

An Introduction to R

Dr. Kapil Kumar

Associate Professor, Department of Statistics
Chaudhary Charan Singh University, Meerut, India
kapilstats@gmail.com

UGC-MMTTC Short Term Programme on Applications of Computational Software in
Mathematical Sciences and Engineering (19–24 May 2025)

UGC-Malaviya Mission Teacher Training Centre, Central University of Haryana,
Mahendergarh

21 May 2025 · Day 3 lecture

Plan of the session

1. Getting started with R and RStudio
2. Numbers, vectors and matrices
3. Data objects and data files
4. Graphics
5. Programming basics
6. A glimpse of statistics in R

By the end you can work in an RStudio project, compute with vectors and matrices, build lists and data frames, import a CSV file, draw and customise plots, write a small function with a loop, and run a first *t*-test and regression.

No prior R needed. The code on the slides runs top to bottom as one script.

PART 1

Getting started

What is R?

- A **language and environment** for statistical computing and graphics
- Free and open source (GNU GPL); Windows, macOS and Linux
- A full programming language: conditions, loops, recursive functions
- Extended by **packages**: about 22,400 on CRAN in May 2025
- An environment for implementing methods, not a fixed menu

A short history **1970s:** John Chambers and colleagues create the S language at Bell Labs.
1992–95: Ross Ihaka and Robert Gentleman (Auckland) develop R, “not unlike S”.
29 Feb 2000: version 1.0.0.
The name comes from the first names of its two authors.

RStudio: one window, four panes

Source

scripts (.R); run a line with Ctrl+Enter

Environment / History

objects you have created, past commands

Console

where code runs and output appears

Files / Plots / Packages / Help

graphs, packages, documentation

Install in this order

1. R from CRAN

cran.r-project.org

2. RStudio Desktop (free)

posit.co/download/rstudio-desktop

RStudio (by Posit) is an IDE: R does the computing, RStudio organises the work.

Scripts, projects and good habits

- Write code in a **script** (File → New File → R Script): it can be saved, rerun and shared
- Anything after # is a comment: explain *why*, not only what
- One **RStudio project** per task: the script plus a data/ folder
- Read files with **relative** paths, e.g. "data/file.csv"

Avoid `setwd` with a hard-coded folder on your own disk: the script then runs only on your computer.

Reproducible by design Restart R and run the whole script from the top. If it still works, anyone can repeat your analysis.

Packages and getting help

- A **package** bundles functions, data and documentation
- `install.packages()` once per computer; `library()` once per session
- Help: `?mean`, `help("t.test")`, `??regression`, `example(persp)`
- R ships with many data sets, e.g. `sleep`

```
# install.packages("lattice") # once
library(lattice)               # each session
col_main <- "#1F5FA8"          # plot colours
col_alt  <- "#B45309"
dim(sleep)                     # a built-in data set
head(sleep, 3)

## [1] 20 3
##   extra group ID
## 1   0.7     1  1
## 2  -1.6     1  2
## 3  -0.2     1  3
```

PART 2

Numbers, vectors and matrices

R as a calculator

Operator	Meaning
+ - * /	arithmetic
^	power
%%	remainder (mod)
%%/%	integer division

Usual precedence: powers, then $*$ $/$, then $+$ $-$. Use brackets when in doubt. Angles are in radians. Each output line below matches one line of code.

```
c(2 + 3 * 5, (2 + 3) * 5)      # precedence
c(17 %% 5, 17 %/% 5)           # mod, quotient
c(sqrt(2), exp(1), log10(100))
c(factorial(5), choose(5, 2))   # 5!, 5C2
sin(pi / 6)                     # radians

## [1] 17 25
## [1] 2 3
## [1] 1.414 2.718 2.000
## [1] 120 10
## [1] 0.5
```

Objects and vectors

- `<-` assigns a value to a name
- Names start with a letter and are **case sensitive**: `x` and `X` are different
- `c()` combines values into a **vector**, the basic object in R
- `sort()` returns values, `order()` returns positions

```
x <- c(2, 4, 3, 5, 7, 9, 15)
x
length(x)
summary(x)
sort(x, decreasing = TRUE)
order(x)           # positions, smallest first

## [1]  2  4  3  5  7  9 15
## [1] 7
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      2.00   3.50   5.00   6.43   8.00  15.00
## [1] 15  9  7  5  4  3  2
## [1] 1 3 2 4 5 6 7
```

