisis estadístico (Wickham et al., 2023).

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.

11.2 Datos ejemplo

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

```
# Cargar el paquete necesario
library(dplyr)
library(tidyr)

# Crear datos de ejemplo
set.seed(123) # Para reproducibilidad

datos_cultivo <- data.frame(
  parcela = 1:20,
  tratamiento = rep(c("Control", "Fertilizante A",
                      "Fertilizante B", "Fertilizante C"), each = 5),
  bloque = rep(1:5, times = 4),
  altura_cm = round(rnorm(20, mean = 65, sd = 10), 1),
  peso_gr = round(rnorm(20, mean = 120, sd = 25), 1),
```

```
  daño_plaga = sample(c("Alto", "Medio", "Bajo"), 20, replace = TRUE),
  fecha_siembra = as.Date("2024-01-01") + sample(1:10, 20, replace = TRUE)
)

# Agregar algunos valores NA para ejemplos posteriores
datos_cultivo$altura_cm[c(3, 15)] <- NA
datos_cultivo$peso_gr[c(7, 18)] <- NA
```

11.3 Operaciones básicas con dplyr

11.3.1 Filtrado de datos con filter()

La función `filter()` permite seleccionar filas de un data frame que cumplen condiciones lógicas. Su sintaxis es:

```
filter(.data, ...)
```

1. `.data`: el data frame o tibble sobre el que se aplicará el filtrado.
2. `...`: una o más condiciones lógicas que deben cumplirse para que una fila sea seleccionada.

Ejemplo:

```
# Filtrar las parcelas con tratamiento "Control"
datos_control <- filter(datos_cultivo, tratamiento == "Control")
head(datos_control) # Visualizar el resultado
```

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

```
# Filtrar parcelas con altura mayor a 65 cm y tratamiento distinto de "Control"
datos_altos <- filter(datos_cultivo, altura_cm > 65, tratamiento != "Control")
head(datos_altos)   # Visualizar el resultado
```

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

En el primer caso, solo se seleccionan las filas donde la columna `tratamiento` es igual a "Control". En el segundo, se seleccionan las filas donde la altura es mayor a 65 cm y el tratamiento no es "Control". Es importante notar que, a diferencia de R base, no es necesario escribir el nombre del data frame en cada condición, lo que simplifica la sintaxis (Wickham & Grolemund, 2017).

11.3.2 Selección de columnas con select()

La función `select()` permite extraer columnas específicas de un data frame. Su sintaxis es:

```
select(.data, ...)
```

1. `.data`: el data frame o tibble de entrada.
2. `...`: nombres de las columnas a seleccionar, o funciones auxiliares como `starts_with()`, `ends_with()`, `contains()`, etc.

Ejemplo:

```
# Seleccionar las columnas altura_cm y peso_gr
datos_mediciones <- select(datos_cultivo, altura_cm, peso_gr)
head(datos_mediciones)  # Visualizar el resultado
```

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

```
# Excluir la columna fecha_siembra
datos_sin_fecha <- select(datos_cultivo, -fecha_siembra)
head(datos_sin_fecha)  # Visualizar el resultado
```

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

```
# Seleccionar columnas que terminan en "cm" o "gr"
datos_numericos <- select(datos_cultivo, ends_with("cm"), ends_with("gr"))
head(datos_numericos)  # Visualizar el resultado
```

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

El uso de funciones auxiliares permite seleccionar columnas de manera flexible, lo que resulta útil en bases de datos extensas (Wickham et al., 2023).

11.3.3 Creación y transformación de variables con mutate()

La función `mutate()` permite crear nuevas columnas o modificar las existentes. Su sintaxis es:

```
mutate(.data, ...)
```

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

Ejemplo:

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

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

La función `ifelse(condición, valor_si_verdadero, valor_si_falso)` permite crear variables categóricas a partir de condiciones lógicas, lo que es común en la estadística clásica para definir grupos o categorías (Wickham & Grolemund, 2017).

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

La función `group_by()` agrupa los datos según una o más variables, y `summarize()` calcula estadísticas resumen por grupo. Sus sintaxis son:

```
group_by(.data, ...)
summarize(.data, ...)
```

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

Ejemplo:

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

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

1. `mean(altura_cm, na.rm = TRUE)`: calcula la media, ignorando los valores NA.
2. `sd(altura_cm, na.rm = TRUE)`: calcula la desviación estándar.
3. `n()`: cuenta el número de observaciones en cada grupo.

Estas funciones son esenciales para obtener resúmenes estadísticos por grupo, como promedios por tratamiento en un experimento clásico (Wickham et al., 2023).

11.3.5 Ordenamiento de datos con arrange()

La función `arrange()` permite ordenar los datos según una o más variables. Su sintaxis es:

```
arrange(.data, ...)
```

1. `.data`: el data frame o tibble de entrada.
2. `...`: variables por las que se desea ordenar; se puede usar `desc()` para orden descendente.

Ejemplo:

```
# Ordenar por altura de menor a mayor
datos_ordenados <- arrange(datos_cultivo, altura_cm)
# Ordenar por tratamiento y peso descendente
datos_ordenados_multi <- arrange(datos_cultivo,
                                 tratamiento,
                                 desc(peso_gr))
```

El ordenamiento es útil para identificar valores extremos o preparar tablas para reportes (Wickham & Grolemund, 2017).

11.4 Introducción a los pipes (%>%)

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

La sintaxis básica del operador pipe es:

```
# Estructura usando pipes
datos %>% funcion()
# Equivalente a la siguiente estructura anidada
funcion(datos)
```

11.4.1 Ventajas del uso de pipes

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

1. **Legibilidad mejorada**: Las operaciones se leen de izquierda a derecha y de arriba hacia abajo, siguiendo el orden natural de lectura.
2. **Reducción de objetos intermedios**: No es necesario crear variables temporales para almacenar resultados intermedios.
3. **Facilidad de depuración**: Cada paso puede ser comentado o modificado independientemente.
4. **Claridad en la secuencia de operaciones**: El flujo de trabajo se hace explícito y fácil de seguir.

11.4.2 Ejemplo práctico

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

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

En este ejemplo, primero se filtran las filas sin valores faltantes en `altura_cm`, luego se agrupan por `tratamiento` y finalmente se calcula la media de altura para cada grupo.

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

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

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

Cada paso es explícito y se puede leer de arriba hacia abajo, lo que facilita la comprensión y depuración del análisis (Wickham & Grolemund, 2017).

11.5 Transformaciones de datos con tidyr

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

11.5.1 Transformación de formato ancho a largo con pivot_longer()

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

La sintaxis principal de `pivot_longer()` es la siguiente:

```
pivot_longer(
  data,          # Data frame o tibble de entrada
  cols,          # Columnas a transformar (vector de nombres, rango o función selectora)
  names_to,      # Nombre de la nueva columna que contendrá los nombres originales de las
  values_to      # Nombre de la nueva columna que contendrá los valores
)
```

Por ejemplo, considérese un subconjunto pequeño de datos con alturas y pesos de tres parcelas:

```
library(tidyr)
# Crear un dataframe para visualizar mejor los ejemplos
mini_datos <- data.frame(
  parcela = c(1, 2, 3),
  altura_cm = c(70, 65, 60),
  peso_gr = c(120, 115, 110)
)
mini_datos # visualizar el data frame original
```

```
  parcela altura_cm peso_gr
1       1        70     120
2       2        65     115
3       3        60     110
```

Para transformar este data frame a formato largo, se utiliza:

```
mini_largo <- pivot_longer(
  data = mini_datos,
  cols = c(altura_cm, peso_gr), # Columnas a transformar
  names_to = "variable",        # Nueva columna para los nombres de las variables origina
  values_to = "valor"           # Nueva columna para los valores
)
mini_largo # Visualizar el resultado
```

```
# A tibble: 6 x 3
  parcela variable  valor
    <dbl> <chr>     <dbl>
```

```
1       1 altura_cm    70
2       1 peso_gr     120
3       2 altura_cm    65
4       2 peso_gr     115
5       3 altura_cm    60
6       3 peso_gr     110
```

En este ejemplo, el argumento `cols` indica las columnas a transformar, `names_to` define el nombre de la columna que almacenará los nombres de las variables originales y `values_to` define el nombre de la columna que almacenará los valores correspondientes (Wickham & Grolemund, 2017).

11.5.2 Transformación de formato largo a ancho con pivot_wider()

La función `pivot_wider()` realiza la transformación inversa, es decir, convierte un data frame en formato largo a formato ancho. Esto es útil para reportes o análisis que requieren que cada variable esté en una columna separada (Wickham & Grolemund, 2017).

La sintaxis principal de `pivot_wider()` es:

```
pivot_wider(
  data,         # Data frame o tibble de entrada
  names_from,   # Columna cuyos valores se usarán como nombres de nuevas columnas
  values_from   # Columna cuyos valores se distribuirán en las nuevas columnas
)
```

Utilizando el subconjunto `mini_largo` generado previamente:

```
mini_ancho <- pivot_wider(
  data = mini_largo,
  names_from = variable,  # Columna que define los nombres de las nuevas columnas
  values_from = valor     # Columna que define los valores a distribuir
)
mini_ancho # Visualizar el resultado
```

```
# A tibble: 3 x 3
  parcela altura_cm peso_gr
    <dbl>     <dbl>   <dbl>
1       1        70     120
2       2        65     115
3       3        60     110
```

En este caso, el argumento `names_from` indica la columna cuyos valores se convertirán en nombres de nuevas columnas y `values_from` indica la columna cuyos valores se colocarán en las nuevas columnas (Wickham & Grolemund, 2017).

11.5.3 Manipulación de variables compuestas con separate() y unite()

En ocasiones, una columna puede contener información de varias variables codificadas en un solo valor. Para dividir o combinar estas variables, se utilizan las funciones `separate()` y `unite()` (Wickham & Grolemund, 2017).

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

```
separate(
  data,          # Data frame o tibble de entrada
  col,           # Columna a dividir
  into,          # Vector con los nombres de las nuevas columnas
  sep            # Carácter separador (por defecto, cualquier carácter no alfanumérico)
)
```

Por ejemplo, considérese el siguiente subconjunto:

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

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

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

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

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

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

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

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

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

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

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

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

11.6 Ejemplo integrado: Preparación de datos para análisis estadístico clásico

A continuación se muestra un flujo de trabajo típico para preparar los datos del experimento agrícola antes de realizar un análisis de varianza (ANOVA), utilizando dplyr y tidyr.

```
# 1. Eliminar valores faltantes en altura y peso
datos_limpios <- datos_cultivo %>%
  filter(!is.na(altura_cm), !is.na(peso_gr))

# 2. Crear variables derivadas
datos_limpios <- datos_limpios %>%
```

```r
  mutate(
    indice_crecimiento = altura_cm / peso_gr,
    categoria_altura = ifelse(altura_cm > 65, "Alto", "Bajo")
  )

# 3. Agrupar por tratamiento y calcular estadísticas descriptivas
resumen_tratamiento <- datos_limpios %>%
  group_by(tratamiento) %>%
  summarize(
    media_altura = mean(altura_cm),
    sd_altura = sd(altura_cm),
    n = n()
  )
resumen_tratamiento
```

```
# A tibble: 4 x 4
  tratamiento    media_altura sd_altura     n
  <chr>                 <dbl>     <dbl> <int>
1 Control                63.5      3.17     4
2 Fertilizante A         63.3     13.1      4
3 Fertilizante B         70.2      4.82     4
4 Fertilizante C         71.3      9.27     4
```

```r
# 4. Transformar a formato largo para análisis multivariado
datos_largo <- datos_limpios %>%
  pivot_longer(
    cols = c(altura_cm, peso_gr),
    names_to = "variable",
    values_to = "valor"
  )
head(datos_largo)
```

```
# A tibble: 6 x 11
  parcela tratamiento bloque daño_plaga fecha_siembra indice_crecimiento
    <int> <chr>        <int> <chr>      <date>                       <dbl>
1       1 Control          1 Medio      2024-01-10                   0.637
2       1 Control          1 Medio      2024-01-10                   0.637
3       2 Control          2 Medio      2024-01-08                   0.547
4       2 Control          2 Medio      2024-01-08                   0.547
5       4 Control          4 Medio      2024-01-09                   0.645
6       4 Control          4 Medio      2024-01-09                   0.645
# i 5 more variables: altura_m <dbl>, peso_kg <dbl>, categoria_altura <chr>,
#   variable <chr>, valor <dbl>
```

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

La manipulación de datos es una etapa fundamental en el análisis estadístico clásico. Existen dos enfoques principales en R: el uso de funciones base y el uso de paquetes del tidyverse, como dplyr y tidyr. A continuación se presenta un cuadro comparativo que resume las principales diferencias entre ambos enfoques, considerando aspectos como sintaxis, legibilidad, flexibilidad y reproducibilidad (Wickham & Grolemund, 2017).

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

Como se observa, el enfoque tidyverse ofrece ventajas notables en términos de claridad, reproducibilidad y facilidad de uso, especialmente en flujos de trabajo complejos o colaborativos. No obstante, el conocimiento de las funciones base de R sigue siendo valioso, ya que permite comprender el funcionamiento interno del lenguaje y resolver tareas específicas de manera eficiente (Wickham & Grolemund, 2017).

Parte IV

Visualización de datos

12 Introducción a la visualización de datos

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

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

12.1 Historia y evolución de la visualización en estadística

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

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

12.2 Principios básicos de la visualización efectiva

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

eficiencia. Estos principios han sido ampliamente discutidos en la literatura especializada, destacando su relevancia en la comunicación gráfica de datos (Tufte, 2001; Cleveland, 1993).

12.2.1 Claridad

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

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

12.2.2 Precisión

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

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

12.2.3 Eficiencia

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

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

12.2.4 Errores comunes a evitar en la visualización de datos

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

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

12.2.5 Recomendaciones para una visualización efectiva

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

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

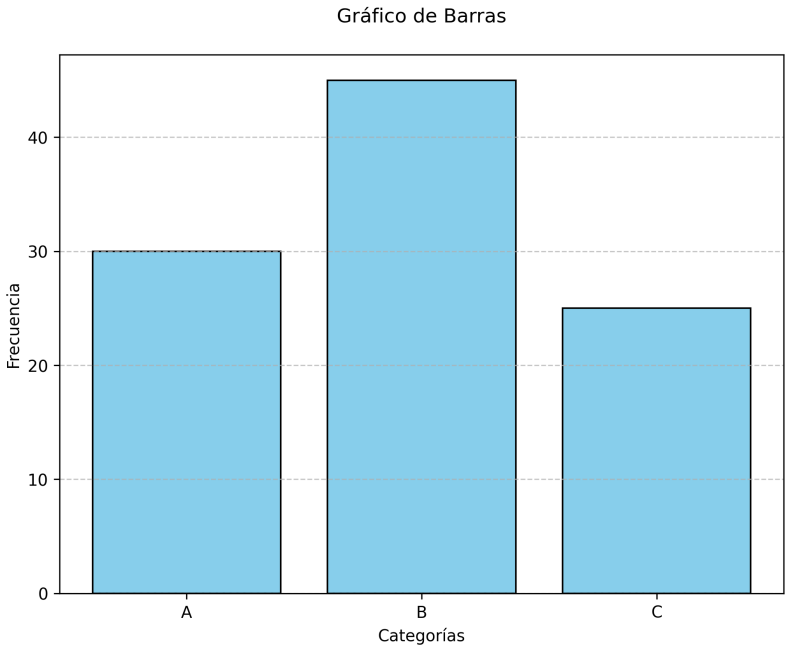
3. Revisar y validar los gráficos antes de su presentación, asegurando que representen fielmente los datos y sean interpretables por el público objetivo.
4. Utilizar recursos visuales (colores, formas, tamaños) de manera coherente y justificada, evitando la sobrecarga visual.
5. Documentar las decisiones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

