R 语言从入门到精通：实战指南

前言

R 语言作为一款开源的编程语言与统计分析工具，凭借其强大的数据处理、统计建模和可视化能力，在数据科学、生物信息学、金融分析等多个领域得到了广泛应用。无论是科研人员、数据分析师，还是编程初学者，掌握 R 语言都能为工作和学习增添强大的助力。

本书旨在为读者提供一套系统、全面且实用的 R 语言学习方案。从最基础的环境搭建开始，逐步深入核心语法、数据结构、数据处理、可视化、统计建模等关键知识点，最后通过实战项目帮助读者巩固所学内容，实现从理论到实践的跨越。书中内容兼顾入门者的易懂性和进阶者的深度需求，每个知识点都配有详细的代码示例和清晰的解释，让读者能够边学边练，快速掌握 R 语言的核心技能。

第一部分：R 语言基础入门

第 1 章：R 语言概述与环境搭建

1.1 R 语言简介

R 语言最初由新西兰奥克兰大学的 Ross Ihaka 和 Robert Gentleman 于 1993 年开发，其设计灵感源自 S 语言。它是一款免费、开源的编程语言，主要用于统计计算、数据分析和数据可视化。R 语言拥有丰富的内置函数和扩展包，能够满足从简单的数据描述到复杂的统计建模等多种需求。

与其他编程语言（如 Python）相比，R 语言在统计分析和数据可视化方面具有独特的优势：它提供了大量专门用于统计检验、回归分析、机器学习等的函数和包，可视化效果精美且高度可定制。同时，R 语言的社区非常活跃，全球各地的开发者不断贡献新的扩展包，使其功能持续丰富和完善。

1.2 R 与 RStudio 的安装

1.2.1 安装 R

1. 访问 R 语言官方网站（https://www.r-project.org/），在首页点击 “Download R” 进入下载页面。
2. 根据自己的操作系统（Windows、macOS、Linux）选择对应的镜像站点。
3. 对于 Windows 系统，下载.exe 安装文件，双击运行后按照默认设置逐步完成安装；对于 macOS 系统，下载.dmg 安装文件，打开后将 R 拖入应用程序文件夹即可。

1.2.2 安装 RStudio

RStudio 是一款专门为 R 语言设计的集成开发环境（IDE），提供了代码编辑、运行、调试、可视化等一站式功能，极大地提升了编程效率。

1. 访问 RStudio 官方网站（https://www.rstudio.com/），点击 “Download RStudio Desktop”。
2. 选择适合自己操作系统的免费版本进行下载。
3. 安装过程简单，按照默认设置完成即可。安装完成后，打开 RStudio，将自动关联已安装的 R 语言环境。

1.3 RStudio 界面介绍

RStudio 的界面主要分为四个核心区域，每个区域都有其特定的功能：

1. **代码编辑区（左上方）**：用于编写和保存 R 代码文件（.R 文件）。支持语法高亮、自动补全、代码折叠等功能，方便用户编写和管理代码。
2. **控制台（左下方）**：是 R 语言的命令行交互区域，用于执行代码并查看运行结果。在代码编辑区选中代码后，按 Ctrl+Enter（Windows）或 Cmd+Enter（macOS）即可在控制台运行该代码。
3. **环境与历史区（右上方）**：包含 “Environment” 和 “History” 两个标签。“Environment” 显示当前工作空间中的变量、数据框、函数等对象；“History” 记录了在控制台中执行过的所有命令，方便用户回顾和重复执行。
4. **文件与图表区（右下方）**：包含 “Files”“Plots”“Packages”“Help” 等多个标签。“Files” 用于浏览本地文件系统；“Plots” 显示代码生成的可视化图表；“Packages” 用于管理 R 语言的扩展包（安装、加载、更新）；“Help” 提供 R 语言函数和包的详细帮助文档。

第 2 章：R 语言核心语法

2.1 基本数据类型

R 语言的基本数据类型主要包括数值型、字符型、逻辑型、整数型和因子型，不同类型的数据在存储和处理方式上有所区别。

2.1.1 数值型（numeric）

数值型是 R 语言中最常用的数据类型，用于表示实数，包括整数和小数。例如：

```
# 定义数值型变量
```

```
x
y
# 查看数据类型
class(x)  # 输出 "numeric"
class(y)  # 输出 "numeric"
```

2.1.2 字符型（character）

字符型用于表示文本数据，需要用单引号或双引号包裹。例如：

```
# 定义字符型变量
name 语言"
message Hello, R!'
# 查看数据类型
class(name)  # 输出 "character"
```

2.1.3 逻辑型（logical）

逻辑型仅包含两个值：TRUE（真）和 FALSE（假），主要用于条件判断。例如：

```
# 定义逻辑型变量
a  > 3  # TRUE
b  == 8  # FALSE
# 查看数据类型
class(a)  # 输出 "logical"
```

2.1.4 整数型（integer）

整数型专门用于表示整数，定义时需要在数值后加 L。例如：

```
# 定义整数型变量
num  5L
```

```
# 查看数据类型
class(num)  # 输出 "integer"
```