Arithmetic is element-wise

$$x^2 = (x_1^2, \dots, x_n^2)$$
$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

`log()`, `sqrt()`, `sin()` and the operators act on every element at once: no loop is needed. This is **vectorisation**. Here we check the textbook formula for s^2 against `var()`.

```
x + 1
x^2
round(log(x), 2)
d <- x - mean(x)           # deviations
sum(d^2) / (length(x) - 1) # s^2 by formula
var(x)                     # built-in agrees

## [1]  3  5  4  6  8 10 16
## [1]  4 16  9 25 49 81 225
## [1] 0.69 1.39 1.10 1.61 1.95 2.20 2.71
## [1] 19.95
## [1] 19.95
```

Extracting elements with []

- Positions start at 1
- A negative index *drops* elements
- A logical vector keeps the TRUE positions
- `which()` turns TRUEs into positions
- Named vectors can be indexed by name

```
x[c(2, 4)]  
x[-2]  
x[x > 5]  
which(x > 5)  
names(x) <- LETTERS[1:7]  
x[c("A", "G")]  
  
## [1] 4 5  
## [1] 2 3 5 7 9 15  
## [1] 7 9 15  
## [1] 5 6 7  
## A G  
## 2 15
```

Sequences, repetition and binding

```
3:8
seq(3, 5, by = 0.5)
seq(0, 1, length.out = 5)
rep(c(2, 3), times = c(5, 4))
rep(1:3, times = 2)
rep(1:3, each = 2)

## [1] 3 4 5 6 7 8
## [1] 3.0 3.5 4.0 4.5 5.0
## [1] 0.00 0.25 0.50 0.75 1.00
## [1] 2 2 2 2 2 3 3 3 3
## [1] 1 2 3 1 2 3
## [1] 1 1 2 2 3 3
```

```
x1 <- c(1, 2, 3, 4)
y1 <- c(2, 1, 4, 3)
cbind(x1, y1)      # side by side
rbind(x1, y1)      # one above the other

##      x1 y1
## [1,]  1  2
## [2,]  2  1
## [3,]  3  4
## [4,]  4  3
##      [,1] [,2] [,3] [,4]
## x1      1   2   3   4
## y1      2   1   4   3
```

Matrices and linear algebra

Solve $Ax = b$:

$$2x_1 + x_2 - x_3 = 8$$

$$-3x_1 - x_2 + 2x_3 = -11$$

$$-2x_1 + x_2 + 2x_3 = -3$$

`%*%` is the matrix product, `*` is element-wise. Also `t()`, `diag()`, `solve(A)` (inverse), `eigen()`.

```
A <- matrix(c( 2,  1, -1,
              -3, -1,  2,
              -2,  1,  2), nrow = 3, byrow = TRUE)
b <- c(8, -11, -3)
det(A)
x_sol <- solve(A, b)      # solves A x = b
x_sol
all.equal(c(A %*% x_sol), b) # check A x = b

## [1] -1
## [1]  2  3 -1
## [1] TRUE
```

PART 3

Data objects and data files

Types of data, and factors for categories

- Basic types: numeric, integer, character, logical, complex
- A vector holds **one** type: mixing values converts all of them to the most general type
- A **factor** stores a categorical variable with its **levels**; `ordered = TRUE` keeps an ordering (Low < Medium < High)

```
sapply(list(3.14, 5L, "R", TRUE, 2+3i), class)
c(1, "two", TRUE)      # all become text
gender <- factor(c("Male", "Female", "Female",
                  "Male", "Male"))

levels(gender)
table(gender)

## [1] "numeric"  "integer"  "character" "logical"
## [5] "complex"
## [1] "1"      "two"    "TRUE"
## [1] "Female" "Male"
## gender
## Female   Male
##        2     3
```

Lists: containers for unlike objects

- A **list** can hold vectors of different types and lengths, matrices, even other lists
- `$name` or `[[ "name" ]]` returns an element; `[ "name" ]` returns a smaller *list*

Why it matters Results of `t.test()` and `lm()` are lists: you extract what you need with `$`.

```
data2 <- list(x = c(2, 5, 4),
              Name = c("X", "Y", "Z"),
              Median = 4)

str(data2)
data2$x
data2[[ "Name" ]][2]

## List of 3
## $ x      : num [1:3] 2 5 4
## $ Name   : chr [1:3] "X" "Y" "Z"
## $ Median: num 4
## [1] 2 5 4
## [1] "Y"
```