12.3 Tipos de gráficos y su utilidad en estadística clásica

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

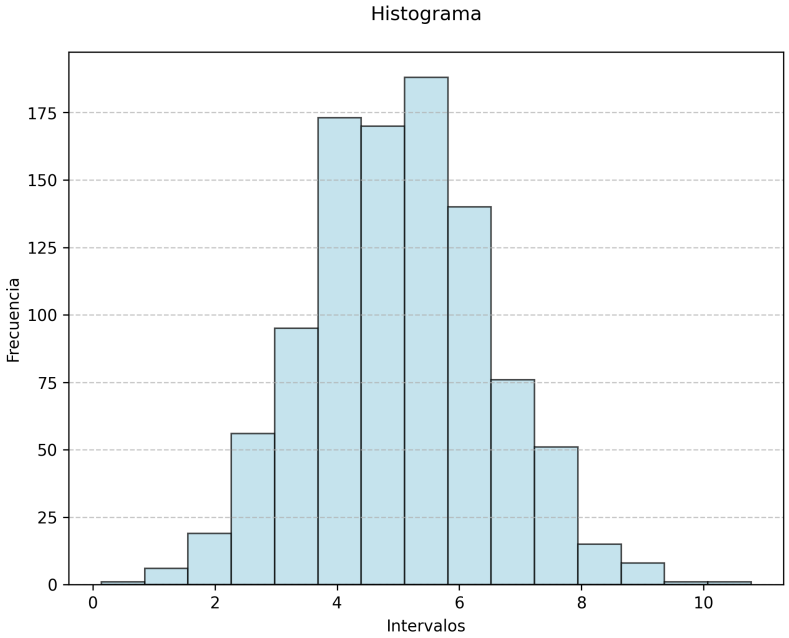
12.3.1 Gráficos de barras

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



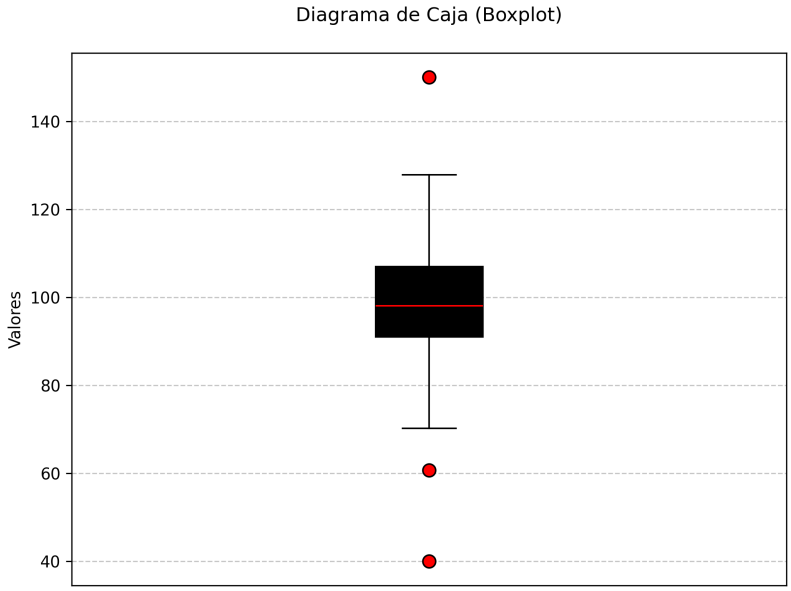
12.3.2 Histogramas

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



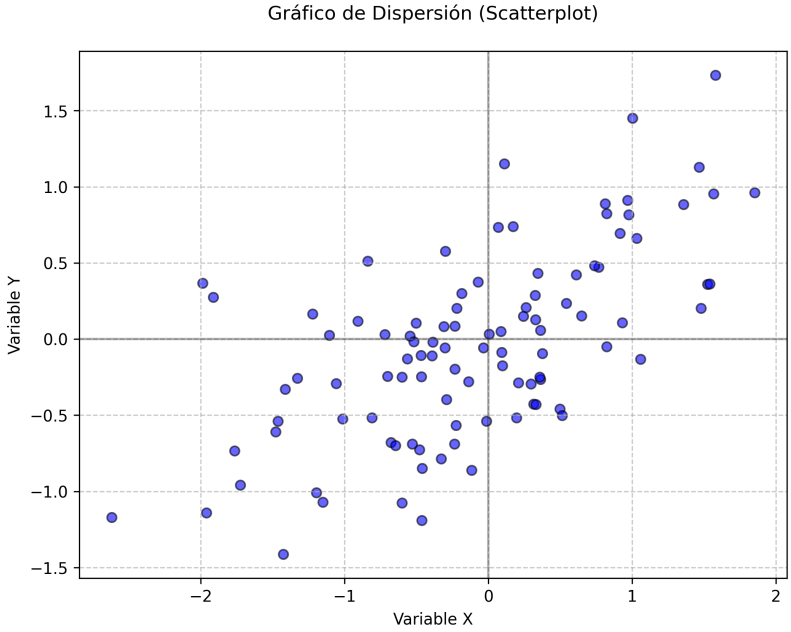
12.3.3 Diagramas de caja (boxplots)

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



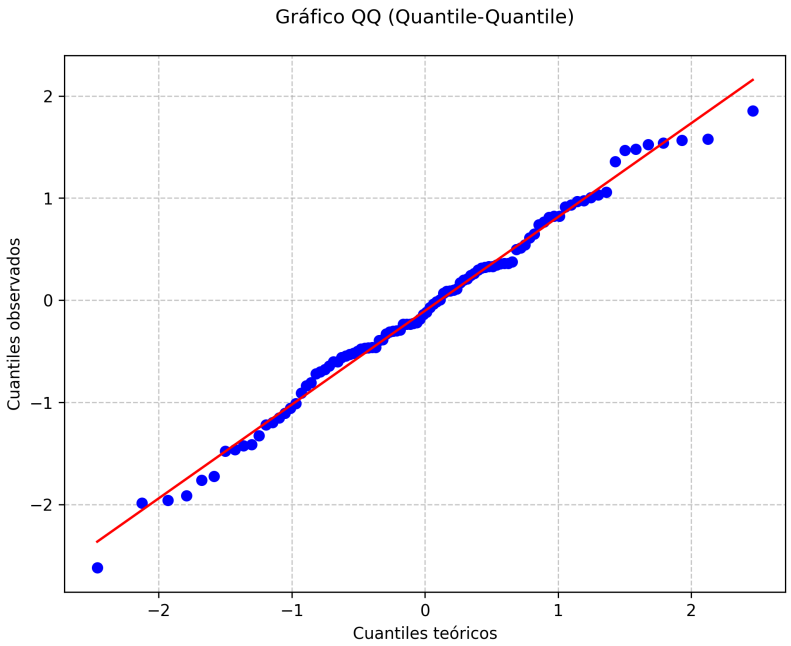
12.3.4 Gráficos de dispersión (scatterplots)

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



12.3.5 Gráficos QQ (quantile-quantile)

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



12.3.6 Gráficos de residuos

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

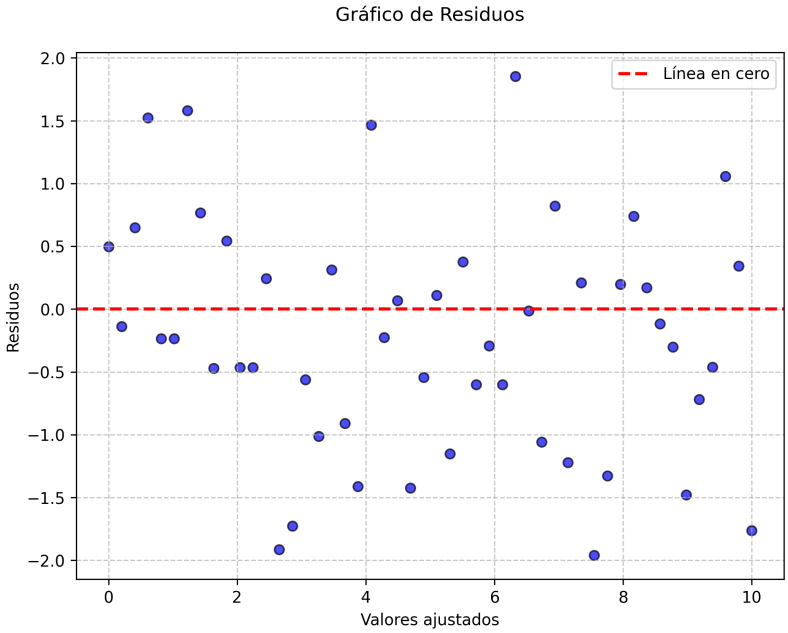


Gráfico de Residuos
Línea en cero
Residuos
Valores ajustados
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
2
4
6
8
10

13 Visualizaciones Base de R

La comprensión de los principios básicos de la visualización efectiva constituye la base para una correcta interpretación y comunicación de los datos en el análisis estadístico. Sin embargo, para llevar estos principios a la práctica, es indispensable conocer y dominar las herramientas que permiten construir representaciones gráficas de calidad. En este sentido, el lenguaje R ofrece un conjunto robusto de funciones base que facilitan la creación de diversos tipos de gráficos, esenciales para la exploración y validación de los supuestos estadísticos en la estadística clásica. El siguiente capítulo se centra en el uso de estas funciones, proporcionando ejemplos y recomendaciones para su aplicación en el análisis de datos, y mostrando cómo los conceptos teóricos previamente expuestos se materializan en la práctica cotidiana del análisis estadístico (Venables & Ripley, 2002).

13.1 Funciones gráficas básicas de R

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

Entre las funciones más utilizadas se encuentran:

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

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

13.2 Creación de gráficos exploratorios

La creación de gráficos exploratorios en R base es fundamental para el análisis inicial de los datos y la detección de patrones, tendencias y anomalías. A continuación se detallan los principales tipos de gráficos, su sintaxis y los argumentos más relevantes, acompañados de ejemplos y explicaciones pedagógicas (Venables & Ripley, 2002; Murrell, 2018).

13.2.1 Histogramas

La función principal para crear histogramas en R es `hist()`. Su sintaxis general es:

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

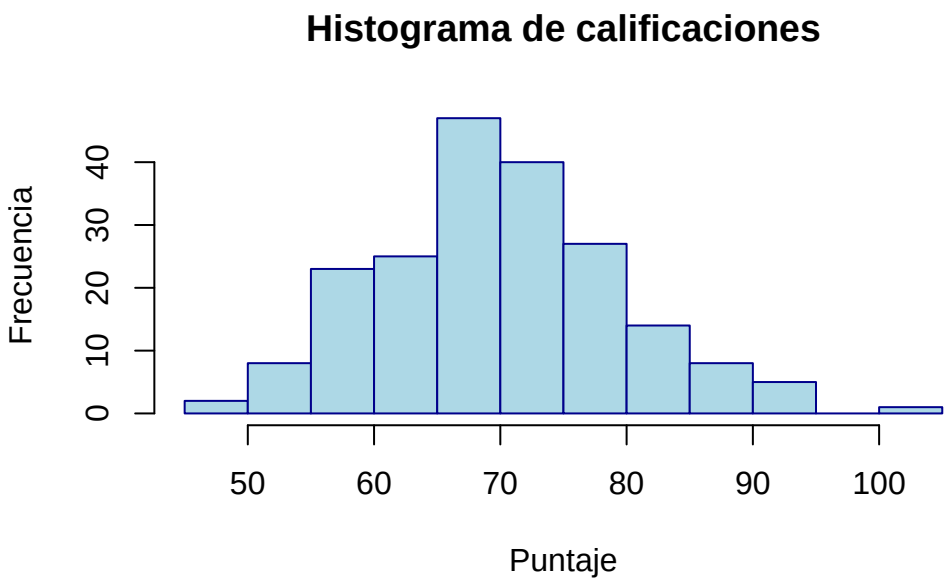
Explicación de los argumentos:

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

Ejemplo detallado:

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

# Histograma personalizado
hist(notas,
     breaks = 15,                # Número de intervalos
```

```
     freq = TRUE,                # Mostrar frecuencias absolutas
     col = "lightblue",          # Color de las barras
     border = "darkblue",        # Color del borde
     main = "Histograma de calificaciones", # Título
     xlab = "Puntaje",           # Etiqueta eje X
     ylab = "Frecuencia")        # Etiqueta eje Y
```



13.2.2 Diagramas de caja (boxplots)

La función principal es `boxplot()`. Su sintaxis general es:

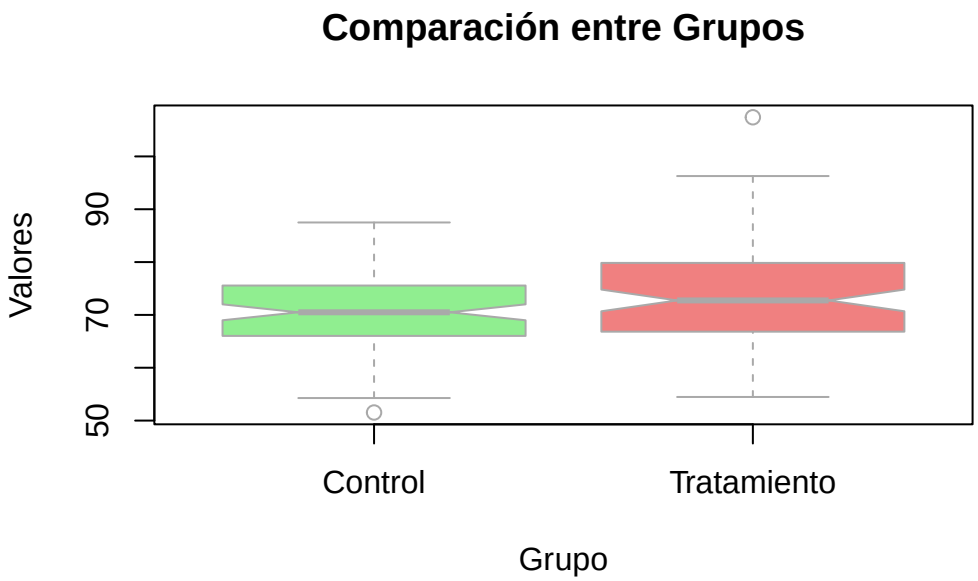
```
boxplot(formula,
        data = NULL,
        main = NULL,
        xlab = NULL,
        ylab = NULL,
        col = NULL,
        border = NULL,
        notch = FALSE,
        outline = TRUE,
        ...)
```

1. `formula`: Expresión del tipo `y ~ grupo` para comparar grupos.
2. `data`: Data frame donde buscar las variables.
3. `main`, `xlab`, `ylab`: Títulos y etiquetas.
4. `col`: Colores de las cajas.

5. `border`: Color del borde de las cajas.
6. `notch`: Si es TRUE, añade una muesca para comparar medianas.
7. `outline`: Si es TRUE, muestra valores atípicos.
8. `...`: Otros argumentos gráficos.

Ejemplo detallado:

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

# Boxplot personalizado
boxplot(valores ~ grupo,
        main = "Comparación entre Grupos",
        xlab = "Grupo",
        ylab = "Valores",
        col = c("lightgreen", "lightcoral"),
        border = "darkgray",
        notch = TRUE,              # Mostrar muesca para comparar medianas
        outline = TRUE)            # Mostrar valores atípicos
```



13.2.3 Gráficos de dispersión

La función principal es `plot()`. Su sintaxis general para dos variables es:

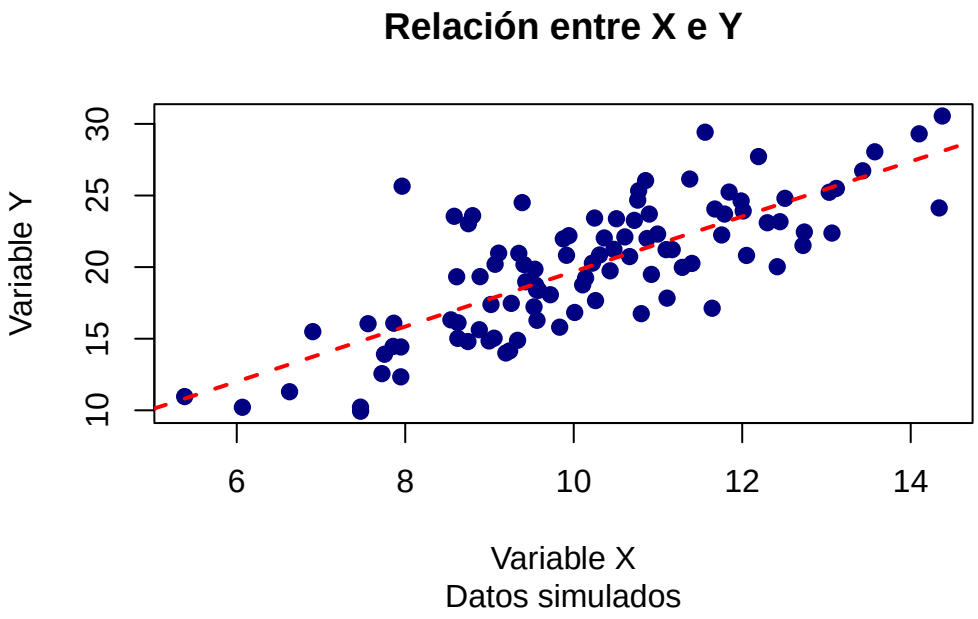
```
plot(x, y,
     type = "p",
     main = NULL,
     sub = NULL,
     xlab = NULL,
     ylab = NULL,
     pch = 1,
     col = NULL,
     cex = 1,
     ...)
```

Explicación de los argumentos:

1. `x`, `y`: Vectores numéricos de igual longitud.
2. `type`: Tipo de gráfico ("p" para puntos, "l" para líneas, "b" para ambos).
3. `main`, `sub`: Título principal y subtítulo.
4. `xlab`, `ylab`: Etiquetas de los ejes.
5. `pch`: Tipo de símbolo para los puntos (1: círculo, 16: círculo sólido, 17: triángulo, etc.).
6. `col`: Color de los puntos.
7. `cex`: Tamaño relativo de los puntos.
8. `...`: Otros argumentos gráficos.