2.1.5 因子型（factor）

因子型用于表示分类数据（如性别、职业、学历等），可以将字符型数据转换为离散的分类变量，便于统计分析。例如：

```
# 定义因子型变量
gender  factor(c("男", "女", "男", "女"))
# 查看因子的水平（分类）
levels(gender)  # 输出 "女" "男"
# 查看数据类型
class(gender)  # 输出 "factor"
```

2.2 变量与赋值

在 R 语言中，变量用于存储数据或计算结果，赋值操作是定义变量的核心。R 语言支持多种赋值符号，常用的有`<-`（推荐使用）和`=`。

2.2.1 变量命名规则

1. 变量名只能以字母、下划线（_）或点（.）开头，不能以数字开头。
2. 变量名中可以包含字母、数字、下划线和点。
3. 变量名区分大小写，例如 `x` 和 `X` 是两个不同的变量。
4. 不能使用 R 语言的关键字（如 `if`、`for`、`function` 等）作为变量名。

2.2.2 赋值操作示例

```
# 用 赋值（推荐）
age   85, 95, 88)  # 向量赋值
# 用 = 赋值
name = "张三"
```

```
# 查看变量的值
age  # 输出 25
name  # 输出 "张三"
score  # 输出 90 85 95 88
```

2.3 运算符

R 语言支持多种运算符，包括算术运算符、赋值运算符、比较运算符、逻辑运算符等，用于实现数据的计算和判断。

2.3.1 算术运算符

用于数值计算，包括+（加）、-（减）、*（乘）、/（除）、^（幂）、%%（取余）、%/%（整除）。示例：

```
a   b  # 输出 13
a - b  # 输出 7
a * b  # 输出 30
a / b  # 输出 3.333...
a ^ 2  # 输出 100
a %% b  # 输出 1（10 除以 3 的余数）
a %/% b  # 输出 3（10 除以 3 的整数部分）
```

2.3.2 比较运算符

用于比较两个值的大小或是否相等，返回逻辑型结果（TRUE/FALSE），包括==（等于）、!=（不等于）、>（大于）、``>=（大于等于）、小于等于）。示例：

```
5 == 5  # 输出 TRUE
5 != 3  # 输出 TRUE
8 > 10  # 输出 FALSE
6 输出 TRUE
```

2.3.3 逻辑运算符

用于逻辑值的运算，包括&（逻辑与）、|（逻辑或）、!（逻辑非）。示例：

```
x <- TRUE
y
x & y  # 输出 FALSE（两者都为真才为真）
x | y  # 输出 TRUE（至少一个为真即为真）
!x  # 输出 FALSE（取反）
```

2.3.4 其他运算符

- %in%：判断元素是否在某个集合中，返回逻辑型结果。示例：

```
3 %in% c(1, 2, 3, 4)  # 输出 TRUE
"a" %in% c("b", "c", "d")  # 输出 FALSE
```

- :：生成连续的整数序列。示例：

```
1:5  # 输出 1 2 3 4 5
5:1  # 输出 5 4 3 2 1
```

第 3 章：R 语言数据结构

R 语言提供了多种数据结构，用于组织和存储不同类型、不同维度的数据，常用的数据结构包括向量、矩阵、数组、数据框和列表。

3.1 向量（Vector）

向量是 R 语言中最基础的数据结构，用于存储同一类型的一维数据。可以通过 c()函数（combine 的缩写）创建向量。

3.1.1 向量的创建

```
# 数值型向量
num_vec  2, 3, 4, 5)
```

```
# 字符型向量
char_vec 苹果", "香蕉", "橙子")
# 逻辑型向量
log_vec  c(TRUE, FALSE, TRUE)
# 因子型向量
factor_vec  factor(c("优秀", "良好", "及格", "良好"))
```

3.1.2 向量的索引与切片

向量的索引用于获取或修改向量中的元素，R 语言的索引从 1 开始（而非 0）。

1. 按位置索引：

```
vec , 20, 30, 40, 50)
vec[1]  # 取第 1 个元素，输出 10
vec[3:5]  # 取第 3 到 5 个元素，输出 30 40 50
vec[c(2, 4)]  # 取第 2 和 4 个元素，输出 20 40
```

1. 按条件索引：

```
vec[vec > 30]  # 取大于 30 的元素，输出 40 50
vec[vec %% 20 == 0]  # 取能被 20 整除的元素，输出 20 40
```

1. 修改元素：

```
vec[2]   # 将第 2 个元素改为 25
vec  # 输出 10 25 30 40 50
```

3.1.3 向量的常用操作

```
vec 3, 1, 4, 1, 5, 9)
length(vec)  # 查看向量长度，输出 6
```

```
sum(vec) # 求和，输出 23
mean(vec) # 求平均值，输出 3.833...
sd(vec) # 求标准差，输出 2.944...
sort(vec) # 排序（默认升序），输出 1 1 3 4 5 9
unique(vec) # 去重，输出 3 1 4 5 9
```