Data frames: the shape of most data sets

- A **data frame** is a list of columns of equal length
- Each column has one type; columns may differ
- Rows are observations, columns are variables

Example: test scores of five students in each of two batches.

```
score <- c(20, 20.5, 13, 16, 11,
           40, 42, 38, 56, 57)
group <- rep(c("Batch_A", "Batch_B"), each = 5)
batch <- data.frame(score, group)
head(batch, 3)
dim(batch)

##   score  group
## 1  20.0 Batch_A
## 2  20.5 Batch_A
## 3  13.0 Batch_A
## [1] 10  2
```

Working with a data frame

```
batch$score[1:3]           # one column
batch[batch$score > 41, ]   # rows by condition
subset(batch, group == "Batch_A" & score > 18)

## [1] 20.0 20.5 13.0
##      score  group
## 7      42 Batch_B
## 9      56 Batch_B
## 10     57 Batch_B
##      score  group
## 1  20.0 Batch_A
## 2  20.5 Batch_A
```

```
batch$pass <- batch$score >= 20 # new column
table(batch$group, batch$pass)
aggregate(score ~ group, data = batch,
          FUN = mean)

##
##           FALSE  TRUE
## Batch_A         3    2
## Batch_B         0    5
##      group score
## 1 Batch_A  16.1
## 2 Batch_B  46.6
```

Importing a CSV file

- `read.csv()` reads a comma-separated file into a data frame
- `read.table()` for text files;
`readxl::read_excel()` for Excel
- Always check what arrived:
`dim()`, `str()`, `summary()`

Illustrative data: printers sold and price charged at 20 stores in one month.

```
printers <- read.csv("data/SellingPrinters.csv")
head(printers, 4)
summary(printers$Price)
anyNA(printers)           # any missing values?
```

##	Store	Sold	Price			
##	1	Store1	100	21000		
##	2	Store2	150	15050		
##	3	Store3	130	16800		
##	4	Store4	140	15750		
##	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
##	13650	14700	16275	16485	18200	21000
##	[1]	FALSE				

Exporting data, and reading it back

- `write.csv()` saves a data frame as a CSV file that any program can open
- `row.names = FALSE` avoids an extra column of row numbers
- `saveRDS()` / `readRDS()` keep R objects exactly, including factors

```
write.csv(batch, "data/batch_scores.csv",
          row.names = FALSE)
batch2 <- read.csv("data/batch_scores.csv")
str(batch2, vec.len = 2)
all.equal(batch, batch2) # same content?

## 'data.frame': 10 obs. of 3 variables:
## $ score: num 20 20.5 13 16 11 ...
## $ group: chr "Batch_A" "Batch_A" ...
## $ pass : logi TRUE TRUE FALSE ...
## [1] TRUE
```

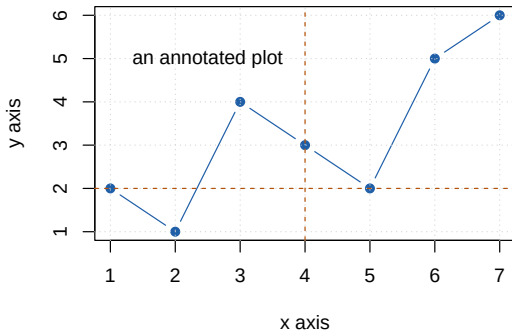
PART 4

Graphics

High-level and low-level graphics

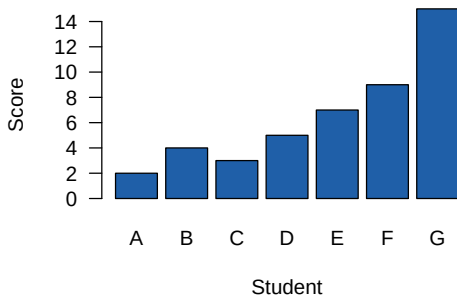
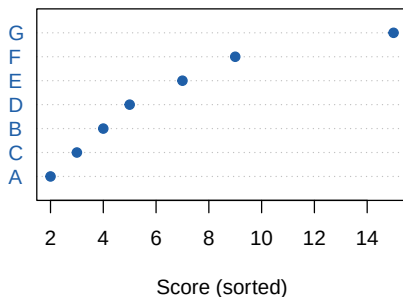
```
x2 <- 1:7; y2 <- c(2, 1, 4, 3, 2, 5, 6)
par(mar = c(4, 4, 1, 1))
plot(x2, y2, type = "b", pch = 19,
     col = col_main, xlab = "x axis",
     ylab = "y axis")
grid()
abline(h = 2, v = 4, col = col_alt, lty = 2)
text(2.5, 5, "an annotated plot")
```

High-level functions start a plot: `plot()`, `hist()`, `boxplot()`. Low-level ones add to it: `points()`, `lines()`, `abline()`, `text()`, `legend()`.