Ejemplo detallado:

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

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

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



13.2.4 Gráficos de líneas

Para series temporales o secuencias, se usa `plot()` con `type = "l"`:

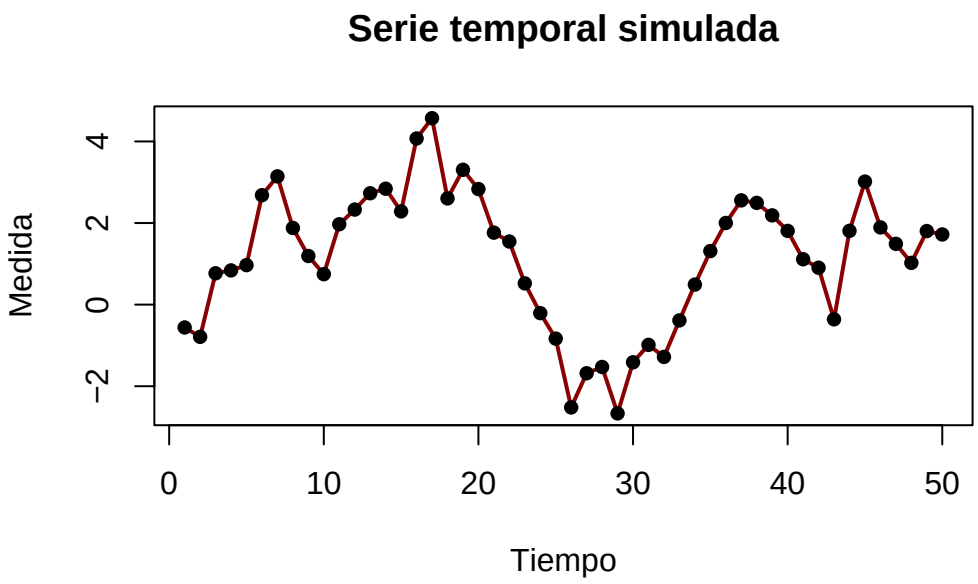
```
plot(x, y,
     type = "l",
     main = NULL,
     xlab = NULL,
     ylab = NULL,
     col = NULL,
     lwd = 1,
     ...)
```

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

Ejemplo detallado:

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

# Gráfico de líneas
plot(tiempo, medidas,
     type = "l",                  # Tipo de gráfico: línea
     main = "Serie temporal simulada",
     xlab = "Tiempo",
```

```
     ylab = "Medida",
     col = "darkred",
     lwd = 2)                      # Grosor de la línea

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



13.3 Visualización para la comprobación de supuestos estadísticos

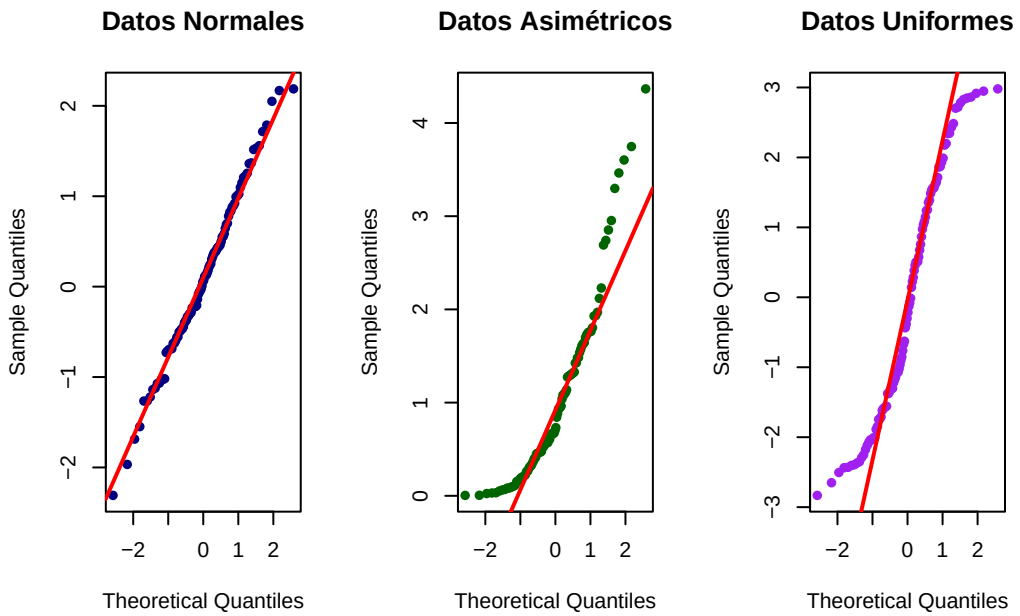
La validación gráfica de supuestos estadísticos es fundamental para garantizar la validez de los análisis en estadística clásica. Esta sección se centra en las herramientas gráficas específicas que R base proporciona para evaluar los supuestos de normalidad, homocedasticidad y linealidad, esenciales en pruebas como ANOVA y regresión lineal (Venables & Ripley, 2002).

13.3.1 Gráficos Q-Q para evaluar normalidad

Los gráficos Q-Q (quantile-quantile) constituyen una herramienta visual poderosa para evaluar el ajuste de los datos a una distribución teórica. Cuando los puntos se alinean sobre la diagonal, se puede inferir que los datos siguen la distribución de referencia, típicamente la normal. Las desviaciones sistemáticas de esta línea sugieren alejamientos de la normalidad que pueden requerir transformaciones de datos o el uso de métodos no paramétricos (Cleveland, 1993).

La sintaxis básica para crear gráficos Q-Q en R incluye dos funciones principales: `qqnorm()` para crear el gráfico base y `qqline()` para añadir la línea de referencia. A continuación se presenta un ejemplo comentado que ilustra su implementación:

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

# Configuración de la ventana gráfica para múltiples gráficos
par(mfrow = c(1,3))

# Gráfico Q-Q para datos normales
qqnorm(datos_normales,
       main = "Datos Normales",
       pch = 16,                # Tipo de punto: círculo sólido
       col = "navy")           # Color de los puntos
qqline(datos_normales,         # Línea de referencia
       col = "red",            # Color de la línea
       lwd = 2)                # Grosor de la línea

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

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



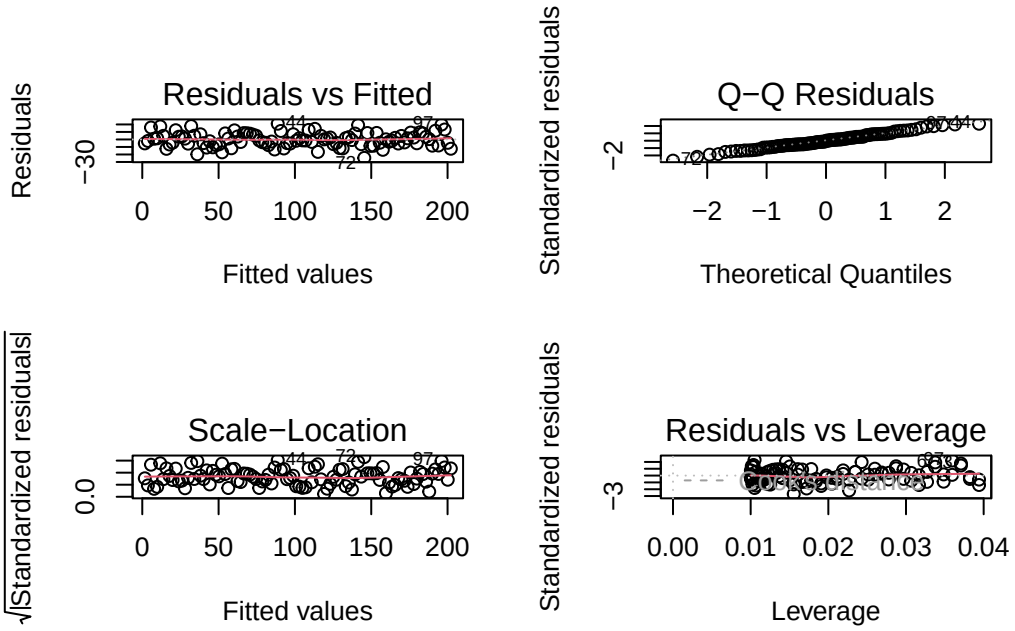

```
# Restaurar la configuración original de la ventana gráfica
par(mfrow = c(1,1))
```

La interpretación de estos gráficos se basa en el patrón que forman los puntos en relación con la línea de referencia. Según Venables & Ripley (2002), las desviaciones más comunes incluyen: 1) Colas pesadas, cuando los extremos se alejan de la línea, 2) Asimetría, cuando se forma un patrón curvilíneo, y 3) Bimodalidad, cuando aparece un patrón en forma de S.

13.3.2 Gráficos de diagnóstico para modelos de regresión

Los gráficos de diagnóstico en regresión permiten evaluar simultáneamente varios supuestos del modelo lineal. R proporciona una serie de gráficos diagnósticos automáticos a través de la función `plot()` aplicada a objetos de clase `lm`. El siguiente ejemplo ilustra su implementación y uso:

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

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

# Gráficos de diagnóstico
par(mfrow = c(2,2))                       # Configurar ventana 2x2
plot(modelo)                              # Generar gráficos diagnósticos
```




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

Los gráficos diagnósticos generados incluyen, según Murrell (2018): 1) Residuos vs valores ajustados, para evaluar linealidad y homocedasticidad, 2) Q-Q de residuos, para verificar normalidad, 3) Scale-Location, para examinar la homogeneidad de varianza, y 4) Residuos vs Leverage, para identificar observaciones influyentes.

13.4 Personalización de gráficos en R base

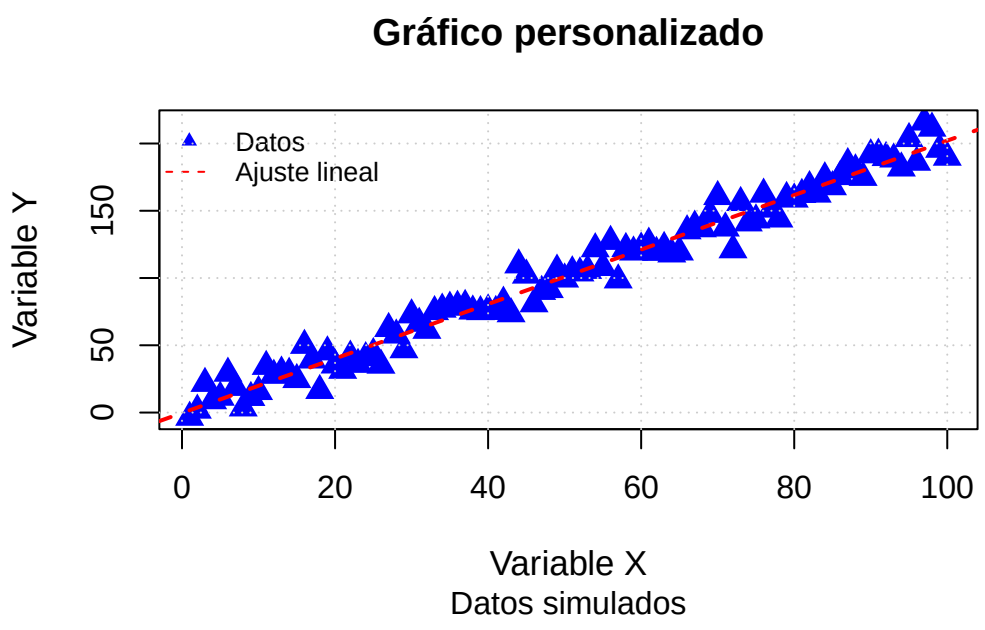
La personalización se logra modificando los argumentos de las funciones gráficas y añadiendo elementos con funciones auxiliares. A continuación se explican los argumentos más importantes y se muestra un ejemplo integral (Murrell, 2018).

13.4.1 Principales argumentos y funciones:

1. `main`, `sub`, `xlab`, `ylab`: Títulos y etiquetas.
2. `col`, `border`, `pch`, `lty`, `lwd`: Colores, símbolos, tipo y grosor de líneas.
3. `cex`, `cex.axis`, `cex.lab`, `cex.main`: Tamaño de símbolos y textos.
4. `legend()`: Añade leyendas.
5. `text()`: Añade texto en posiciones específicas.
6. `abline()`: Añade líneas horizontales, verticales o de regresión.
7. `grid()`: Añade cuadrícula.

13.4.2 Ejemplo integral

```r
# Gráfico de dispersión personalizado
plot(x, y,
     main = "Gráfico personalizado",
     sub = "Datos simulados",
     xlab = "Variable X",
     ylab = "Variable Y",
     col = "blue",                # Color de los puntos
     pch = 17,                    # Triángulo sólido
     cex = 1.5,                   # Tamaño de los puntos
     cex.main = 1.2,              # Tamaño del título
     cex.lab = 1.1)               # Tamaño de etiquetas

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

# Añadir leyenda
legend("topleft",
       legend = c("Datos", "Ajuste lineal"),
       pch = c(17, NA),
       lty = c(NA, 2),
       col = c("blue", "red"),
       bty = "n",
       cex = 0.8)

# Añadir cuadrícula
grid(col = "gray80", lty = "dotted")
```



Este ejemplo muestra cómo modificar títulos, etiquetas, colores, símbolos, tamaños, añadir líneas de tendencia, leyendas y cuadrículas para lograr una visualización clara y profesional (Murrell, 2018; Venables & Ripley, 2002).

14 Visualización de datos con ggplot2

14.1 Contexto y origen de la base de datos utilizada

La base de datos empleada en este capítulo corresponde a un estudio transversal realizado en la Universidad de San Carlos de Guatemala en 2002. El objetivo del estudio fue caracterizar a la población estudiantil de primer ingreso, recolectando información de 460 estudiantes de diversas facultades. Las variables incluidas abarcan datos sociodemográficos (facultad, edad, sexo, estado civil, jornada de estudio, año de ingreso), antropométricos (peso, talla, IMC), clínicos (presión arterial) y de hábitos (tabaquismo, consumo de alcohol, actividad física, entre otros). Esta base de datos, disponible en formato Excel y CSV, constituye un recurso ideal para ilustrar las capacidades de visualización de ggplot2 en el análisis estadístico aplicado a datos reales (Wickham, 2016).

Antes de iniciar cualquier análisis, es fundamental importar correctamente la base de datos y asegurarse de que las variables tengan el tipo de dato adecuado. Se recomienda guardar el archivo en la carpeta de trabajo del proyecto y utilizar los siguientes comandos para su importación y verificación:

```
# Instalar y cargar los paquetes necesarios
if (!require("tidyverse")) install.packages("tidyverse")
if (!require("readxl")) install.packages("readxl")


# Importar la base de datos USAC 2002
USAC2002 <- read_excel("Base_de_datos_USAC_2002.xlsx", sheet = "DATOS")

# Comprobar la estructura de las variables
str(USAC2002)
```

```
tibble [460 x 16] (S3: tbl_df/tbl/data.frame)
 $ FACULTAD   : chr [1:460] "Admon." "Admon." "Admon." "Admon." ...
 $ EDAD       : num [1:460] 21 21 31 38 24 34 34 22 42 39 ...
 $ SEXO       : chr [1:460] "M" "M" "F" "M" ...
 $ EST_CIVIL  : chr [1:460] "soltero" "soltero" "casado" "unido" ...
 $ TRABAJA    : num [1:460] 2 1 1 1 1 2 1 1 1 1 ...
 $ JORNADA    : chr [1:460] "vespertina" "vespertina" "nocturna" "nocturna" ...
 $ AÑO_ING    : num [1:460] 1998 1998 1991 1992 1995 ...
 $ PESO_lbs   : num [1:460] 150 150 197 120 175 156 190 137 198 150 ...
 $ PESO_kg    : num [1:460] 68.2 68.2 89.5 54.5 79.5 ...
 $ TALLA      : num [1:460] 1.7 1.72 1.55 1.7 1.7 1.7 1.74 1.71 1.82 1.68 ...
 $ IMC        : num [1:460] 23.6 23 37.3 18.9 27.5 ...
```