3.2 矩阵（Matrix）

矩阵是用于存储同一类型的二维数据结构，由行和列组成。可以通过 matrix()函数创建矩阵。

3.2.1 矩阵的创建

```
# 基本创建：data 为数据，nrow 为行数，ncol 为列数
mat 1:12, nrow = 3, ncol = 4)
# 查看矩阵
mat
# 输出：
#      [,1] [,2] [,3] [,4]
# [1,]    1    4    7   10
# [2,]    2    5    8   11
# [3,]    3    6    9   12
# 按行填充数据（默认按列填充）
mat_row (data = 1:12, nrow = 3, ncol = 4, byrow = TRUE)
# 查看按行填充的矩阵
mat_row
# 输出：
#      [,1] [,2] [,3] [,4]
# [1,]    1    2    3    4
# [2,]    5    6    7    8
# [3,]    9   10   11   12
# 给矩阵的行和列命名
```

```
rownames(mat)  c("行 1", "行 2", "行 3")
colnames(mat) ", "列 2", "列 3", "列 4")
mat
```

3.2.2 矩阵的索引与切片

矩阵的索引格式为 mat[行索引, 列索引]，索引可以是位置、名称或条件。

1. 按位置索引：

```
mat[2, 3]  # 取第 2 行第 3 列的元素，输出 8
mat[1:2, 2:4]  # 取第 1-2 行、第 2-4 列的子矩阵
mat[, 3]  # 取第 3 列（所有行），输出 7 8 9
mat[2, ]  # 取第 2 行（所有列），输出 2 5 8 11
```

1. 按名称索引：

```
mat["行 2", "列 3"]  # 输出 8
mat[c("行 1", "行 3"), c("列 1", "列 4")]  # 取指定行和列的子矩阵
```

1. 按条件索引：

```
mat[mat > 5]  # 取所有大于 5 的元素，输出 6 7 8 9 10 11 12
```

3.2.3 矩阵的常用操作

```
# 查看矩阵维度（行数和列数）
dim(mat)  # 输出 3 4
# 矩阵转置
t(mat)  # 行变列，列变行
# 矩阵乘法（需满足列数等于行数）
mat1 , nrow = 2)
```

```
mat2 (5:8, nrow = 2)
mat1 %*% mat2  # 矩阵乘法
# 按行/列求和、求均值
rowSums(mat)  # 按行求和
colMeans(mat)  # 按列求均值
```

3.3 数组（Array）

数组是矩阵的扩展，用于存储同一类型的多维数据（维度可以大于 2）。通过 array()函数创建，指定 dim 参数定义各维度的长度。

3.3.1 数组的创建

```
# 创建 3 维数组：数据为 1-24，维度为 2 行、3 列、4 层
arr  array(data = 1:24, dim = c(2, 3, 4))
# 查看数组
arr
# 给各维度命名
dimnames(arr) (
  c("行 1", "行 2"),  # 第 1 维（行）名称
  c("列 1", "列 2", "列 3"),  # 第 2 维（列）名称
  c("层 1", "层 2", "层 3", "层 4")  # 第 3 维（层）名称
)
```

3.3.2 数组的索引

数组的索引格式为 arr[维度 1 索引, 维度 2 索引, 维度 3 索引, ...]。

```
arr[1, 2, 3]  # 取第 1 行、第 2 列、第 3 层的元素
arr[, , "层 2"]  # 取第 2 层的所有行和列（返回一个矩阵）
arr["行 1", c("列 1", "列 3"), 1:2]  # 取指定行、列、层的元素
```

3.4 数据框（Data Frame）

数据框是 R 语言中最常用的数据结构，用于存储不同类型的二维数据（类似 Excel 表格），每一列可以是不同的数据类型（如数值型、字符型、逻辑型），但每一列内部的数据类型必须一致。通过 data.frame()函数创建。

3.4.1 数据框的创建

```
# 创建数据框
df  姓名 = c("张三", "李四", "王五", "赵六"),
  年龄 = c(22, 25, 23, 24),
  性别 = factor(c("男", "男", "女", "女")),
  成绩 = c(85, 92, 88, 90)
)
# 查看数据框
df
# 输出:
#   姓名 年龄 性别 成绩
# 1 张三  22  男   85
# 2 李四  25  男   92
# 3 王五  23  女   88
# 4 赵六  24  女   90
```

3.4.2 数据框的索引与切片

数据框的索引方式灵活，支持按位置、名称、条件索引。

1. 按列名索引（最常用）：

```
df$姓名  # 取"姓名"列，输出 "张三" "李四" "王五" "赵六"
df[c("姓名", "成绩")]  # 取"姓名"和"成绩"两列（返回数据框）
```

1. 按位置索引:

```
df[2, 3]  # 取第 2 行第 3 列的元素，输出 "男"
df[1:3, ]  # 取第 1-3 行（所有列）
df[, 2:4]  # 取第 2-4 列（所有行）
```

1. 按条件索引（筛选行）：

```
# 筛选成绩大于 90 的行
df[df$成绩 > 90, ]
# 筛选性别为女且年龄小于 24 的行
df[df$性别 == "女" & df$年龄  ]
```