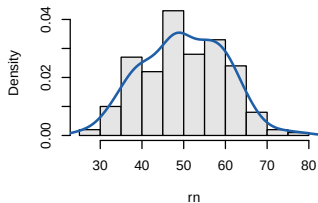
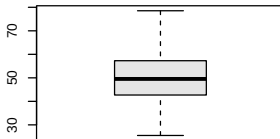
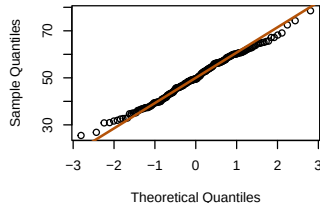
Bar charts and dot charts

```
par(mfrow = c(1, 2), mar = c(4, 4, 0.5, 1))      # x: the seven named scores from before
barplot(x, las = 1, col = col_main, xlab = "Student", ylab = "Score")
dotchart(sort(x), pch = 19, col = col_main, xlab = "Score (sorted)")
```



Looking at a distribution

```
set.seed(2025); rn <- rnorm(200, mean = 50, sd = 10) # 200 draws from N(50, 10^2)
par(mfrow = c(1, 3), mar = c(4, 4, 1, 1))
hist(rn, freq = FALSE, col = "grey90", main = ""); lines(density(rn), col = col_main, lwd = 2)
boxplot(rn, col = "grey90"); qqnorm(rn, main = ""); qqline(rn, col = col_alt, lwd = 2)
```

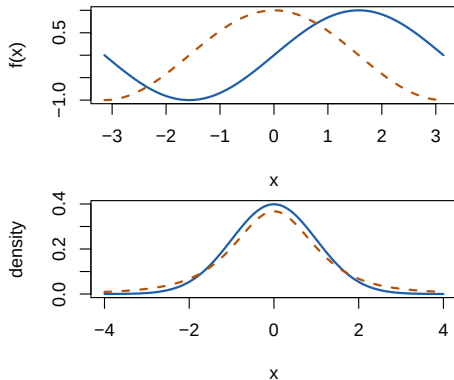


Symmetric, bell-shaped, no outliers, points on the Q-Q line: as expected for normal draws.

Plotting mathematical functions with `curve()`

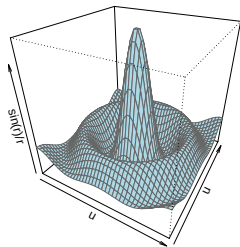
```
par(mfrow = c(2, 1), mar = c(4, 4, 0.5, 1))
curve(sin(x), -pi, pi, col = col_main,
      lwd = 2, ylab = "f(x)")
curve(cos(x), add = TRUE, col = col_alt,
      lwd = 2, lty = 2)
curve(dnorm(x), -4, 4, col = col_main,
      lwd = 2, ylab = "density")
curve(dt(x, df = 3), add = TRUE,
      col = col_alt, lwd = 2, lty = 2)
```

Top: $\sin x$ (solid) and $\cos x$ (dashed). Bottom: the $N(0, 1)$ density (solid) and the t_3 density (dashed), whose tails are heavier.

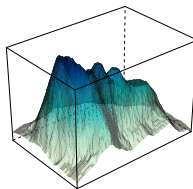


Surfaces in three dimensions

```
u <- seq(-10, 10, length.out = 40)
r <- sqrt(outer(u^2, u^2, "+"))
par(mar = c(0, 0, 0, 0))
persp(u, u, sin(r) / r, theta = 30, phi = 30,
      col = "lightblue", border = "grey40")
```



```
# lattice: heights of the Maunga Whau volcano
wireframe(volcano, shade = TRUE, zoom = 1.05,
          xlab = "", ylab = "", zlab = "",
          par.settings = list(
            axis.line = list(col = NA)))
```