```
$ OBESIDAD   : num [1:460] 2 2 1 2 1 2 1 2 1 2 ...
$ Fuma       : chr [1:460] "2" "2" "1" "2" ...
$ Alcohol    : num [1:460] 2 1 2 2 2 2 1 1 2 1 ...
$ IMC_Clase  : chr [1:460] "NORMAL" "NORMAL" "OBESO" "NORMAL" ...
$ Grupos_Edad: chr [1:460] "Adulto_joven" "Adulto_joven" "Adulto_joven" "Adulto_maduro"
```

Este paso permite identificar si existen variables que requieren ser convertidas a factores (por ejemplo, SEXO, FACULTAD, JORNADA) o si es necesario ajustar nombres para facilitar su uso en R. La correcta preparación de los datos es esencial para obtener visualizaciones precisas y significativas (Wickham, 2016).

14.2 Introducción al paquete ggplot2

ggplot2 es un paquete de R que se utiliza para crear gráficos estadísticos de alta calidad de manera sencilla y flexible. Forma parte del conjunto de herramientas conocido como tidyverse, que está diseñado para facilitar el análisis y la visualización de datos. La principal característica de ggplot2 es que se basa en la "gramática de los gráficos", una idea desarrollada por Wilkinson (2005) y adaptada por Wickham (2016), que permite construir gráficos complejos a partir de piezas simples y combinables.

A diferencia de los gráficos base de R, donde cada tipo de gráfico tiene su propia función y la personalización puede ser complicada, ggplot2 utiliza una estructura modular. Esto significa que se puede empezar con un gráfico básico y, poco a poco, ir añadiendo o modificando elementos para adaptarlo a lo que se necesita. Así, se pueden crear gráficos claros, atractivos y personalizados para comunicar los resultados de un análisis de datos de forma efectiva (Wickham, 2016).

14.2.1 Ventajas principales de ggplot2

1. Permite crear muchos tipos de gráficos, como barras, líneas, puntos, histogramas y boxplots, entre otros.
2. Cada parte del gráfico se puede personalizar fácilmente: colores, títulos, etiquetas, escalas, temas y más.
3. Se integra muy bien con otras herramientas del tidyverse, lo que facilita trabajar con datos y visualizarlos en un solo flujo de trabajo.
4. Utiliza una lógica de "capas", lo que significa que se pueden añadir diferentes elementos (como puntos, líneas o etiquetas) uno sobre otro, de manera ordenada y controlada.

14.2.2 ¿Cómo funciona la gramática de los gráficos?

La gramática de los gráficos es como una receta que indica qué ingredientes debe tener un gráfico y cómo combinarlos. En ggplot2, cada gráfico se construye a partir de varios componentes básicos (Wickham, 2016; Wilkinson, 2005):

1. **Datos**: Es el conjunto de información que se quiere visualizar, normalmente en forma de tabla o data frame.
2. **Mapeos estéticos (aes)**: Son las instrucciones que indican cómo se relacionan las variables de los datos con los elementos visuales del gráfico, como la posición en los ejes, el color o el tamaño de los puntos.
3. **Geometrías (geoms)**: Son las formas que se usan para mostrar los datos, por ejemplo, barras para un gráfico de barras, puntos para un gráfico de dispersión, o cajas para un boxplot.
4. **Escalas**: Permiten controlar cómo se muestran los valores en el gráfico, por ejemplo, los colores, los tamaños o los intervalos de los ejes.
5. **Sistemas de coordenadas**: Determinan el tipo de espacio en el que se dibuja el gráfico, como el sistema cartesiano (el más común) o el sistema polar (para gráficos circulares).
6. **Facetas**: Sirven para dividir el gráfico en varios paneles, mostrando diferentes grupos de datos uno al lado del otro, lo que facilita la comparación entre categorías.
7. **Temas**: Permiten cambiar el aspecto general del gráfico, como el fondo, los textos y las líneas de cuadrícula, para que el resultado sea más claro y profesional.

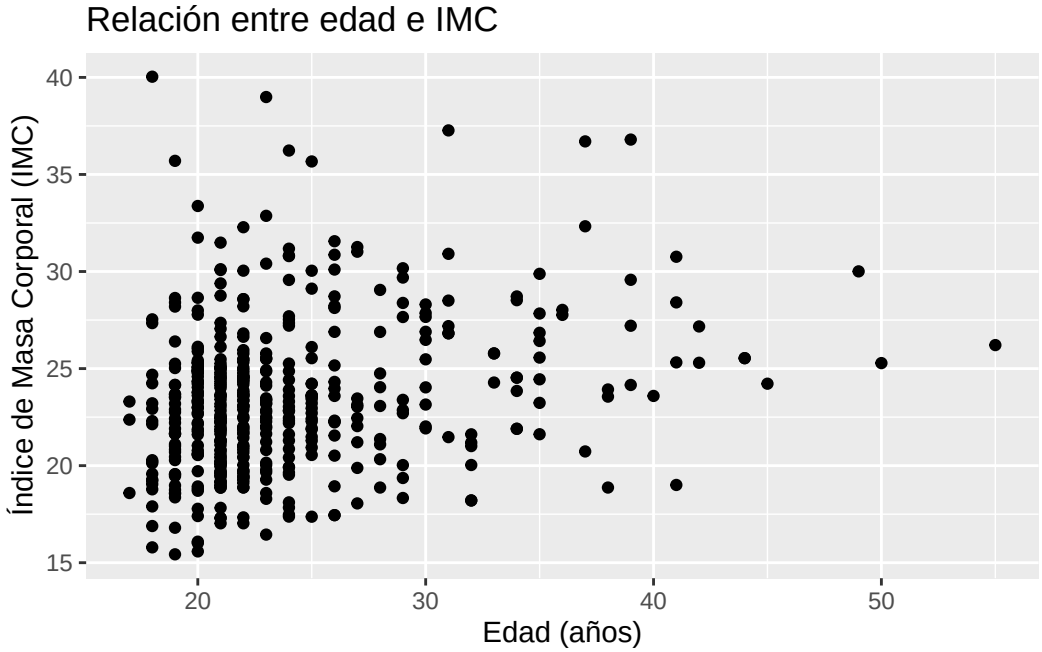
14.3 Estructura básica de un gráfico en ggplot2

La construcción de un gráfico en ggplot2 sigue una lógica de capas, donde cada componente se añade mediante el operador `+`. El proceso básico incluye:

1. **Iniciar el objeto gráfico** con la función `ggplot()`, especificando el conjunto de datos y los mapeos estéticos principales mediante `aes()`. Por ejemplo, se puede representar la EDAD en el eje X y el IMC en el eje Y.
2. **Añadir una o más capas geométricas**, como `geom_point()` para puntos, `geom_histogram()` para histogramas, o `geom_boxplot()` para diagramas de caja.
3. **Incorporar escalas** para controlar la interpretación de los valores, como escalas de color o tamaño.
4. **Añadir etiquetas y títulos** con `labs()` o `ggtitle()`, y modificar la apariencia general del gráfico con funciones de tema como `theme_minimal()`.
5. **Opcionalmente, añadir facetas** para dividir el gráfico en paneles según una variable categórica, como SEXO o JORNADA.

Ejemplo básico:

```
ggplot(data = USAC2002, aes(x = EDAD, y = IMC)) +
  geom_point() +
  labs(title = "Relación entre edad e IMC",
       x = "Edad (años)",
       y = "Índice de Masa Corporal (IMC)")
```



En este ejemplo, cada línea añade un componente al gráfico. El mapeo estético `aes(x = EDAD, y = IMC)` define qué variables se representan en los ejes, mientras que `geom_point()` indica que se utilizarán puntos para visualizar la relación.

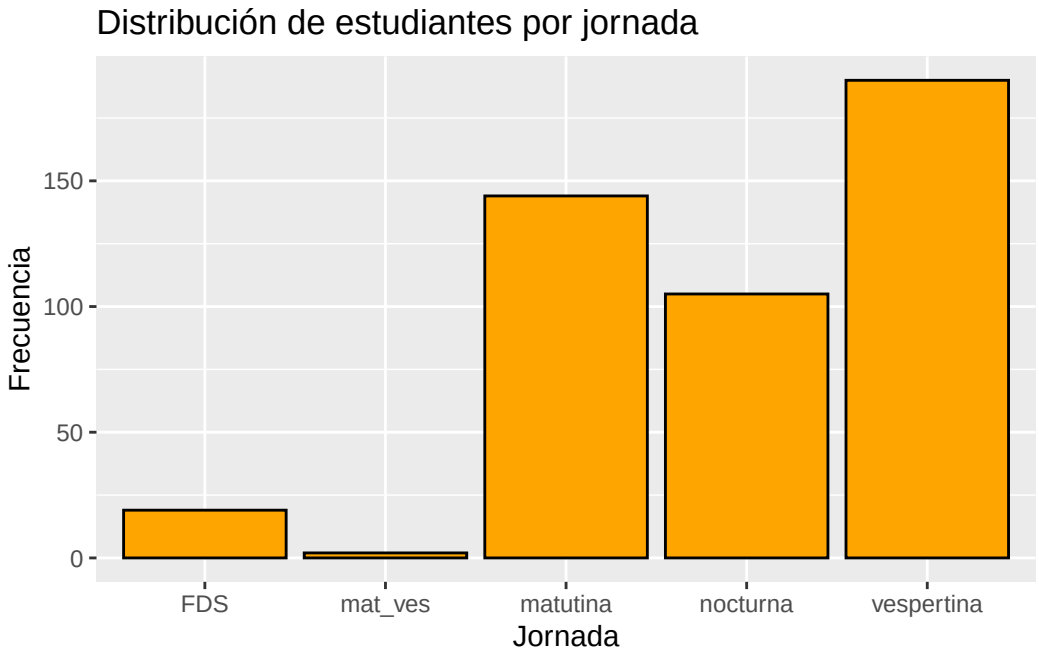
14.4 Creación de gráficos exploratorios y descriptivos

En el análisis de datos, la visualización inicial es clave para comprender la estructura y las características principales de las variables. ggplot2 permite construir de manera eficiente los gráficos más utilizados en la exploración y descripción de datos, facilitando la identificación de patrones, tendencias y diferencias entre grupos (Wickham, 2016).

14.4.1 Gráficos de barras para variables categóricas

El gráfico de barras es una herramienta fundamental para mostrar la cantidad de observaciones en cada categoría de una variable cualitativa. En ggplot2, este tipo de gráfico se genera de forma automática a partir de los datos, permitiendo comparar visualmente la frecuencia de cada grupo.

```r
# Crear gráfico de barras para la variable JORNADA
# Este gráfico muestra la distribución de estudiantes en las diferentes jornadas

ggplot(data = USAC2002,                    # Especificamos la base de datos
       aes(x = JORNADA)) +                 # Variable categórica en el eje X
  geom_bar(fill = "orange",                # Color de relleno de las barras
           color = "black") +              # Color del borde de las barras
  labs(title = "Distribución de estudiantes por jornada",  # Título del gráfico
       x = "Jornada",                      # Etiqueta del eje X
       y = "Frecuencia")                   # Etiqueta del eje Y
```



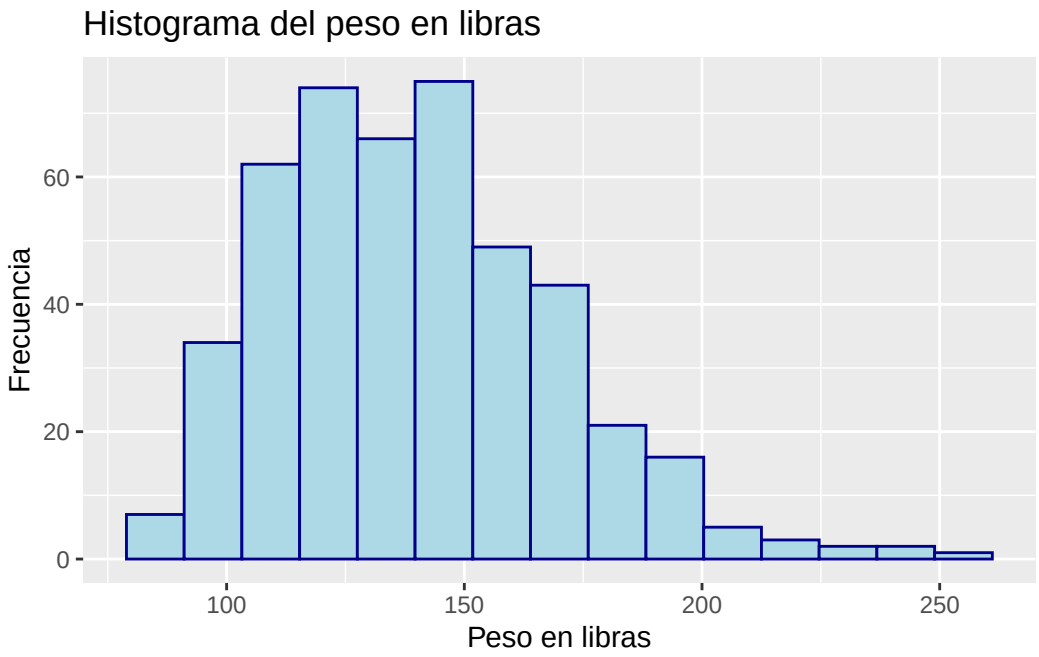
Explicación del código:

1. La función `ggplot()` inicia la construcción del gráfico especificando los datos y el mapeo estético.
2. `geom_bar()` crea automáticamente las barras contando las observaciones en cada categoría.
3. Los argumentos `fill` y `color` personalizan la apariencia de las barras.
4. La función `labs()` añade las etiquetas necesarias para la interpretación del gráfico.

14.4.2 Histogramas para variables continuas

El histograma es el gráfico más adecuado para examinar la distribución de una variable numérica. Permite observar la forma general de los datos, la presencia de asimetrías y la existencia de valores extremos.

```r
# Crear histograma para la variable PESO_lbs
# Este gráfico muestra la distribución del peso de los estudiantes

ggplot(data = USAC2002,                      # Especificamos la base de datos
       aes(x = PESO_lbs)) +                  # Variable numérica en el eje X
  geom_histogram(bins = 15,                  # Número de intervalos
                 fill = "lightblue",         # Color de relleno de las barras
                 color = "darkblue") +       # Color del borde de las barras
  labs(title = "Histograma del peso en libras",  # Título del gráfico
       x = "Peso en libras",                 # Etiqueta del eje X
       y = "Frecuencia")                     # Etiqueta del eje Y
```



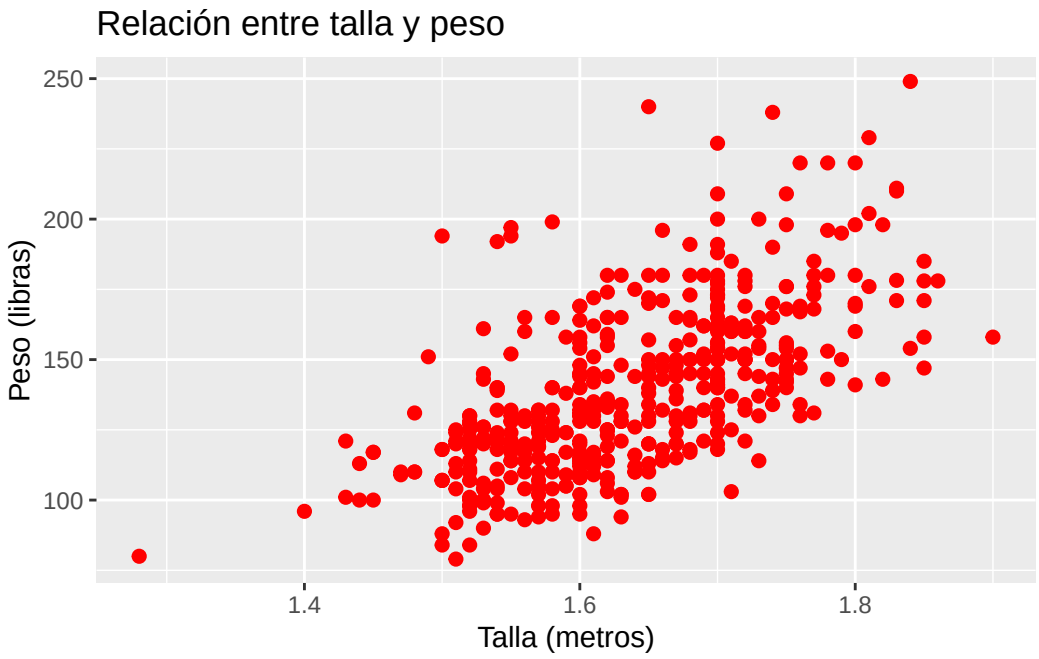
Explicación del código:

1. El parámetro `bins` determina el número de intervalos en que se dividirán los datos.
2. Los colores se eligen para contrastar el relleno con el borde de las barras.
3. Las etiquetas proporcionan contexto sobre la variable analizada.

14.4.3 Gráficos de dispersión para relaciones entre variables numéricas

El gráfico de dispersión es la opción principal para explorar la relación entre dos variables cuantitativas. Cada punto representa una observación, ubicándose según sus valores en los ejes X e Y.

```r
# Crear gráfico de dispersión para TALLA vs PESO_lbs
# Este gráfico muestra la relación entre la talla y el peso de los estudiantes

ggplot(data = USAC2002,                    # Especificamos la base de datos
       aes(x = TALLA,                      # Variable numérica en el eje X
           y = PESO_lbs)) +                # Variable numérica en el eje Y
  geom_point(color = "red",                # Color de los puntos
             size = 2) +                   # Tamaño de los puntos
  labs(title = "Relación entre talla y peso",  # Título del gráfico
       x = "Talla (metros)",               # Etiqueta del eje X
       y = "Peso (libras)")                # Etiqueta del eje Y
```



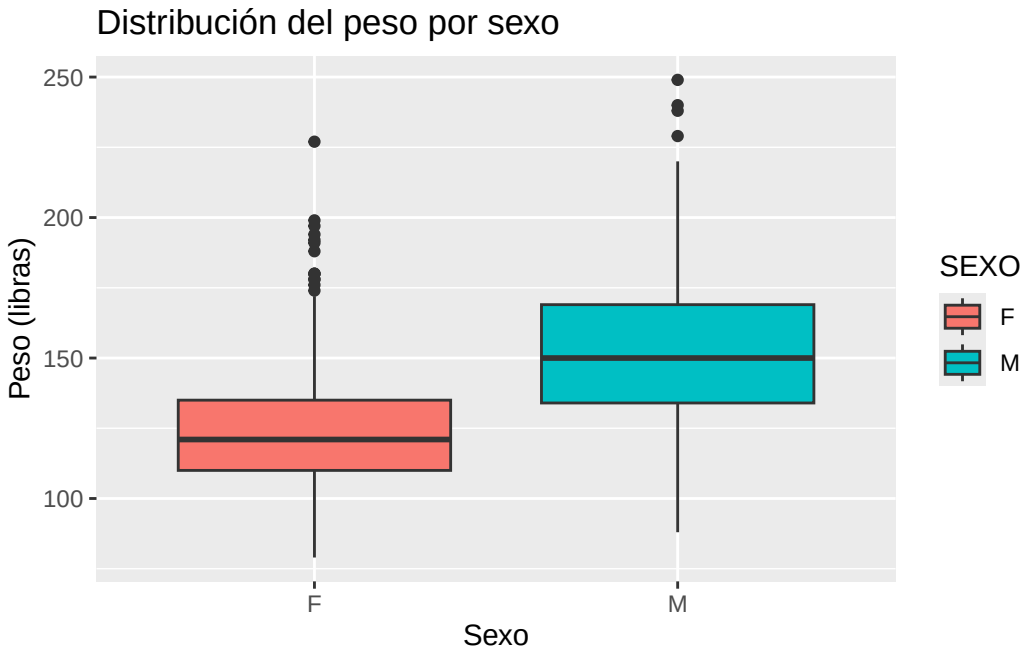
Explicación del código:

1. `geom_point()` crea un punto por cada par de valores (TALLA, PESO_lbs).
2. El argumento `size` controla el tamaño de los puntos para mejorar su visibilidad.
3. Las etiquetas incluyen las unidades de medida para mayor claridad.

14.4.4 Boxplots para comparación de grupos

El boxplot es un gráfico que resume la distribución de una variable numérica y facilita la comparación entre diferentes grupos definidos por una variable categórica (Wickham, 2016).