3.4.3 数据框的常用操作

```
# 查看数据框基本信息
str(df)  # 查看数据结构（列类型、前几个元素）
head(df)  # 查看前 6 行（默认）
tail(df)  # 查看后 6 行（默认）
nrow(df)  # 查看行数，输出 4
ncol(df)  # 查看列数，输出 4
colnames(df)  # 查看列名，输出 "姓名" "年龄" "性别" "成绩"
# 新增列
df$排名  c(4, 1, 3, 2)  # 新增"排名"列
df$是否及格  >= 60  # 新增逻辑型列
# 删除列
df$排名 "排名"列
# 排序
df_sorted 成绩, decreasing = TRUE), ]  # 按成绩降序排序
```

3.5 列表（List）

列表是 R 语言中最灵活的数据结构，可以存储不同类型、不同结构的数据（如向量、矩阵、数据框、甚至另一个列表）。通过 list()函数创建。

3.5.1 列表的创建

```
# 创建列表
my_list 姓名 = "张三",
  基本信息 = c(年龄 = 25, 性别 = "男", 身高 = 175),
  成绩 = matrix(c(85, 90, 88, 92), nrow = 2),
  爱好 = c("读书", "运动", "编程")
)
# 查看列表
my_list
```

3.5.2 列表的索引与修改

列表的索引有三种方式：[[ ]]、[ ]和$。

1. 按名称索引（$或[[ ]]）：

```
my_list$姓名  # 输出 "张三"
my_list[["基本信息"]]  # 输出 年龄 性别 身高 25 男 175
my_list[["成绩"]][1, 2]  # 取列表中矩阵的第 1 行第 2 列元素，输出 90
```

1. 按位置索引（[[ ]]或[ ]）：

```
my_list[[2]]  # 取第 2 个元素（基本信息），输出 年龄 性别 身高 25 男 175
my_list[1:2]  # 取第 1-2 个元素（返回列表）
```

1. 修改列表元素：

```
my_list$姓名  # 修改"姓名"元素
```

```
my_list[[3]] 0, 95, 92, 98), nrow = 2)  # 修改"成绩"矩阵
```

3.5.3 列表的常用操作

```
length(my_list)  # 查看列表长度（元素个数），输出 4
names(my_list)  # 查看列表元素名称，输出 "姓名" "基本信息" "成绩" "爱好"
unlist(my_list)  # 将列表转换为向量（仅当所有元素可转换为同一类型时适用）
```

第二部分：R 语言数据处理与可视化

第 4 章：数据读取与保存

在实际数据分析中，数据通常存储在外部文件（如 CSV、Excel、文本文件等）中，R 语言提供了多种函数用于读取和保存这些外部数据。

4.1 读取常见数据文件

4.1.1 读取 CSV 文件

CSV（Comma-Separated Values）文件是最常用的数据存储格式，R 语言通过 read.csv()函数读取。

```
# 基本用法：read.csv(文件路径, header = TRUE, stringsAsFactors = FALSE)
# header = TRUE 表示第一行为列名，stringsAsFactors = FALSE 表示字符型不自动转为因子型
df_csv .csv("data.csv", header = TRUE, stringsAsFactors = FALSE)
# 读取本地文件（相对路径）：文件在当前工作目录下的 data 文件夹中
df_csv2 .csv("data/data.csv")
# 读取网络 CSV 文件（通过 URL）
df_url https://example.com/data.csv")
# 查看读取的数据
head(df_csv)
```

4.1.2 读取 Excel 文件

读取 Excel 文件（.xlsx 或.xls 格式）需要安装并加载 readxl 包（开源免费）。

1. 安装并加载包:

```
install.packages("readxl") # 安装包（仅需安装一次）
library(readxl) # 加载包（每次使用前需加载）
```

1. 读取 Excel 文件:

```
# 读取.xlsx 文件，指定工作表（通过工作表名称或索引）
df_excel cel("data.xlsx", sheet = "Sheet1") # 按名称
df_excel2 _excel("data.xlsx", sheet = 1) # 按索引（第 1 个工作表）
# 读取.xls 文件（同样支持）
df_xls ("data.xls", sheet = "数据")
# 查看数据
str(df_excel)
```

4.1.3 读取文本文件

文本文件（.txt）的读取使用 read.table()函数，需指定分隔符（如空格、制表符等）。

```
# 读取以空格分隔的文本文件
df_txt data.txt", header = TRUE, sep = " ", stringsAsFactors = FALSE)
# 读取以制表符分隔的文本文件（类似 CSV，分隔符为\t）
df_tab .table("data_tab.txt", header = TRUE, sep = "\t")
# 处理缺失值：na.strings = "NA" 表示将"NA"识别为缺失值
df_na _na.txt", header = TRUE, sep = ",", na.strings = "NA")
```

4.2 数据保存

4.2.1 保存为 CSV 文件

使用 write.csv()函数将数据框保存为 CSV 文件。

```
# 基本用法：write.csv(数据框, 文件路径, row.names = FALSE)
# row.names = FALSE 表示不保存行索引
write.csv(df_csv, "output_data.csv", row.names = FALSE)
# 保存到指定文件夹（若文件夹不存在，需先创建）
dir.create("output") # 创建 output 文件夹
write.csv(df_csv, "output/result.csv", row.names = FALSE)
```