PART 5

Programming basics

Writing your own function

Anatomy

```
name <- function(arguments) {  
  # body: the calculations  
  return(value)  
}
```

If you can compute it at the console, you can wrap it in a function. The body is vectorised, so a vector argument works too. A missing bracket gives an error: read the message.

```
square <- function(number) {  
  answer <- number * number  
  return(answer)  
}  
  
square(3)  
square(0.8)  
square(1:5)  
  
## [1] 9  
## [1] 0.64  
## [1] 1 4 9 16 25
```

Example: standard error of the mean

$$SE(\bar{x}) = \frac{s}{\sqrt{n}}, \quad n = \text{number of observed values}$$

Base R has no SE function, so we write one. The naive version fails once a value is missing: `var()` gives NA and `length()` counts the NA.

Fix Keep the observed values first: `x[!is.na(x)]`.

```
se_naive <- function(x) sqrt(var(x) / length(x))
se_mean <- function(x) {
  x <- x[!is.na(x)]      # observed values only
  sqrt(var(x) / length(x))
}

set.seed(1)
y1 <- rnorm(100, mean = 5, sd = 1)
y1[50] <- NA              # one value goes missing
se_naive(y1)              # fails
se_mean(y1)               # works

## [1] NA
## [1] 0.09039
```

Returning several results: normal MLEs

For $x_1, \dots, x_n \sim N(\mu, \sigma^2)$:

$$\hat{\mu} = \bar{x}, \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

The MLE of σ^2 divides by n , whereas `var()` and `sd()` divide by $n - 1$. A function returns one object, so several results go into a [list](#).

```
normal_mle <- function(x) {  
  n <- length(x)  
  var_est <- var(x) * (n - 1) / n  
  list(mean_mle = mean(x), sd_mle = sqrt(var_est))  
}  
  
set.seed(123)  
w <- rnorm(100, mean = 5, sd = 2)  
str(normal_mle(w))  
sd(w)                                # divisor n - 1  
  
## List of 2  
##  $ mean_mle: num 5.18  
##  $ sd_mle  : num 1.82  
## [1] 1.826
```

Decisions: if, else and ifelse()

```
set.seed(7)
u1 <- runif(1, min = 0, max = 10)
if (u1 > 3) {
  y <- 10
} else {
  y <- 0
}
c(u1 = u1, y = y)

##      u1      y
## 9.889 10.000
```

- if needs a *single* TRUE or FALSE
- else must follow the closing } on the same line
- For a whole vector use ifelse(test, yes, no)

```
ifelse(c(2, 8, 5, 11) > 5, "high", "low")

## [1] "low"  "high" "low"  "high"
```

Loops: computing a double sum

$$S = \sum_{i=1}^n \sum_{j \neq i} \left(\frac{1}{x_i} - \frac{1}{x_j} \right)^2$$

The loop visits $i = 1, \dots, n$; $x[-i]$ holds all x_j with $j \neq i$, so the inner sum is vectorised.

Check `outer()` forms all $1/x_i - 1/x_j$ at once (diagonal = 0).

```
double_sum <- function(x) {  
  s <- 0  
  for (i in seq_along(x)) {  
    s <- s + sum((1 / x[i] - 1 / x[-i])^2)  
  }  
  s  
}  
v <- 1:10  
double_sum(v)  
sum(outer(1 / v, 1 / v, "-")^2) # same answer  
  
## [1] 13.84  
## [1] 13.84
```

The apply family: loops without writing loops

```
A1 <- matrix(1:20, nrow = 5, byrow = TRUE)
dim(A1)           # rows 1:4, 5:8, ...
apply(A1, 1, mean) # 1 = rows
apply(A1, 2, sd)   # 2 = columns
lst <- list(A1 = A1, B1 = 1:5, C1 = 8)
sapply(lst, mean)  # one mean per element

## [1] 5 4
## [1] 2.5 6.5 10.5 14.5 18.5
## [1] 6.325 6.325 6.325 6.325
##   A1   B1   C1
## 10.5 3.0 8.0
```

Which one?

`apply(X, MARGIN, FUN)`: rows or columns of a matrix.

`lapply(X, FUN)`: each element of a list; returns a list.

`sapply(X, FUN)`: as `lapply()`, but simplifies to a vector or matrix.