```r
# Crear boxplot para PESO_lbs por SEXO
# Este gráfico compara la distribución del peso entre hombres y mujeres

ggplot(data = USAC2002,                    # Especificamos la base de datos
       aes(x = SEXO,                       # Variable categórica en el eje X
           y = PESO_lbs,                   # Variable numérica en el eje Y
           fill = SEXO)) +                 # Color según el sexo
  geom_boxplot() +                         # Crear el boxplot
  labs(title = "Distribución del peso por sexo",  # Título del gráfico
       x = "Sexo",                         # Etiqueta del eje X
       y = "Peso (libras)")               # Etiqueta del eje Y
```



Explicación del código:

1. El mapeo estético `fill = SEXO` asigna automáticamente diferentes colores a cada grupo.
2. `geom_boxplot()` crea las cajas que muestran la distribución de cada grupo.
3. Las etiquetas ayudan a interpretar la comparación entre grupos.

Estos gráficos constituyen la base del análisis exploratorio y descriptivo en R con ggplot2, permitiendo obtener una visión clara y rápida de los datos antes de aplicar técnicas estadísticas más avanzadas (Wickham, 2016).

14.5 Personalización de gráficos en ggplot2

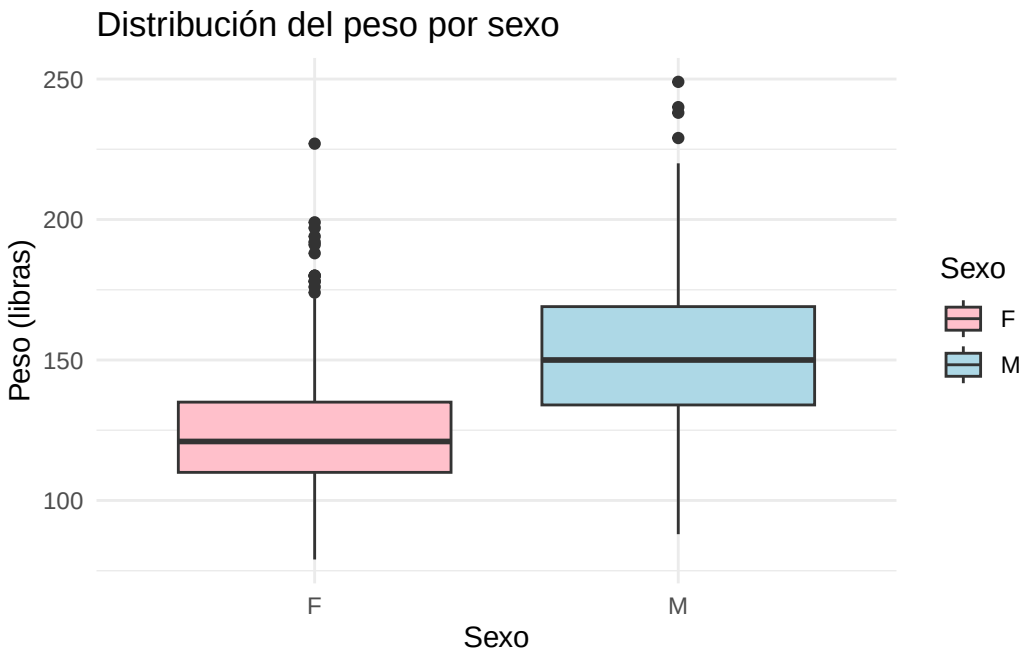
La personalización de gráficos es un aspecto fundamental para comunicar efectivamente los resultados de un análisis estadístico. ggplot2 ofrece una amplia gama de opciones para

adaptar cada elemento visual según las necesidades específicas del usuario y el contexto de presentación (Wickham, 2016).

14.5.1 Modificación de colores y escalas

La elección adecuada de colores puede mejorar significativamente la interpretación de un gráfico. ggplot2 permite personalizar los colores tanto de manera directa como a través de escalas predefinidas o personalizadas.

```
# Ejemplo de personalización de colores en un boxplot
# Este gráfico compara el peso entre sexos con colores específicos

ggplot(data = USAC2002,                          # Especificamos la base de datos
       aes(x = SEXO,                             # Variable categórica en eje X
           y = PESO_lbs,                         # Variable numérica en eje Y
           fill = SEXO)) +                       # Color de relleno según sexo
  geom_boxplot() +                               # Crear boxplot
  scale_fill_manual(                             # Personalizar colores de relleno
    values = c("pink", "lightblue")) +           # Asignar colores específicos
  labs(title = "Distribución del peso por sexo",  # Título principal
       x = "Sexo",                               # Etiqueta eje X
       y = "Peso (libras)",                      # Etiqueta eje Y
       fill = "Sexo") +                          # Título de la leyenda
  theme_minimal()                                # Tema minimalista
```



Explicación del código:

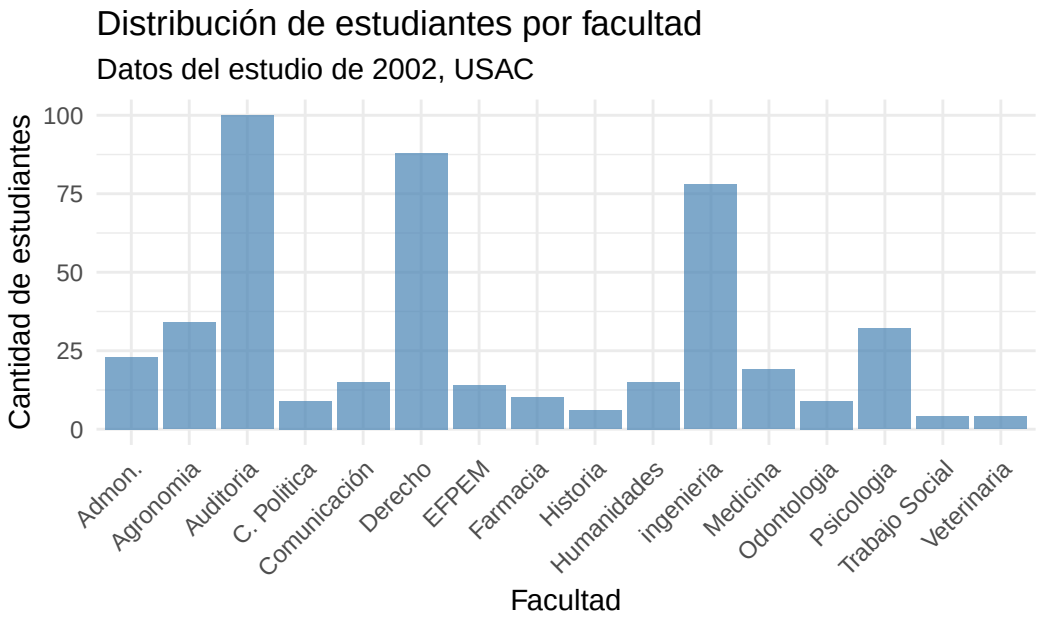
1. La función `scale_fill_manual()` permite asignar colores específicos a cada categoría.

2. Los colores se eligen para maximizar el contraste y la legibilidad.
3. El argumento `fill` en `labs()` personaliza el título de la leyenda.

14.5.2 Etiquetas, títulos y leyendas

Las etiquetas y títulos son esenciales para proporcionar contexto y facilitar la interpretación del gráfico. ggplot2 ofrece múltiples opciones para personalizar estos elementos.

```
# Ejemplo de personalización completa de etiquetas
# Este gráfico incluye título, subtítulo y nota al pie

ggplot(data = USAC2002,                      # Especificamos la base de datos
       aes(x = FACULTAD)) +                  # Variable categórica en eje X
  geom_bar(fill = "steelblue",              # Color de las barras
           alpha = 0.7) +                    # Transparencia de las barras
  labs(
    title = "Distribución de estudiantes por facultad",     # Título principal
    subtitle = "Datos del estudio de 2002, USAC",          # Subtítulo
    x = "Facultad",                                        # Etiqueta eje X
    y = "Cantidad de estudiantes",                         # Etiqueta eje Y
    caption = "Fuente: Estudio realizado en 2002"         # Nota al pie
  ) +
  theme_minimal() +                          # Tema base minimalista
  theme(
    axis.text.x = element_text(            # Personalizar texto eje X
      angle = 45,                          # Rotar texto 45 grados
      hjust = 1                           # Alinear texto
    )
  )
```



Fuente: Estudio realizado en 2002

Explicación del código:

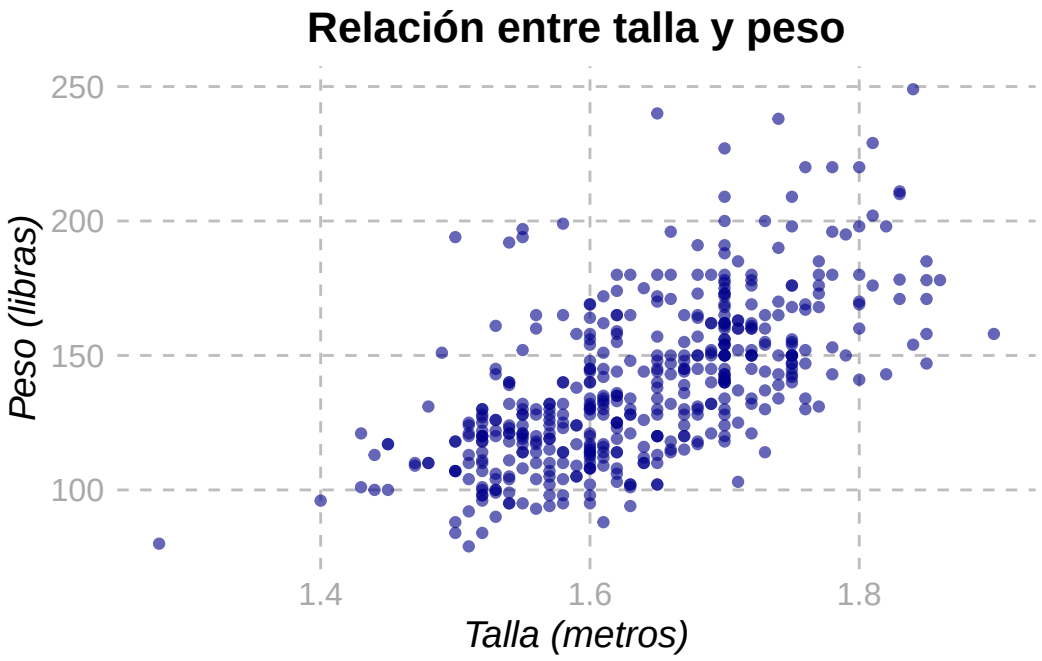
1. La función `labs()` permite añadir múltiples elementos informativos.
2. El argumento `alpha` controla la transparencia de las barras.
3. La rotación del texto en el eje X mejora la legibilidad cuando hay muchas categorías.

14.5.3 Aplicación y personalización de temas

Los temas en ggplot2 permiten controlar la apariencia general del gráfico, desde el fondo hasta los elementos más pequeños. La personalización puede ser global o específica para cada elemento (Wickham, 2016).

```
# Ejemplo de personalización avanzada de tema
# Este gráfico muestra múltiples personalizaciones de elementos visuales

ggplot(data = USAC2002,                          # Especificamos la base de datos
       aes(x = TALLA, y = PESO_lbs)) +           # Variables en ejes X e Y
  geom_point(color = "darkblue",                 # Color de los puntos
             alpha = 0.6) +                      # Transparencia de los puntos
  theme_minimal() +                              # Tema base minimalista
  theme(
    # Personalización del título
    plot.title = element_text(
      hjust = 0.5,                              # Centrar título
      size = 16,                                # Tamaño de fuente
      face = "bold"                             # Texto en negrita
    ),
```

```r
  # Personalización del texto de los ejes
  axis.text = element_text(
    size = 12,                          # Tamaño de fuente
    color = "darkgray"                  # Color del texto
  ),
  # Personalización de títulos de ejes
  axis.title = element_text(
    size = 14,                          # Tamaño de fuente
    face = "italic"                     # Texto en cursiva
  ),
  # Personalización de la cuadrícula
  panel.grid.major = element_line(
    color = "gray",                     # Color de líneas principales
    linetype = "dashed"                 # Tipo de línea
  ),
  panel.grid.minor = element_blank()    # Eliminar líneas secundarias
) +
labs(title = "Relación entre talla y peso",
     x = "Talla (metros)",
     y = "Peso (libras)")
```



Explicación del código:

1. `theme_minimal()` establece un tema base limpio y profesional.
2. `element_text()` permite personalizar todos los elementos de texto.
3. `element_line()` controla la apariencia de las líneas de la cuadrícula.
4. `element_blank()` elimina elementos no deseados.

La personalización adecuada de un gráfico puede mejorar significativamente su capacidad para comunicar información, haciendo que los datos sean más accesibles y comprensibles para diferentes audiencias (Wickham, 2016).

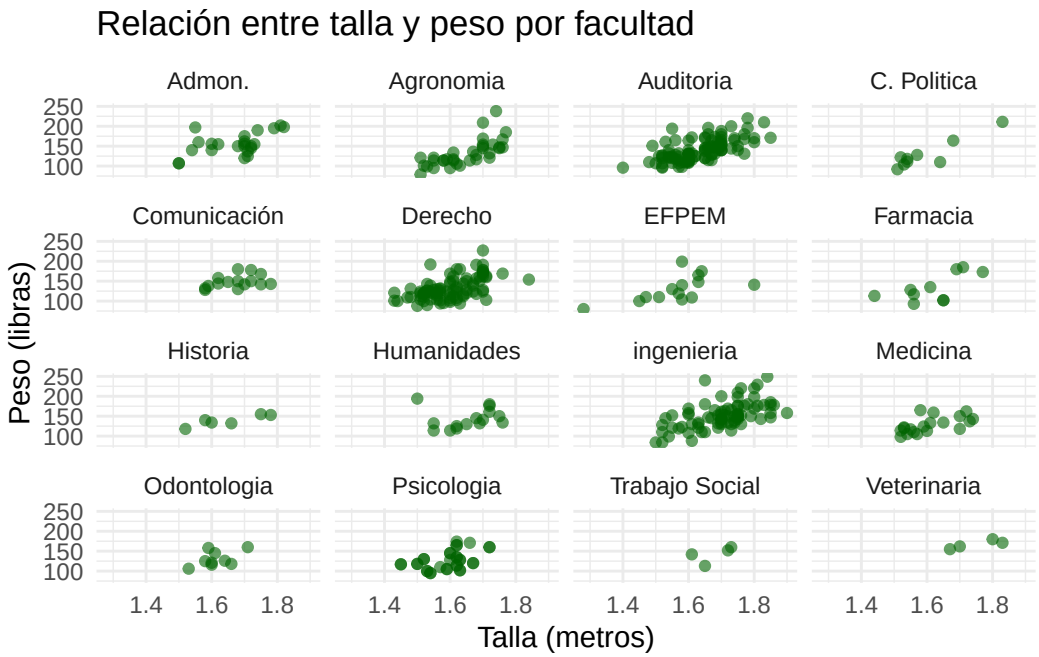
14.6 Uso de facetas para comparación de grupos

Las facetas en ggplot2 son una herramienta poderosa para dividir un gráfico en varios subgráficos, cada uno correspondiente a un grupo definido por una o más variables categóricas. Esto facilita la comparación visual entre diferentes segmentos de los datos, permitiendo identificar patrones, similitudes o diferencias que podrían pasar desapercibidas en un solo gráfico global (Wickham, 2016).

14.6.1 Facet_wrap y facet_grid: sintaxis y aplicaciones

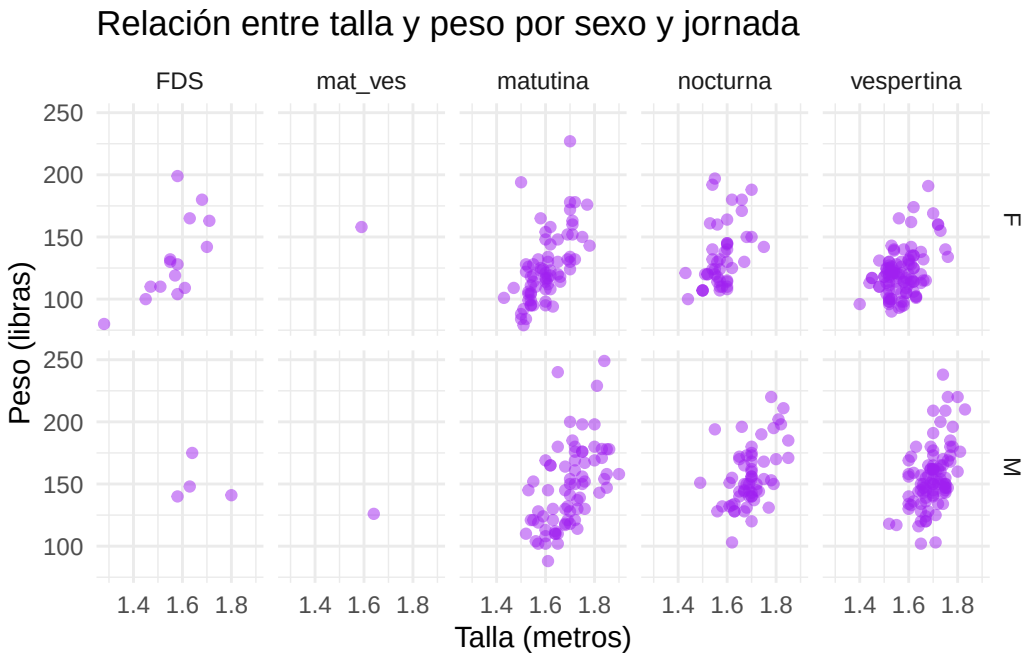
ggplot2 ofrece dos funciones principales para crear facetas: `facet_wrap()` y `facet_grid()`. Cada una tiene una lógica y utilidad específica.