4.2.2 保存为 Excel 文件

保存为 Excel 文件需要安装并加载 writexl 包或 openxlsx 包，这里以 writexl 为例（用法更简洁）。

1. 安装并加载包：

```
install.packages("writexl")
library(writexl)
```

1. 保存 Excel 文件：

```
# 保存为.xlsx 文件
write_xlsx(df_excel, "output/result.xlsx")
# 保存多个数据框到不同工作表
write_xlsx(
  list(工作表 1 = df1, 工作表 2 = df2),
  "output/multi_sheet.xlsx"
)
```

4.2.3 保存为 R 数据文件

R 数据文件（.RData 或.Rda 格式）可以保存 R 语言中的任意对象（数据框、矩阵、列表等），便于后续直接加载使用，保存和加载速度快。

```
# 保存单个对象
```

```
save(df_csv, file = "output/data.RData")
# 保存多个对象
save(df_csv, df_excel, my_list, file = "output/multi_data.RData")
# 加载 R 数据文件（加载后对象直接进入工作空间）
load("output/data.RData")
```

第 5 章：数据清洗与预处理

原始数据往往存在缺失值、重复值、异常值等问题，需要进行清洗和预处理才能用于后续分析。本节将介绍使用基础 R 函数和 dplyr 包（高效数据处理包）进行数据清洗的常用方法。

5.1 缺失值处理

缺失值在 R 语言中用 NA 表示，处理缺失值的核心是识别缺失值并根据情况进行删除或填充。

5.1.1 识别缺失值

```
# 模拟含缺失值的数据框
df_na  a = c(1, 2, NA, 4, 5),
  b = c("x", NA, "z", "y", NA),
  c = c(TRUE, FALSE, TRUE, NA, FALSE)
)
# 查看数据框
df_na
# 识别缺失值（返回逻辑型矩阵）
is.na(df_na)
# 统计每列缺失值个数
colSums(is.na(df_na))
# 统计每行缺失值个数
rowSums(is.na(df_na))
# 查看含有缺失值的行
df_na[!complete.cases(df_na), ]  # complete.cases()返回无缺失值的行（TRUE）
```

5.1.2 缺失值处理方法

1. 删除缺失值：当缺失值比例较低（如小于 5%）时，可直接删除含缺失值的行或列。

```
# 删除所有含缺失值的行（默认）
df_del_row na)
# 删除缺失值比例超过 50%的列
df_del_col Means(is.na(df_na)) 5]
```

1. 填充缺失值：当缺失值比例较高时，可通过均值、中位数、众数等进行填充。

```
# 数值型列用均值填充
df_na$a[is.na(df_na$a)] (df_na$a, na.rm = TRUE)  # na.rm = TRUE 表示计算均值时忽略 NA
# 数值型列用中位数填充（适用于存在异常值的情况）
df_na$a[is.na(df_na$a)]  median(df_na$a, na.rm = TRUE)
# 字符型/因子型列用众数填充
get_mode  function(x) {
  names(sort(table(x, useNA = "no"), decreasing = TRUE))[1]
}
df_na$b[is.na(df_na$b)] $b)
# 用前一个非缺失值填充（向前填充）
df_na$c  zoo::na.locf(df_na$c)  # 需要加载 zoo 包
# 用后一个非缺失值填充（向后填充）
df_na$c .locf(df_na$c, fromLast = TRUE)
```

5.2 重复值处理

重复值会影响分析结果的准确性，需要识别并删除。

```
# 模拟含重复值的数据框
df_dup
  id = c(1, 2, 2, 3, 4, 4, 4),
```

```
  name = c("张三", "李四", "李四", "王五", "赵六", "赵六", "赵六"),
  score = c(85, 90, 90, 88, 92, 92, 92)
)
# 查看数据框
df_dup
# 识别重复行（返回逻辑型向量，TRUE 表示重复行）
duplicated(df_dup)
# 统计重复行个数
sum(duplicated(df_dup))
# 删除重复行（保留第一行）
df_unique _dup)
# 按指定列删除重复行（如按 id 列，保留第一行）
df_unique_id [!duplicated(df_dup$id), ]
# 按指定列删除重复行，保留最后一行
df_unique_last _dup[!duplicated(df_dup$id, fromLast = TRUE), ]
```

5.3 异常值处理

异常值（离群点）是指与其他数据点差异极大的数据，可能由数据录入错误或真实极端情况导致，需要识别并处理。

5.3.1 识别异常值

常用的异常值识别方法包括箱线图法（IQR 法则）和标准差法。

1. 箱线图法（IQR 法则）：

```
# 模拟含异常值的数值型数据
score , 90, 88, 92, 86, 95, 20, 89)  # 20 为异常值
# 计算四分位数和 IQR（四分位距）
q1  0.25)
q3 (score, 0.75)
iqr
# 定义异常值阈值（上下限）
```

```
lower_bound  1.5 * iqr
upper_bound 3 + 1.5 * iqr
# 识别异常值
outliers <- score[score _bound | score > upper_bound]
outliers  # 输出 20
# 用箱线图可视化异常值
boxplot(score, main = "成绩箱线图（含异常值）")
```