PART 6

A glimpse of statistics in R

Comparing two batches: Welch two-sample t -test

```
tt <- t.test(score ~ group, data = batch2)
tt$estimate                                # group means
round(c(t = unname(tt$statistic),
        df = unname(tt$parameter),
        p = tt$p.value), 4)
round(tt$conf.int[1:2], 2)                # 95% CI

## mean in group Batch_A mean in group Batch_B
##                16.1                46.6
##      t      df      p
## -6.7744  5.6050  0.0007
## [1] -41.71 -19.29
```

$$H_0 : \mu_A = \mu_B \text{ VS } H_1 : \mu_A \neq \mu_B$$

$$t = \frac{\bar{x}_A - \bar{x}_B}{\sqrt{s_A^2/n_A + s_B^2/n_B}}$$

Reading it Batch B scored 30.5 points higher on average; the 95% CI for $\mu_A - \mu_B$ is (-41.7, -19.3), $p = 0.0007$: a clear difference, even with five students per batch.

Paired data: did a film reduce smoking?

```
smoke <- read.csv("data/smoker_data.csv")
smoke$Before - smoke$After      # per smoker
pt <- t.test(smoke$Before, smoke$After,
             paired = TRUE, alternative = "greater")
round(c(t = unname(pt$statistic),
         df = unname(pt$parameter),
         p = pt$p.value), 4)

## [1]  2  2 -1  0  0  4  3  2
##      t      df      p
## 2.5100 7.0000 0.0202
```

Mean daily cigarettes of 8 heavy smokers a month before and after an anti-smoking film (illustrative data). The question asks for a *decline*, so $H_1 : \mu_{\text{before}} - \mu_{\text{after}} > 0$ is one-sided.

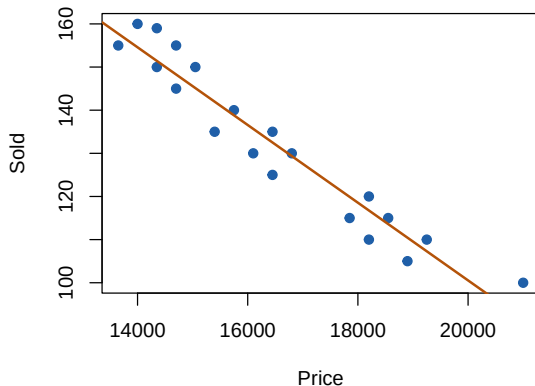
Reading it Mean reduction 1.5 cigarettes a day, $p = 0.02$: evidence of a decline at the 5% level. Without a control group we cannot credit the film alone.

Printer sales and price: look first

```
par(mar = c(4, 4, 1, 1))
plot(Sold ~ Price, data = printers,
     pch = 19, col = col_main)
fit <- lm(Sold ~ Price, data = printers)
abline(fit, col = col_alt, lwd = 2)
cor(printers$Price, printers$Sold)

## [1] -0.9626
```

A strong, roughly linear, negative relationship: stores that charged more sold fewer printers. A straight line looks reasonable.



Fitting the regression line with `lm()`

```
cf <- coef(summary(fit))
cbind(round(cf[, 1:3], 4), p = signif(cf[, 4], 3))
summary(fit)$r.squared
predict(fit, newdata = data.frame(Price = 15000))
```

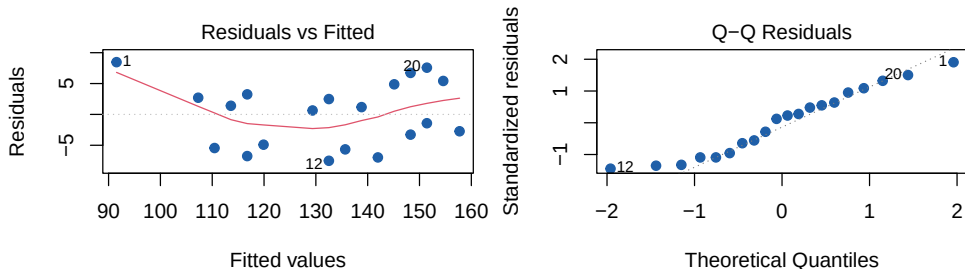
```
##           Estimate Std. Error t value      p
## (Intercept)  280.676     9.9255   28.28 2.27e-16
## Price        -0.009     0.0006  -15.07 1.19e-11
## [1] 0.9265
##      1
## 145.6
```

$$\text{Sold}_i = \beta_0 + \beta_1 \text{Price}_i + \varepsilon_i$$
$$i = 1, \dots, 20$$

Reading it Each extra 1,000 in price lowers expected sales by about 9 printers ($p < 0.001$); $R^2 = 0.927$. At a price of 15,000 the model expects about 146 printers.