```
# Ejemplo 1: Uso de facet_wrap para comparar por facultad
# Este gráfico muestra la relación entre talla y peso, generando un subgráfico para cada

ggplot(data = USAC2002, aes(x = TALLA, y = PESO_lbs)) +  # Variables numéricas en los eje
  geom_point(color = "darkgreen", alpha = 0.6) +         # Puntos verdes con transparenci
  facet_wrap(~ FACULTAD) +                               # Un panel por cada facultad
  labs(
    title = "Relación entre talla y peso por facultad",   # Título principal
    x = "Talla (metros)",                                # Etiqueta eje X
    y = "Peso (libras)"                                  # Etiqueta eje Y
  ) +
  theme_minimal()                                        # Tema limpio y profesional
```



1. `facet_wrap(~ FACULTAD)` divide el gráfico en tantos paneles como valores únicos tenga la variable FACULTAD, permitiendo comparar la relación entre talla y peso en cada facultad de manera individual.
2. Es útil cuando se desea comparar un solo criterio de agrupación y se prefiere que los paneles se organicen en una cuadrícula flexible.

```
# Ejemplo 2: Uso de facet_grid para comparar por sexo y jornada
# Este gráfico muestra la relación entre talla y peso, generando una matriz de subgráfico

ggplot(data = USAC2002, aes(x = TALLA, y = PESO_lbs)) +  # Variables numéricas en los eje
  geom_point(color = "purple", alpha = 0.5) +             # Puntos morados con transparen
  facet_grid(SEXO ~ JORNADA) +                            # Filas por SEXO, columnas por
  labs(
    title = "Relación entre talla y peso por sexo y jornada",  # Título principal
    x = "Talla (metros)",                                      # Etiqueta eje X
    y = "Peso (libras)"                                        # Etiqueta eje Y
  ) +
  theme_minimal()                                           # Tema limpio y profesional
```



1. `facet_grid(SEXO ~ JORNADA)` crea una matriz de paneles, donde las filas corresponden a los niveles de SEXO y las columnas a los niveles de JORNADA.
2. Es especialmente útil para comparar dos criterios de agrupación de manera cruzada y ordenada.

14.6.2 Cuadro resumen de diferencias de las funciones face_wrap y face_grid

Función	Uso principal	Organización de paneles	Variables categóricas involucradas	Ejemplo de sintaxis
`facet_wrap()`	Comparar grupos definidos por una sola variable	Cuadrícula flexible (ajuste automático)	Una variable categórica	`facet_wrap(~ FACULTAD)`
`facet_grid()`	Comparar grupos definidos por dos variables	Matriz (filas y columnas fijas)	Dos variables categóricas (filas y columnas)	`facet_grid(SEXO ~ JORNADA)`

14.7 Comparación entre ggplot2 y el sistema gráfico base de R

Antes de elegir una herramienta de visualización en R, es importante conocer las diferencias clave entre **ggplot2** y el sistema gráfico base. Cada uno tiene ventajas y limitaciones que

pueden hacerlos más adecuados según el contexto y los objetivos del análisis (Wickham, 2016).

Característica	ggplot2	Sistema gráfico base de R
Enfoque	Modular, basado en la gramática de los gráficos	Funciones específicas para cada gráfico
Personalización	Avanzada y flexible	Limitada y menos intuitiva
Sintaxis	Declarativa y estructurada	Imperativa y secuencial
Integración con tidyverse	Total	Parcial o nula
Curva de aprendizaje	Inicialmente más alta	Más baja para gráficos simples
Ideal para	Informes profesionales, gráficos complejos	Exploración rápida, gráficos sencillos

En resumen, **ggplot2** es preferible cuando se requiere personalización, reproducibilidad y presentación profesional, mientras que el sistema gráfico base resulta útil para análisis exploratorios rápidos o cuando se necesita generar gráficos simples con poco código. La elección depende de las necesidades del usuario y del contexto del análisis (Wickham, 2016).

Parte V

Gestión y Exportación de resultados

15 Introducción a la Gestión de Proyectos en R

La gestión de proyectos en R es esencial para mantener el orden, la claridad y la eficiencia en el análisis estadístico de datos, incluso en proyectos sencillos. Adoptar buenas prácticas desde el inicio permite evitar errores, facilita la revisión del trabajo y mejora la comunicación de los resultados, tanto para el propio usuario como para otros que puedan consultar el proyecto en el futuro (Wickham & Grolemund, 2017).

15.1 Importancia de la gestión de proyectos en análisis estadístico

En el análisis estadístico, la gestión de proyectos consiste en organizar todos los elementos necesarios para el trabajo en un solo lugar. Esto incluye los datos, los scripts de análisis, los resultados exportados y cualquier archivo adicional relevante. Mantener todos estos archivos juntos en una carpeta específica para cada proyecto ayuda a evitar confusiones, pérdidas de información y errores al ejecutar los análisis.

La gestión adecuada de proyectos permite:

1. Retomar el trabajo en cualquier momento sin perder el contexto.
2. Compartir el proyecto con otras personas de manera sencilla.
3. Garantizar que los resultados obtenidos sean reproducibles y verificables.
4. Evitar la mezcla de archivos de diferentes análisis, lo que puede llevar a errores o a la utilización de datos incorrectos.

En proyectos simples, donde el análisis se limita a un solo conjunto de datos y un flujo de trabajo lineal, una carpeta por proyecto es suficiente para mantener el orden y la trazabilidad del trabajo (Wickham & Grolemund, 2017).

15.2 Organización de archivos en proyectos de R

Para proyectos de análisis estadístico simples, se recomienda crear una carpeta exclusiva para cada proyecto. Dentro de esta carpeta deben almacenarse todos los archivos relacionados con el análisis, lo que incluye:

1. El archivo de datos (por ejemplo, un archivo CSV o Excel).

2. El archivo del proyecto de RStudio (con extensión `.Rproj`), que facilita la gestión y el acceso al proyecto.
3. El script de análisis en R (por ejemplo, `analisis.R`), donde se escribe y ejecuta el código.
4. Los resultados exportados, como gráficos en formato PNG o PDF y tablas en formato CSV o Excel.

Esta organización básica permite que, al abrir la carpeta del proyecto, se tenga acceso inmediato a todos los elementos necesarios para reproducir el análisis o continuar trabajando. Además, facilita la identificación de los archivos y su propósito, evitando la dispersión de información.

15.3 Uso de RStudio Projects para la gestión eficiente

RStudio Projects es una herramienta integrada en el entorno RStudio que permite gestionar proyectos de manera eficiente, incluso en análisis simples. Al crear un proyecto en RStudio, se genera un archivo con extensión `.Rproj` dentro de la carpeta del proyecto. Este archivo define el directorio de trabajo y centraliza todos los archivos relacionados.

Las ventajas de utilizar RStudio Projects en proyectos simples incluyen:

1. Facilita la apertura y cierre del proyecto, restaurando el entorno de trabajo tal como se dejó la última vez.
2. Asegura que el directorio de trabajo sea siempre el correcto, evitando errores al cargar o guardar archivos.
3. Permite mantener separados los análisis de diferentes proyectos, lo que reduce el riesgo de mezclar datos o resultados.

Para crear un proyecto en RStudio, se debe seleccionar la opción "File > New Project", elegir "New Directory" y luego "New Project". Se asigna un nombre y una ubicación a la carpeta del proyecto, y RStudio creará automáticamente el archivo `.Rproj` en esa carpeta. A partir de ese momento, todos los archivos del análisis deben guardarse en esa misma carpeta para mantener la organización y la reproducibilidad (Wickham & Grolemund, 2017).

15.4 Principios de reproducibilidad y documentación en proyectos de R

La reproducibilidad es un principio fundamental en el análisis estadístico. Consiste en la capacidad de repetir un análisis y obtener los mismos resultados, utilizando los mismos datos y scripts. Para lograrlo, es esencial mantener todos los archivos del proyecto juntos y documentar adecuadamente cada paso del proceso.

En proyectos simples, la reproducibilidad se puede asegurar mediante:

1. El uso de scripts bien comentados, donde se explique cada parte del análisis.
2. La inclusión de los datos originales en la carpeta del proyecto.
3. La exportación de los resultados (gráficos y tablas) en formatos accesibles y guardados en la misma carpeta.
4. El uso del archivo `.Rproj` para centralizar el entorno de trabajo.

Además, es recomendable agregar comentarios en el script de análisis que expliquen el propósito de cada sección del código, los pasos seguidos y cualquier decisión relevante tomada durante el análisis. Esta documentación facilita la revisión, el aprendizaje y la colaboración, incluso en proyectos individuales (Wickham & Grolemund, 2017).

16 Exportación de Resultados de Análisis en R

La exportación de resultados constituye una etapa fundamental en el análisis estadístico de datos, ya que permite almacenar y compartir los productos del análisis, como gráficos y tablas, para su posterior utilización en informes, presentaciones o análisis adicionales. La correcta elección del formato de exportación garantiza la accesibilidad, reutilización y compatibilidad de los resultados con otras herramientas y plataformas (R Core Team, 2023; Wickham, 2016).

En el contexto del análisis estadístico clásico, la exportación de resultados facilita la comunicación de hallazgos y la integración de los mismos en documentos científicos, reportes técnicos o presentaciones. R ofrece funciones específicas para exportar tanto gráficos como tablas de datos en los formatos más utilizados en la práctica profesional y académica, asegurando la calidad y la fidelidad de la información exportada (R Core Team, 2023).

16.1 Exportación de gráficos: formatos PNG y PDF

La exportación de gráficos es fundamental para documentar visualmente los resultados del análisis. En R, la función `ggsave()` del paquete ggplot2 permite guardar gráficos en diversos formatos, siendo PNG y PDF los más empleados en la estadística clásica.

16.1.1 Sintaxis general de ggsave()

La función `ggsave()` del paquete ggplot2 permite guardar gráficos en diferentes formatos. Su sintaxis básica es:

```
ggsave(
  filename,
  plot = last_plot(),
  device = NULL,
  path = NULL,
  scale = 1,
  width = NA,
  height = NA,
  units = c("in", "cm", "mm"),
  dpi = 300,
  limitsize = TRUE
)
```

A continuación, se describen los argumentos principales de la función:

1. **filename**: Es el nombre del archivo de salida, incluyendo la extensión (por ejemplo, `"grafico.png"` o `"grafico.pdf"`). La extensión determina el formato del archivo.
2. **plot**: Permite especificar el objeto gráfico que se desea guardar. Si se omite, se guarda el último gráfico creado en la sesión de R.
3. **device**: Indica el tipo de formato del archivo, como `"png"` o `"pdf"`. Si no se especifica, el formato se deduce automáticamente a partir de la extensión del archivo.
4. **path**: Define el directorio donde se guardará el archivo. Si no se proporciona, el archivo se guarda en el directorio de trabajo actual.
5. **scale**: Ajusta el tamaño del gráfico multiplicando las dimensiones especificadas en width y height por el valor de scale. El valor predeterminado es 1 (tamaño original).
6. **width y height**: Determinan el ancho y la altura del gráfico en las unidades especificadas por units. Si no se definen, se usan las dimensiones predeterminadas.
7. **units**: Especifica las unidades de medida para width y height. Puede ser "in" (pulgadas), "cm" (centímetros) o "mm" (milímetros).
8. **dpi**: Define la resolución del gráfico en puntos por pulgada, relevante para formatos rasterizados como PNG. El valor predeterminado es 300, adecuado para impresión.
9. **limitsize**: Controla si se permite guardar gráficos con dimensiones muy grandes (mayores a 50 pulgadas). Si está en TRUE, se genera un error al intentar guardar gráficos excesivamente grandes.

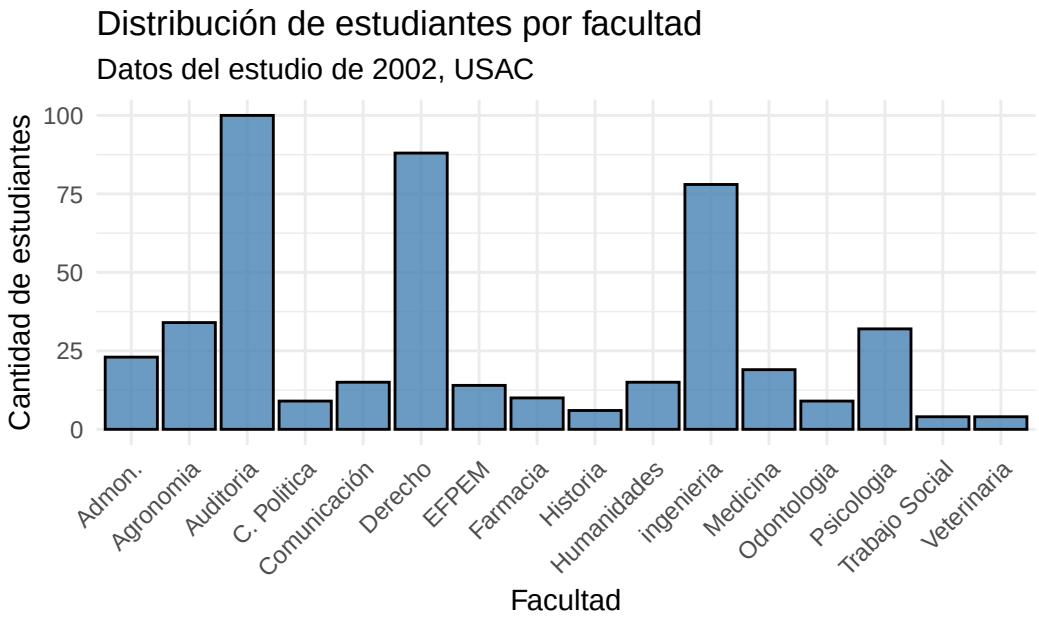
Esta explicación permite comprender tanto la estructura general de la función como el propósito de cada argumento, facilitando su uso correcto en la exportación de gráficos en R (Wickham, 2016).

16.1.2 Ejemplo práctico: creación y exportación de un gráfico

Supóngase que se ha creado un gráfico de barras con ggplot2:

```
# Cargar el paquete tidyverse, que incluye ggplot2
if (!require("tidyverse")) install.packages("tidyverse")
library(tidyverse)

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

# Crear un gráfico de barras
mi_grafico <- 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(
    axis.text.x = element_text(
      angle = 45,
      hjust = 1
    )
  )
mi_grafico
```



Guardar el gráfico en formato PNG

```
# Guardar el gráfico en formato PNG con dimensiones de 8x6 pulgadas
ggsave(
  filename = "grafico.png", # Nombre del archivo de salida
  plot = mi_grafico,        # Objeto gráfico a guardar
  width = 8,                # Ancho en pulgadas
  height = 6,               # Alto en pulgadas
  dpi = 300                 # Resolución adecuada para impresión
)
```

En este ejemplo, el archivo “grafico.png” se guardará en el directorio de trabajo actual, con alta calidad para impresión o presentaciones digitales.

Guardar el gráfico en formato PDF

```r
# Guardar el gráfico en formato PDF con dimensiones de 8x6 pulgadas
ggsave(
  filename = "grafico.pdf", # Nombre del archivo de salida
  plot = mi_grafico,        # Objeto gráfico a guardar
  width = 8,                # Ancho en pulgadas
  height = 6                # Alto en pulgadas
  # No es necesario especificar dpi, ya que PDF es un formato vectorial
)
```

El formato PDF es ideal para informes y publicaciones científicas, ya que permite escalar el gráfico sin pérdida de calidad (Wickham, 2016).

16.2 Exportación de tablas de datos: formatos CSV y Excel

La exportación de tablas de datos es fundamental para compartir información, documentar resultados o realizar análisis adicionales en otras herramientas. Los formatos más utilizados en la estadística clásica son CSV y Excel, por su compatibilidad y facilidad de uso.

16.2.1 Exportar a CSV con write.csv()

La función `write.csv()` permite exportar un data frame o matriz a un archivo de texto plano en formato CSV (Comma Separated Values). Este formato es ampliamente compatible con programas de hojas de cálculo y software estadístico.

Sintaxis general de `write.csv()`:

```r
write.csv(
  x,
  file,
  row.names = TRUE,
  na = "NA",
  fileEncoding = "",
)
```

Explicación de los argumentos principales:

1. **x**: Es el objeto de datos que se desea exportar, generalmente un data frame o una matriz.
2. **file**: Especifica el nombre del archivo de salida, incluyendo la extensión `.csv`. El archivo se guardará en el directorio de trabajo actual, a menos que se indique una ruta diferente.
3. **row.names**: Indica si se deben incluir los nombres de las filas como una columna adicional en el archivo exportado. El valor predeterminado es `TRUE`, pero es común establecerlo en `FALSE` para evitar agregar una columna innecesaria.

4. **na**: Define la cadena de texto que se utilizará para representar los valores faltantes (`NA`) en el archivo exportado. El valor predeterminado es `"NA"`.
5. **fileEncoding**: Permite especificar la codificación del archivo de salida, útil para asegurar la compatibilidad con otros sistemas operativos o programas. El valor predeterminado es una cadena vacía, lo que significa que se utiliza la codificación por defecto del sistema.

Ejemplo:

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

# Exportar el data frame a un archivo CSV
write.csv(
  x = mi_tabla,          # Objeto de datos a exportar
  file = "resultados.csv", # Nombre del archivo de salida
  row.names = FALSE      # No incluir los nombres de las filas
)
```

El archivo "resultados.csv" se guardará en el directorio de trabajo actual y podrá ser abierto en cualquier editor de texto o programa de hojas de cálculo (R Core Team, 2023).