1. 标准差法：

```
# 定义异常值：偏离均值超过 3 个标准差
mean_score )
sd_score (score)
outliers_sd (score - mean_score) > 3 * sd_score]
outliers_sd  # 输出 20
```

5.3.2 异常值处理方法

```
# 1. 删除异常值
score_clean _bound & score ]
# 2. 替换为阈值（ Winsorization 处理）
score_winsor
score_winsor[score_winsor _bound] score_winsor[score_winsor > upper_bound]  3. 替换为均值/中位数（适用于异常值由录入错误导致）
score_replace
score_replace[score_replace  > upper_bound] (score)
```

5.4 使用 dplyr 包进行高效数据处理

dplyr 是 R 语言中用于数据处理的核心包，提供了一套简洁、高效的函数，支持数据筛选、排序、分组、汇总等操作，语法清晰易懂。

5.4.1 dplyr 包的安装与加载

```
install.packages("dplyr") # 安装包
library(dplyr) # 加载包
```

5.4.2 dplyr 核心函数

dplyr 的核心函数包括 filter()（筛选行）、arrange()（排序）、select()（选择列）、mutate()（新增列）、group_by()（分组）、summarise()（汇总）等，这些函数可以通过管道符%>%串联使用，使代码更简洁。

1. 筛选行（filter ()）：

```
# 筛选成绩大于 90 且年龄小于 25 的行
df_filter (成绩 > 90, 年龄 5) # 多个条件用逗号分隔（逻辑与）
```

1. 排序（arrange ()）：

```
# 按年龄升序排序，年龄相同按成绩降序排序
df_arrange  arrange(年龄, desc(成绩)) # desc()表示降序
```

1. 选择列（select ()）：

```
# 选择姓名、年龄、成绩列
df_select
  select(姓名, 年龄, 成绩)
# 排除性别列（保留其他列）
df_select2
  select(-性别)
# 选择以"姓"开头的列
df_select3
  select(starts_with("姓"))
```

1. 新增列（mutate ()）：

```
# 新增"总分"列（假设成绩为单科分，总分=成绩*2）
# 新增"等级"列（成绩>=90 为"A"，80-89 为"B"，否则为"C"）
df_mutate  %>%
  mutate(
    总分 = 成绩 * 2,
    等级 = case_when(
      成绩 >= 90 ~ "A",
      成绩 >= 80 ~ "B",
      TRUE ~ "C"
    )
  )
```

1. 分组汇总（group_by () + summarise ()）：

```
# 按性别分组，计算每组的平均年龄、最高成绩、人数
df_summary  group_by(性别) %>%
  summarise(
    平均年龄 = mean(年龄),
    最高成绩 = max(成绩),
    人数 = n()  # n()表示每组的行数
  )
```

1. 管道符的使用：

管道符%>%的作用是将前一个函数的输出作为后一个函数的输入，避免嵌套函数，使代码更易读。例如：

```
# 不使用管道符（嵌套写法）
summarise(group_by(filter(df, 年龄  性别), 平均成绩 = mean(成绩))
# 使用管道符（清晰易懂）
```

```
df %>%
  filter(年龄 25) %>%
  group_by(性别) %>%
  summarise(平均成绩 = mean(成绩))
```

第 6 章：R 语言数据可视化

数据可视化是数据分析的重要环节，通过图表可以直观地展示数据的分布、趋势和关系。R 语言拥有强大的可视化功能，常用的包包括基础绘图包（graphics）、ggplot2 包（语法优雅、高度可定制）等。

6.1 基础绘图包（graphics）

基础绘图包是 R 语言内置的绘图工具，无需安装，通过 plot()、hist()、boxplot()等函数可以快速绘制常见图表。

6.1.1 散点图（plot ()）

散点图用于展示两个数值型变量之间的关系。

```
# 模拟数据
x (100, mean = 50, sd = 10)  # 100 个服从正态分布的 x 值
y  x + rnorm(100, mean = 0, sd = 5)  # y 与 x 呈线性关系，加随机噪声
# 绘制散点图
plot(
  x = x,
  y = y,
  main = "x 与 y 的散点图",  # 图表标题
  xlab = "变量 x",  # x 轴标签
  ylab = "变量 y",  # y 轴标签
  col = "blue",  # 点的颜色
  pch = 16,  # 点的形状（16 为实心圆）
  cex = 0.8  # 点的大小
)
```

```
# 添加拟合直线
abline(lm(y ~ x), col = "red", lwd = 2)  # lm(y ~ x)为线性回归模型，lwd 为线宽
# 添加图例
legend(
  "bottomright",  # 图例位置
  legend = c("数据点", "拟合直线"),
  col = c("blue", "red"),
  pch = c(16, NA),
  lty = c(NA, 1),
  lwd = c(NA, 2)
)
```