Checking the fit: residual diagnostics

```
par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))  
plot(fit, which = 1:2, col = col_main, pch = 19)
```



No clear curve or funnel in the residuals; the Q-Q plot is close to a line (Shapiro-Wilk $p = 0.24$). The line is adequate within the observed prices only.

Least squares by matrix algebra: Hald's cement data

```
cement <- read.csv("data/CementData.csv")
X <- cbind(1, as.matrix(cement[, 1:4]))
y <- cement$Heat
beta <- solve(t(X) %*% X, t(X) %*% y)
fit_c <- lm(Heat ~ ., data = cement)
round(cbind(matrix = beta[, 1],
             lm = coef(fit_c)), 4)

##           matrix      lm
##           62.4054 62.4054
## Ingredient_1  1.5511  1.5511
## Ingredient_2  0.5102  0.5102
## Ingredient_3  0.1019  0.1019
## Ingredient_4 -0.1441 -0.1441
```

Normal equations and solution:

$$X^T X \hat{\beta} = X^T y$$
$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Caution In all 13 samples the percentages add to 95–99%: nearly collinear columns. `lm()` uses a QR decomposition, which is safer.

Much more is one function call away

Task	Function(s)
Correlation test	<code>cor.test()</code>
One-way ANOVA	<code>aov()</code> , <code>TukeyHSD()</code>
Contingency tables	<code>chisq.test()</code> , <code>fisher.test()</code>
Rank-based tests	<code>wilcox.test()</code> , <code>kruskal.test()</code>
Generalised linear models	<code>glm()</code>
Time series	<code>ts()</code> , <code>arima()</code>
Simulation	<code>rnorm()</code> , <code>sample()</code> , <code>replicate()</code>
Reproducible reports	<code>knitr</code> , <code>rmarkdown</code>

The pattern is always the same Get the data into a data frame, look at it, call the function, read the result, check the assumptions.

Tomorrow's session, *Exploratory Data Analysis using R*, builds on exactly this pattern.

Check your understanding

1. Who were the original developers of R?
(a) Chambers and Ihaka (b) Ihaka and Gentleman (c) Gentleman and Allaire
2. What does `mean(x)` return?
(a) the sum (b) the maximum (c) the average (d) the median
3. What does `summary(x)` give for a numeric vector?
(a) missing values only (b) min, quartiles, mean, max (c) a graph
4. Which function reads a CSV file?
(a) `readLines()` (b) `read.csv()` (c) `load()`
5. Which RStudio pane shows graphs?
(a) Console (b) Source (c) Environment (d) Plots
6. What does `x[-2]` return?
(a) the element -2 (b) all but the second element (c) an error

Answers: 1 (b), 2 (c), 3 (b), 4 (b), 5 (d), 6 (b).

Key take-aways

1. **Work in a script inside a project** and read data with relative paths; then the analysis can be rerun by anyone.
2. **Everything is an object**, and the vector is the basic one: arithmetic and most functions act element by element.
3. **Pick the right container:** vector, matrix, list or data frame; factors for categories.
4. **Look at the data** with `str()`, `summary()` and a plot before any test.
5. **Write a function** as soon as you repeat a calculation, and handle missing values explicitly.
6. **Read output, not only p -values:** estimates, intervals and residual checks.

Next session (22 May): *Exploratory Data Analysis using R*, putting these tools to work on real data sets.

References

- Dalgaard, P. (2008). *Introductory Statistics with R*, 2nd Ed., Springer, New York.
- Jones, O., Harden, S., and Crawley, M. J. (2022). *The R Book*, 3rd Ed., Wiley, Hoboken.
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. <https://www.R-project.org>
- Verzani, J. (2014). *Using R for Introductory Statistics*, 2nd Ed., CRC Press, Boca Raton.
- Wickham, H., Çetinkaya-Rundel, M., and Golemund, G. (2023). *R for Data Science*, 2nd Ed., O'Reilly, Sebastopol. (Free online: r4ds.hadley.nz)
- Software: CRAN <https://cran.r-project.org>; RStudio Desktop <https://posit.co/download/rstudio-desktop/>

Thank you

Questions and discussion