16.2.2 Exportar a Excel con write_xlsx() del paquete writexl

La función `write_xlsx()` del paquete writexl permite exportar un data frame o una lista de data frames a un archivo en formato Excel (`.xlsx`). Este formato es ideal para compartir datos estructurados y aprovechar las funcionalidades avanzadas de hojas de cálculo.

Sintaxis general de `write_xlsx()`

```
write_xlsx(
  x,           # Objeto de datos a exportar
  path,        # Nombre del archivo de salida
  col_names = TRUE,
  format_headers = TRUE
)
```

Explicación de los argumentos principales:

1. **x**: Es el objeto de datos a exportar, que puede ser un data frame o una lista de data frames (en este caso, cada data frame se guardará en una hoja diferente del archivo Excel).

2. **path**: Especifica el nombre del archivo de salida, incluyendo la extensión `.xlsx`. El archivo se guardará en el directorio de trabajo actual, a menos que se indique una ruta diferente.
3. **col_names**: Indica si se deben incluir los nombres de las columnas en la primera fila del archivo. El valor predeterminado es `TRUE`.
4. **format_headers**: Determina si los encabezados de las columnas deben tener un formato especial (por ejemplo, negrita). El valor predeterminado es `TRUE`.

Ejemplo:

```
# Instalar y cargar el paquete writexl si no está disponible
if (!require("writexl")) install.packages("writexl")

# Exportar el data frame a un archivo Excel
write_xlsx(
  x = mi_tabla,            # Objeto de datos a exportar
  path = "resultados.xlsx" # Nombre del archivo de salida
  # col_names y format_headers se mantienen en TRUE por defecto
)
```

El archivo "resultados.xlsx" se podrá abrir en Microsoft Excel o software compatible, permitiendo aprovechar las funcionalidades avanzadas de hojas de cálculo (R Core Team, 2023).

16.3 Comparación de formatos y recomendaciones de uso

La elección del formato de exportación depende del objetivo y del público destinatario. A continuación se presenta un cuadro comparativo que distingue entre formatos de imágenes y de datos, resumiendo sus principales características, ventajas y desventajas (Wickham, 2016; R Core Team, 2023):

Tipo	Formato	Uso principal	Ventajas	Desventajas
Imagen	PNG	Presentaciones y documentos digitales	Alta calidad, ampliamente compatible	No escalable sin pérdida de calidad
Imagen	PDF	Publicaciones científicas e informes impresos	Escalable, ideal para impresión	Menos compatible con editores básicos
Datos	CSV	Análisis de datos en herramientas simples	Ligero, multi-plataforma, fácil de manipular	No admite formatos complejos (fórmulas, etc)

Tipo	Formato	Uso principal	Ventajas	Desventajas
Datos	Excel	Compartir datos estructurados y análisis	Compatible con herramientas avanzadas	Requiere software específico

17 Uso de Git y GitHub en Proyectos de R

El control de versiones y la colaboración en línea son prácticas cada vez más importantes en el análisis estadístico y la ciencia de datos. Git y GitHub permiten gestionar de manera eficiente los cambios en los archivos de un proyecto, compartir el trabajo con otros y mantener un historial completo de todas las modificaciones realizadas. Aunque estas herramientas pueden parecer complejas al principio, su integración con RStudio y su utilidad en proyectos de cualquier tamaño justifican su aprendizaje y uso desde etapas tempranas (Bryan, 2018).

17.1 Introducción al control de versiones y colaboración

El control de versiones es una metodología que permite registrar, organizar y recuperar los cambios realizados en los archivos de un proyecto a lo largo del tiempo. Git es el sistema de control de versiones más utilizado y se integra fácilmente con RStudio, lo que facilita su adopción en proyectos de análisis estadístico.

Ventajas del control de versiones con Git:

1. Permite guardar el historial de cambios, facilitando la recuperación de versiones anteriores de los archivos.
2. Ayuda a identificar cuándo, cómo y por qué se realizaron modificaciones, lo que mejora la trazabilidad y la transparencia.
3. Facilita la colaboración entre varios usuarios, permitiendo que cada uno trabaje en su propia copia del proyecto y luego integre los cambios.
4. Reduce el riesgo de pérdida de información, ya que los archivos pueden ser restaurados a cualquier estado anterior.
5. Permite experimentar con nuevas ideas sin temor a perder el trabajo anterior, gracias a la posibilidad de crear ramas (branches) y fusionarlas posteriormente.

GitHub es una plataforma en línea que permite alojar repositorios de Git, compartir proyectos y colaborar con otros usuarios. Además, ofrece herramientas para la gestión de proyectos, seguimiento de problemas (issues), revisión de código y documentación.

En el contexto de proyectos de R, Git y GitHub permiten mantener un registro ordenado de los scripts, datos y resultados, facilitando la colaboración y la reproducibilidad del análisis (Bryan, 2018).

17.2 Subida de un proyecto de R a GitHub

Subir un proyecto de R a GitHub implica crear un repositorio en la plataforma y sincronizarlo con la carpeta local del proyecto. Este proceso puede realizarse desde la interfaz de RStudio o utilizando la línea de comandos. A continuación se describe el proceso paso a paso para un usuario principiante:

1. Crear una cuenta en GitHub
Para comenzar, es necesario registrarse en https://github.com/ y crear una cuenta personal.

2. Crear un repositorio nuevo en GitHub
Una vez dentro de la cuenta, se debe hacer clic en el botón "New repository". Se recomienda asignar un nombre descriptivo al repositorio (por ejemplo, "analisis_estadistico") y, opcionalmente, agregar una breve descripción. Es posible elegir si el repositorio será público (visible para todos) o privado (solo accesible para el usuario y quienes él autorice). Al crear el repositorio, se puede dejar vacío, ya que los archivos se agregarán desde la computadora local.

3. Inicializar Git en la carpeta del proyecto local
En la computadora, se debe ubicar la carpeta del proyecto de R (la que contiene el archivo `.Rproj`, los datos, el script y los resultados exportados).

1. Si se utiliza RStudio, se puede activar el control de versiones seleccionando "Tools > Project Options > Git/SVN" y eligiendo Git.
2. Si se prefiere la terminal, se debe abrir una consola en la carpeta del proyecto y ejecutar el comando:

```
git init
```

Esto crea una carpeta oculta llamada `.git` que permitirá a Git rastrear los cambios en los archivos del proyecto.

4. Conectar el repositorio local con el remoto en GitHub

Para vincular la carpeta local con el repositorio creado en GitHub, se debe copiar la URL del repositorio (por ejemplo, `https://github.com/usuario/analisis_estadistico.git`) y ejecutar el siguiente comando en la terminal:

```
git remote add origin https://github.com/usuario/analisis_estadistico.git
```

5. Agregar y confirmar los archivos del proyecto

Se deben agregar los archivos del proyecto al control de versiones con el comando:

```
git add .
```

El punto (`.`) indica que se agregarán todos los archivos de la carpeta.

Luego, se realiza el primer "commit" (registro de cambios) con un mensaje descriptivo:

```
git commit -m "Primer commit: subida inicial del proyecto"
```

6. Subir los archivos a GitHub

Finalmente, se suben los archivos al repositorio remoto con el comando:

```
git push -u origin master
```

En algunos casos, la rama principal puede llamarse "main" en lugar de "master", por lo que el comando sería:

```
git push -u origin main
```

Una vez completados estos pasos, el proyecto estará disponible en GitHub, permitiendo su consulta, descarga y colaboración. Desde la interfaz web de GitHub, se pueden visualizar los archivos, el historial de cambios y la documentación del proyecto (Bryan, 2018).

17.3 Modificación y seguimiento de proyectos en GitHub

Una vez que el proyecto está en GitHub, es posible continuar trabajando en él y mantener un registro detallado de todas las modificaciones. El flujo de trabajo básico consiste en:

1. Realizar cambios en los archivos del proyecto
Por ejemplo, modificar el script de análisis, agregar nuevos datos, actualizar los resultados exportados o mejorar la documentación.

2. Guardar los cambios en Git
Cada vez que se desee registrar un avance, se puede utilizar el punto para indicarle al software que suba todos los archivos que fueron modificados y realizar un commit con un mensaje descriptivo. Por ejemplo:

```
git add .
git commit -m "Actualización del script con nuevos gráficos"
```

Es importante que el mensaje del commit sea claro y específico, para facilitar la comprensión del historial de cambios.

3. Sincronizar los cambios con GitHub

Para mantener el repositorio remoto actualizado y respaldado, se utiliza el comando:

```
git push
```

Esto sube los cambios al repositorio en línea, donde pueden ser consultados por otros usuarios o por el propio autor desde cualquier lugar.

4. Visualizar el historial y colaborar
GitHub permite revisar el historial completo de commits, comparar versiones de archivos y, en proyectos colaborativos, gestionar solicitudes de cambio (pull requests) y comentarios. Esto facilita la colaboración y la revisión del trabajo en equipo.

El uso regular de Git y GitHub asegura que el proyecto esté siempre respaldado, documentado y listo para ser compartido o retomado en cualquier momento (Bryan, 2018).

17.4 Importación de repositorios de GitHub para trabajo local o personal

Importar un repositorio de GitHub permite descargar una copia completa del proyecto para trabajar localmente, modificarlo o adaptarlo a nuevas necesidades. Este proceso se conoce como "clonar" un repositorio y es útil tanto para uso personal como para colaborar en proyectos de otros usuarios.

Pasos para clonar un repositorio de GitHub:

1. Obtener la URL del repositorio
En la página del repositorio en GitHub, hacer clic en el botón "Code" y copiar la URL que aparece (por ejemplo, `https://github.com/usuario/analisis_estadistico.git`).

2. Clonar el repositorio en la computadora local
Abrir una terminal o la consola de RStudio y ejecutar el siguiente comando:

```
git clone https://github.com/usuario/analisis_estadistico.git
```

Esto creará una carpeta local con todos los archivos y el historial del proyecto.

3. Trabajar localmente
Una vez clonado el repositorio, se puede abrir la carpeta en RStudio, modificar los archivos, ejecutar los scripts y exportar nuevos resultados. Si se tiene permiso para hacerlo, los cambios pueden subirse nuevamente a GitHub con los comandos `git add`, `git commit` y `git push`. Si el objetivo es solo uso personal, los cambios pueden mantenerse localmente sin necesidad de sincronizarlos con el repositorio remoto.

Ventajas de clonar repositorios:

1. Permite reutilizar análisis existentes y aprender de otros proyectos.
2. Facilita la colaboración en equipo, ya que todos los miembros trabajan con la misma versión del proyecto.
3. Asegura la trazabilidad y la integridad del trabajo, ya que todo el historial de cambios se conserva.

La importación de repositorios es una práctica recomendada para quienes desean aprovechar recursos existentes, colaborar en proyectos abiertos o mantener una copia de seguridad de su trabajo (Bryan, 2018).

Parte VI

Referencias

18 Referencias

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

Belsley, D. A., Kuh, E., & Welsch, R. E. (1980). Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. Wiley. https://doi.org/10.1002/0471725153

Bryan, J. (2018). Happy Git and GitHub for the useR. https://happygitwithr.com/

Cleveland, W. S. (1993). Visualizing Data. Hobart Press.

Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2003). *Applied multiple regression/correlation analysis for the behavioral sciences* (3rd ed.). Lawrence Erlbaum Associates Publishers.

Cui, B. (2020). Automate Data Exploration and Treatment [R package DataExplorer version 0.8.2]. *R-Project.org*. https://cran.r-project.org/package=DataExplorer

Draper, N. R., & Smith, H. (1998). Applied Regression Analysis (3rd ed.). Wiley. https://doi.org/10.1002/9781118625590

Field, A. P. (2013). *Discovering statistics using IBM SPSS statistics: and sex and drugs and rock'n'roll* (4th edition). Sage.

Friendly, M. (2008). A Brief History of Data Visualization. In: Handbook of Data Visualization. Springer Handbooks Comp.Statistics. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-33037-0_2

Gentleman, R., & Temple Lang, D. (2007). Statistical analyses and reproducible research. Journal of Computational and Graphical Statistics, 16(1), 1–23. https://doi.org/10.1198/106186007X178663

Grolemund, G., & Wickham, H. (2017). *R for data science*. O'Reilly Media. https://r4ds.had.co.nz/

Hernández, F., Usuga, O., & Mazo, M. (12 de agosto de 2024). *Modelos de Regresión con R*. Github.io. https://fhernanb.github.io/libro_regresion/

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. https://doi.org/10.1080/10618600.1996.10474713

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

Kutner, M. H., Nachtsheim, C. J., Neter, J., & Li, W. (2005). Applied Linear Statistical Models (5th ed.). McGraw-Hill, Irwin, New York.

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

Montgomery, D.C., Peck, E.A. and Vining, G.G. (2012) Introduction to Linear Regression Analysis. Vol. 821, John Wiley & Sons, Hoboken.

Murrell, P. (2018). R Graphics (3rd ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9780429422768

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

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

Rosales Castillo, J. M. (2005). Micropropagación de Calahuala Phlebodium psedoaureum (Cav.) Lellinger con tres tipos de explantes en diferentes medios de cultivo in vitro. Tesis Ing. Agr. Guatemala, Universidad de San Carlos de Guatemala, Facultad de Agronomía. 51 p.

Tabachnick, B., & Fidell, L. (2013). Using Multivariate Statistics (6th ed.). Boston, MA: Pearson.

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

Tufte, E. (2001). The Visual Display of Quantitative Information (2nd ed.). Graphics Press.

Tukey, J. W. (1977). Exploratory Data Analysis. Addison-Wesley.

Venables, W. N., & Ripley, B. D. (2002). Modern Applied Statistics with S (4th ed.). Springer. https://doi.org/10.1007/978-0-387-21706-2

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis* (2. ed.). Springer Cham. https://doi.org/10.1007/978-3-319-24277-4

Wilkinson, L. (2005). *The grammar of graphics* (2nd ed.). Springer. https://doi.org/10.1007/0-387-28695-0

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

Xie, Y., Allaire, J.J., & Grolemund, G. (2018). R Markdown: The Definitive Guide (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781138359444

Parte VII

Ejemplos de Análisis Estadístico con R

19 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.