6.1.2 直方图（hist ()）

直方图用于展示单个数值型变量的分布情况。

```
# 模拟数据（正态分布）
data_hist norm(500, mean = 0, sd = 1)
# 绘制直方图
hist(
  x = data_hist,
  main = "正态分布数据直方图",
  xlab = "数值",
  ylab = "频数",
  col = "lightgreen",  # 柱子颜色
  border = "black",  # 柱子边框颜色
  breaks = 30,  # 柱子数量（默认自动计算）
  prob = TRUE  # 显示概率密度（而非频数）
)
# 添加密度曲线
lines(density(data_hist), col = "red", lwd = 2)
```

6.1.3 箱线图（boxplot ()）

箱线图用于展示数值型变量的分布特征（中位数、四分位数、异常值）。

```
# 模拟数据（三组不同分布的数据）
data_box   组 1 = rnorm(100, mean = 10, sd = 2),
  组 2 = rnorm(100, mean = 15, sd = 3),
  组 3 = rnorm(100, mean = 8, sd = 1.5)
)
# 绘制箱线图
boxplot(
  x = data_box,
  main = "三组数据的箱线图",
  xlab = "组别",
  ylab = "数值",
  col = c("lightblue", "lightyellow", "lightpink"),
  horizontal = FALSE  # 垂直箱线图（TRUE 为水平）
)
```

6.1.4 条形图（barplot ()）

条形图用于展示分类变量的频数或统计量。

```
# 模拟数据（分类变量频数）
category ("A", "B", "C", "D")
frequency 5, 42, 28, 50)
# 绘制条形图
barplot(
  height = frequency,
  names.arg = category,  # x 轴分类名称
  main = "各分类的频数条形图",
```

```
  xlab = "分类",
  ylab = "频数",
  col = "orange",
  border = "black"
)
# 在柱子上方添加数值标签
text(
  x = 1:length(category),
  y = frequency + 1,
  labels = frequency,
  adj = 0.5  # 标签水平居中
)
```

6.2 ggplot2 包可视化

ggplot2 是基于 “图形语法” 的可视化包，由 Hadley Wickham 开发，具有语法优雅、高度可定制、图表美观等优点，是 R 语言中最受欢迎的可视化包。

6.2.1 ggplot2 包的安装与加载

```
install.packages("ggplot2")  # 安装包
library(ggplot2)  # 加载包
```

6.2.2 ggplot2 核心语法

ggplot2 的核心语法结构为：

```
ggplot(data = 数据框, aes(x = x 变量, y = y 变量, ...)) +
  几何对象函数(...) + 主题函数(...)
```

- data：指定绘图数据框。
- aes()：映射数据到图形属性（x 轴、y 轴、颜色、形状、填充等）。
- 几何对象函数：指定图表类型（如 geom_point()散点图、geom_histogram()直方图等）。

- 主题函数：调整图表的外观（如 theme_bw()、theme_minimal()等）。

6.2.3 常见图表绘制（ggplot2）

1. 散点图（geom_point ()）：

```
# 模拟数据
df_ggplot   x = rnorm(100, 50, 10),
  y = 2 * x + rnorm(100, 0, 5),
  group = sample(c("A", "B"), 100, replace = TRUE)  # 分组变量
)
# 绘制散点图（按 group 分组，不同颜色和形状）
ggplot(data = df_ggplot, aes(x = x, y = y, color = group, shape = group)) +
  geom_point(size = 2, alpha = 0.7) +  # alpha 为透明度（0-1）
  geom_smooth(method = "lm", se = FALSE) +  # 添加线性拟合线，se=FALSE 不显示置信区间
  labs(
    title = "x 与 y 的散点图（按分组）",
    x = "变量 x",
    y = "变量 y",
    color = "分组",
    shape = "分组"
  ) +
  theme_bw() +  # 白色背景主题
  theme(
    plot.title = element_text(hjust = 0.5),  # 标题水平居中
    legend.position = "bottom"  # 图例位置在底部
  )
```

1. 直方图（geom_histogram ()）：

```
# 模拟数据
```

```
df_hist  = rnorm(500, 0, 1))
# 绘制直方图（添加密度曲线）
ggplot(data = df_hist, aes(x = value)) +
  geom_histogram(aes(y = ..density..), bins = 30, fill = "lightgreen", color = "black") +
  geom_density(color = "red", size = 1) +
  labs(
    title = "正态分布数据直方图",
    x = "数值",
    y = "概率密度"
  ) +
  theme_minimal()
```

1. 箱线图（geom_boxplot ()）：

```
# 模拟数据
df_box   group = rep(c("组 1", "组 2", "组 3"), each = 100),
  value = c(rnorm(100, 10, 2), rnorm(100, 15, 3), rnorm(100, 8, 1.5))
)
# 绘制箱线图（按 group 分组，不同填充颜色）
ggplot(data = df_box, aes(x = group, y = value, fill = group)) +
  geom_boxplot(alpha = 0.7) +
  labs(
    title = "三组数据的箱线图",
    x = "组别",
    y = "数值",
    fill = "组别"
  ) +
  scale_fill_manual(values = c("lightblue", "lightyellow", "lightpink")) +  # 自定义填充颜色
  theme(plot.title = element_text(hjust = 0.5))
```

1. 条形图（geom_bar ()/geom_col ()）：

- geom_bar()：用于绘制频数条形图（自动统计频数）。

- geom_col()：用于绘制已知数值的条形图（直接使用 y 轴变量）。

```
# 模拟数据（分类变量频数）
df_bar (
  category = c("A", "B", "C", "D"),
  frequency = c(35, 42, 28, 50)
)
# 绘制条形图（geom_col()）
ggplot(data = df_bar, aes(x = category, y = frequency, fill = category)) +
  geom_col(color = "black") +
  geom_text(aes(label = frequency), vjust = -0.3) +  # 在柱子上方添加数值标签
  labs(
    title = "各分类的频数条形图",
    x = "分类",
    y = "频数",
    fill = "分类"
  ) +
  scale_fill_brewer(palette = "Set2") +  # 使用 RColorBrewer 配色方案
  theme(
    plot.title = element_text(hjust = 0.5),
    legend.position = "none"  # 隐藏图例（分类与 x 轴一致，无需重复）
  ) +
  ylim(0, max(df_bar$frequency) * 1.1)  # 调整 y 轴范围，避免标签超出
```

1. 折线图（geom_line ()）：

折线图用于展示数据随时间或连续变量的变化趋势。

```
# 模拟数据（时间序列）
df_line
  date = seq.Date(as.Date("2023-01-01"), as.Date("2023-01-30"), by = "day"),
```

```
  value = cumsum(rnorm(30, 0.5, 1)) # 累积和，模拟趋势
)
# 绘制折线图
ggplot(data = df_line, aes(x = date, y = value)) +
  geom_line(color = "blue", size = 1) +
  geom_point(color = "red", size = 1.5) + # 添加数据点
  labs(
    title = "2023 年 1 月数据变化趋势",
    x = "日期",
    y = "数值"
  ) +
  theme_bw() +
  theme(
    plot.title = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1) # x 轴标签旋转 45 度，避免重叠
  )
```

6.3 图表保存

使用 ggsave()函数（ggplot2 包）可以将绘制的图表保存为图片文件（PNG、JPG、PDF 等格式），支持自定义图片大小和分辨率。

```
# 保存 ggplot2 绘制的图表
p _scatter, aes(x = x, y = y)) + geom_point() # 假设 p 为绘制好的图表
# 保存为 PNG 格式
ggsave(
  filename = "scatter_plot.png",
  plot = p,
  width = 10, # 图片宽度（单位：英寸）
  height = 6, # 图片高度（单位：英寸）
  dpi = 300, # 分辨率（像素/英寸）
```

```
  path = "output/plots"  # 保存路径
)
# 保存为 PDF 格式（矢量图，放大不失真）
ggsave(
  filename = "scatter_plot.pdf",
  plot = p,
  width = 10,
  height = 6,
  path = "output/plots"
)
# 保存基础绘图包绘制的图表
png("output/plots/histogram.png", width = 800, height = 500, res = 150)  # 打开 PNG 设备
hist(data_hist, col = "lightgreen")  # 绘制图表
dev.off()  # 关闭设备，保存图片
```

第三部分：R 语言统计建模与实战

第 7 章：统计分析基础

R 语言的核心优势之一是强大的统计分析功能，涵盖描述性统计、假设检验、方差分析等多种统计方法。本节将介绍常用的统计分析方法及实现代码。

7.1 描述性统计

描述性统计用于概括数据的基本特征，包括集中趋势（均值、中位数、众数）、离散程度（标准差、方差、四分位距）、分布形态（偏度、峰度）等。

7.1.1 数值型变量的描述性统计

```
# 模拟数据
data_stats (200, mean = 50, sd = 10)
# 基础描述性统计（summary()函数）
summary(data_stats)
```

```
# 输出：最小值、第一四分位数、中位数、均值、第三四分位数、最大值
# 详细描述性统计（psych 包的 describe()函数）
install.packages("psych")
library(psych)
describe(data_stats)
# 输出：n（样本量）、mean（均值）、sd（标准差）、median（中位数）、min（最小值）、max（最大值）、skew（偏度）、kurtosis（峰度）等
```

7.1.2 数据框的描述性统计

```
# 对数据框的数值型列进行描述性统计
describe(df[, c("年龄", "成绩")]) # 仅选择数值型列
# 使用 dplyr 包按分组进行描述性统计
df %>%
  group_by(性别) %>%
  summarise(
    年龄均值 = mean(年龄),
    年龄标准差 = sd(年龄),
    成绩中位数 = median(成绩),
    成绩四分位距 = IQR(成绩)
  )
```

7.2 假设检验

假设检验是统计推断的核心方法，用于判断样本数据是否支持关于总体的某个假设。常用的假设检验包括 t 检验、卡方检验、方差分析等。

7.2.1 t 检验（t-test）

t 检验用于检验两个总体的均值是否存在显著差异，分为独立样本 t 检验和配对样本 t 检验。

1. 独立样本 t 检验（检验两个独立总体的均值差异）：

```
# 模拟数据（两组独立样本）
group1 norm(50
```

> （注：文档部分内容可能由 AI 生成）