19.1 Medidas principales en estadística descriptiva

19.1.1 Medidas de tendencia central

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

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

Fórmula:

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

Ejemplo en R:

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

```
[1] 30
```

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

Ejemplo en R:

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

```
[1] 30
```

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

Ejemplo de función para calcular la moda:

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

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

19.1.2 Medidas de dispersión

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

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

Fórmula de la varianza muestral:

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

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

```
[1] 250
```

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

Fórmula:

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

Ejemplo en R:

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

```
[1] 15.81139
```

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

Ejemplo en R:

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

```
[1] 40
```

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

Fórmula:

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

Ejemplo en R:

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

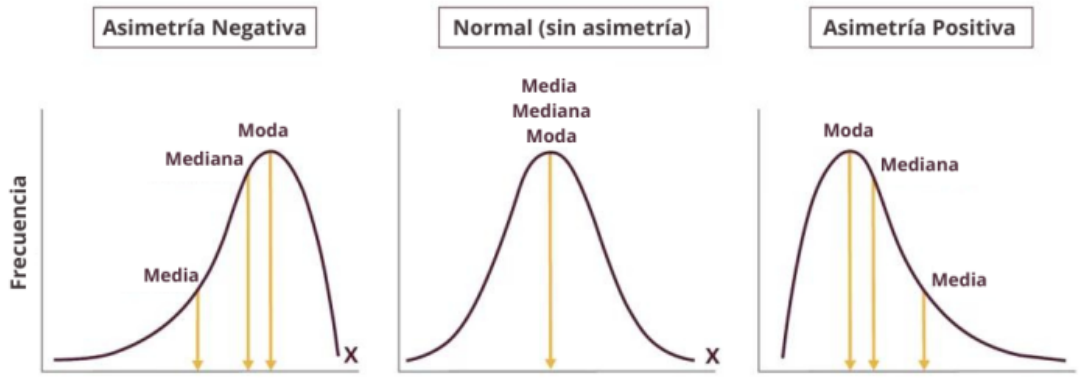
```
[1] 20
```

19.1.3 Medidas de forma

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

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

Guía gráfica para interpretar asimetría:



Fórmula:

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

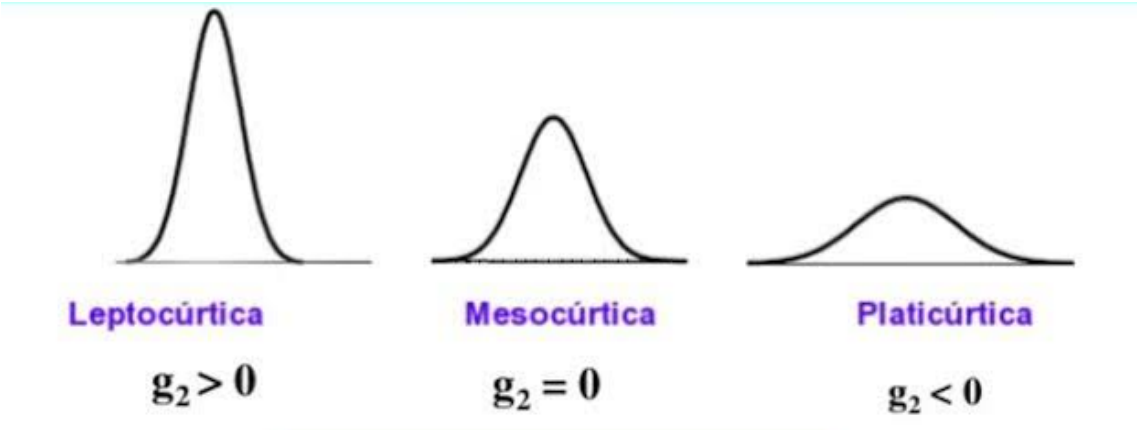
Ejemplo en R (usando el paquete `psych`):

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

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

```
[1] 0
```

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

Guía gráfica para interpretar curtosis:



Fórmula:

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

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

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

```
[1] -1.912
```

19.2 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.

19.2.1 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")
```

19.2.2 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()
```

19.2.3 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")
```

19.2.4 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)))
```

19.2.5 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))
```

19.3 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.

19.3.1 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"
```

19.3.2 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"
```

19.3.3 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"
```

19.3.4 Resumen general con `summary()`

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

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

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

19.4 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.

19.4.1 Instalación y carga del paquete

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

19.4.2 Análisis descriptivo general

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

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

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

# Mostrar resultados
print(resultado_general)
```

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

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

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

19.4.3 Análisis descriptivo categorizado

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

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

# Mostrar resultados
print(resultado_agrupado)
```

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

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

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

19.4.4 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")
```

19.4.5 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.

19.4.6 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.

19.5 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.

19.5.1 Establecer funciones auxiliares

19.5.1.1 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 = ", "))
}
```

19.5.1.2 Asimetría

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

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

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

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

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

19.5.1.3 Curtosis

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

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

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

  # Manejar el caso de desviación estándar cero
```

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

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

19.5.2 Función principal: análisis por categorías

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

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

19.5.3 Función para análisis de múltiples variables numéricas

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

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

19.5.4 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.

19.5.4.1 Preparación del análisis

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

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

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

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

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

19.5.4.2 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.

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

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

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

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

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

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

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

20 Regresión lineal simple usando R

La regresión lineal simple es una técnica estadística utilizada para modelar la relación entre una variable dependiente y una variable independiente. Su propósito principal es predecir el valor de la variable dependiente a partir de la variable independiente, lo que permite entender cómo varía una en función de la otra. Esta técnica es fundamental en el análisis estadístico, ya que proporciona una base para la inferencia y la toma de decisiones en diversos campos (Montgomery, Peck, & Vining, 2012).

Por ejemplo, en economía, la regresión lineal simple puede utilizarse para predecir el consumo en función del ingreso. En biología, se aplica para analizar la relación entre la dosis de un fármaco y la respuesta de un organismo. En ciencias sociales, se utiliza para estudiar cómo factores como la educación influyen en los ingresos de una población (Field, 2013).

20.1 Conceptos fundamentales

Las **variables dependientes** son aquellas que se desean predecir o explicar, mientras que las **variables independientes** son las que se utilizan para realizar dicha predicción. La diferencia radica en que la variable dependiente es el resultado que se estudia, mientras que la variable independiente es el factor que se manipula o se observa (Cohen, Cohen, West, & Aiken, 2003).

La **relación lineal** implica que existe una conexión directa entre las variables, de tal manera que un cambio en la variable independiente produce un cambio proporcional en la variable dependiente.

20.2 Modelo de Regresión Lineal Simple

El modelo de regresión lineal simple se expresa mediante la siguiente ecuación:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

En esta ecuación, $\boldsymbol{Y}$ representa la variable dependiente, $\boldsymbol{X}$ es la variable independiente, $\boldsymbol{\beta 0}$ es la intersección (el valor de $\boldsymbol{Y}$) cuando $\boldsymbol{X}$) es cero) y $\boldsymbol{\beta 1}$ es la pendiente (que indica el cambio en $\boldsymbol{Y}$) por cada unidad de cambio en $\boldsymbol{X}$. El término epilson ( ) representa el error del modelo, que captura la variabilidad en $\boldsymbol{Y}$ que no se explica por $\boldsymbol{X}$ (Kutner, Nachtsheim, Neter, & Li, 2005).

20.3 Supuestos de la Regresión Lineal

Para que el modelo de regresión lineal sea válido, es fundamental que se cumplan ciertos supuestos:

1. **Linealidad**: La relación entre las variables debe ser lineal.
2. **Independencia**: Las observaciones deben ser independientes entre sí.
3. **Homoscedasticidad**: La varianza de los errores debe ser constante a lo largo de todos los niveles de $\boldsymbol{X}$.
4. **Normalidad**: Los errores deben seguir una distribución normal (Tabachnick & Fidell, 2013).

20.4 Evaluación del Modelo

La calidad del modelo de regresión lineal simple se evalúa a través de varias métricas:

1. **R-cuadrado**: Indica la proporción de la variabilidad en la variable dependiente que es explicada por la variable independiente. Un valor cercano a 1 sugiere un buen ajuste del modelo.
2. **Análisis de residuos**: Es crucial analizar los residuos para verificar los supuestos del modelo, asegurando que no haya patrones sistemáticos que indiquen un mal ajuste (Belsley, Kuh, & Welsch, 1980).

20.5 Ejemplo Práctico

Este ejemplo práctico ilustra cómo realizar un análisis de regresión lineal simple utilizando un conjunto de datos sobre esporofitos. Los datos provienen de un estudio realizado en el laboratorio de cultivo de tejidos de la Facultad de Agronomía de la Universidad de San Carlos de Guatemala. En este estudio, se llevó a cabo la reproducción del helecho conocido como calahuala (*Phlebodium pseudoaureum* (Cav.) Lellinger).

Se midió la altura de cada esporofito y se cuantificó la cantidad de esporofitos germinados en 30 frascos que contenían medio de cultivo Murashige y Skoog. Los resultados obtenidos se presentan en el siguiente archivo. Este análisis se basa en la investigación de Rosales Castillo (2005), quien realizó una micropropagación de *Calahuala* utilizando tres tipos de explantes en diferentes medios de cultivo in vitro.

El objetivo de este análisis es evaluar la relación entre la altura de los esporofitos y la cantidad de esporofitos germinados, utilizando la regresión lineal simple como herramienta estadística. A través de este proceso, se busca no solo ajustar un modelo que explique esta relación, sino también verificar los supuestos que sustentan la validez del modelo.

20.5.1 Instalación y Carga de Paquetes

Para comenzar, es necesario instalar y cargar los paquetes requeridos. Estos paquetes proporcionan funciones útiles para la manipulación de datos y la visualización.

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

# Se utiliza para evaluar el supuesto de homocedasticidad
if (!require("car")) install.packages("car")

# Se utiliza para importar archivos de Excel
if (!require("readxl")) install.packages("readxl")
```

Explicación:

1. **`tidyverse`**: Este paquete incluye varias herramientas para la manipulación y visualización de datos, como `ggplot2`, `dplyr` y `tidyr`.
2. **`car`**: Proporciona funciones para realizar pruebas de hipótesis y diagnósticos de modelos, incluyendo la evaluación de homocedasticidad.
3. **`readxl`**: Permite importar datos desde archivos de Excel, facilitando la carga de conjuntos de datos.

20.5.2 Importación de Datos

A continuación, se importan los datos desde un archivo Excel.

```
# Importar un archivo csv
datos <- read_excel("esporofitos.xlsx")

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

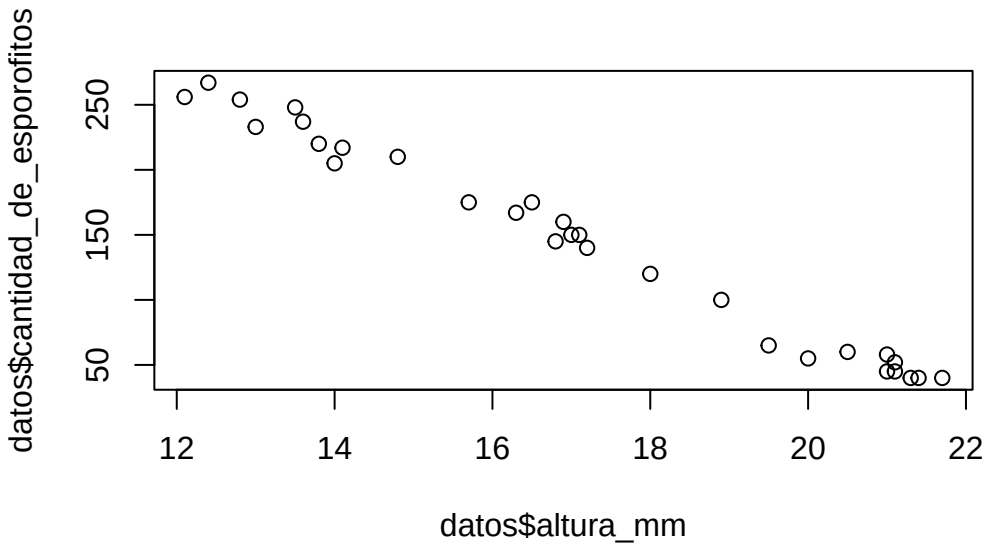
```
# A tibble: 6 x 3
  frasco cantidad_de_esporofitos altura_mm
   <dbl>                   <dbl>     <dbl>
1      1                      40      21.4
2      2                      45      21
3      3                      60      20.5
4      4                      55      20
5      5                      58      21
6      6                      40      21.7
```

Este código carga el archivo de Excel que contiene los datos sobre esporofitos y muestra las primeras filas del conjunto de datos para verificar que se haya importado correctamente.

20.5.3 Visualización de Datos

Se elabora un gráfico de dispersión para observar la relación entre las variables.

```
# Elaboración de un gráfico de dispersión entre altura y cantidad
plot( datos$altura_mm, datos$cantidad_de_esporofitos)
```



Este gráfico permite visualizar la relación entre la cantidad de esporofitos y la altura en milímetros, facilitando la identificación de patrones. Como se puede apreciar en el gráfico anterior hay una relación inversamente proporcional entre la altura de los esporofitos y la cantidad de esporofitos.

20.5.4 Ajuste del Modelo de Regresión

Se ajusta el modelo de regresión lineal simple utilizando la función `lm()`.

```
# Ajuste del modelo
regsimple <- lm(datos$altura_mm ~ datos $cantidad_de_esporofitos)
summary(regsimple)
```

```
Call:
lm(formula = datos$altura_mm ~ datos$cantidad_de_esporofitos)

Residuals:
     Min       1Q   Median       3Q      Max
-0.78523 -0.15056  0.01664  0.22403  0.62850
```

```
Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                   22.8933399  0.1446248  158.29   <2e-16 ***
datos$cantidad_de_esporofitos -0.0401248  0.0008826  -45.46   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3753 on 28 degrees of freedom
Multiple R-squared:  0.9866,    Adjusted R-squared:  0.9862
F-statistic:  2067 on 1 and 28 DF,  p-value: < 2.2e-16
```

El resumen del modelo ajustado proporciona información sobre los coeficientes, errores estándar y estadísticas de ajuste, permitiendo evaluar la calidad del modelo.

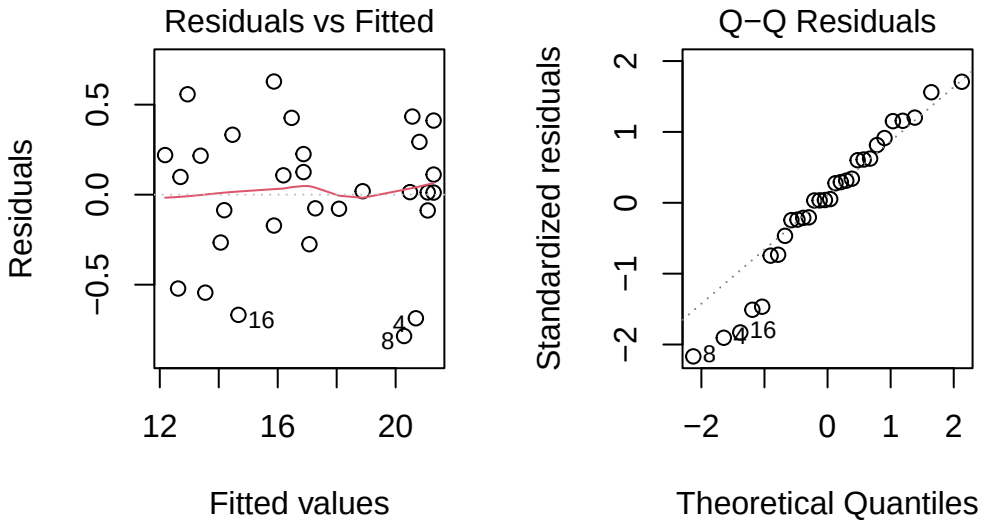
Explicación del código:

1. **`lm()`**: Esta función ajusta un modelo de regresión lineal donde la cantidad de esporofitos es la variable dependiente y la altura es la variable independiente.
2. **`summary(regsimple)`**: Proporciona un resumen del modelo ajustado, incluyendo coeficientes, errores estándar y estadísticas de ajuste.

20.5.5 Gráficos de Diagnóstico para evaluar los supuestos del modelo

Se generan gráficos de diagnóstico para evaluar los supuestos del modelo.

```
# Gráficos de diagnóstico de los supuestos
par(mfrow=c(1,2)) # Crea una matriz de dos gráficos
plot(regsimple, which=1:2)
```



```r
par(mfrow=c(1,1)) # Devuelve a su estado normal el área de gráficos
```

Estos gráficos ayudan a verificar la linealidad y la homocedasticidad, asegurando que los supuestos del modelo se cumplan.

Explicación del código:

`plot(regsimple, which=1:2)`: Genera los gráficos de residuos y ajuste, que ayudan a verificar la linealidad y la homocedasticidad.

20.5.6 Prueba de Normalidad de los Residuos

Se realiza la prueba de Shapiro-Wilk para evaluar la normalidad de los residuos.

```r
# Realizar la prueba de Shapiro-Wilk en los residuos
shapiro.test(residuals(regsimple))
```

```
    Shapiro-Wilk normality test

data:  residuals(regsimple)
W = 0.95651, p-value = 0.2516
```

Explicación:

`shapiro.test(residuals(regsimple))`: Esta prueba evalúa si los residuos del modelo siguen una distribución normal. Un valor p por debajo del nivel de significancia (< 0.05) indica que los residuos no son normales. Para el ejemplo el valor de p es de 0.2516 por lo que no se rechaza la hipótesis nula y por lo tanto no hay suficiente evidencia estadística para indicar que los residuos no son normales.

20.5.7 Prueba de Homocedasticidad de la varianza

Finalmente, se evalúa el supuesto de homocedasticidad utilizando la prueba de heterocedasticidad.

```r
# Realizar prueba para el supuesto de homocedasticidad
ncvTest(regsimple)
```

```
Non-constant Variance Score Test
Variance formula: ~ fitted.values
Chisquare = 0.07681442, Df = 1, p = 0.78166
```

Explicación:

`ncvTest(regsimple)`: Esta función evalúa si la varianza de los residuos es constante a lo largo de los valores de la variable independiente. Un valor p menor al nivel de significancia (<0.05) sugiere que hay heterocedasticidad, lo que puede invalidar el modelo. Para este ejemplo el valor de p es 0.78166 por lo que no hay evidencia estadística suficiente para indicar que la varianza de los residuos no es contante.

21 Regresión múltiple usando R

21.1 Notas:

En esta sección se desarrollará un ejemplo de regresión multiple

Esta pendiente de finalización y carga