

Data Analytics and Introduction to R

Dr. Kapil Kumar

Assistant Professor, Department of Statistics
Central University of Haryana, Mahendergarh, India
kapilstats@gmail.com

One Day E-Workshop on Data Analytics, National Statistics Day 2022
Department of Mathematics & Statistics, Manipal University Jaipur

29 June 2022 · Online

Plan of the session

1. Statistics and data analytics
2. Getting to know R and RStudio
3. First steps: vectors and data files
4. Programming: functions, loops, `apply()`
5. Basic statistics and graphics

By the end you can name the four types of data analytics, install R and RStudio, summarise vectors, read a CSV file, write a small function with a loop, and draw standard graphs.

How to follow Every code box runs top to bottom in a fresh R session.

PART 1

Statistics and data analytics

What is statistics?

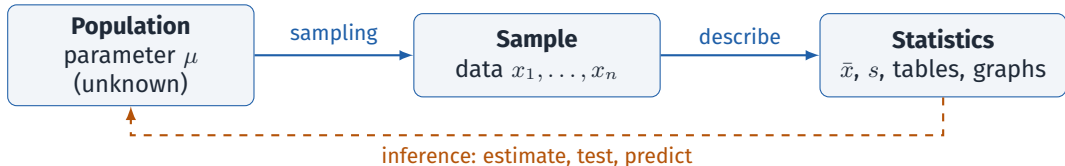
- **Statistics** is the science of collecting, organising, summarising, presenting, analysing and interpreting data.
- It is the basic toolkit of a data scientist: gather data, analyse them and report what was found.
- Two branches are used throughout data analytics:
 - descriptive statistics
 - inferential (inductive) statistics

National Statistics Day India marks 29 June, the birth anniversary of P. C. Mahalanobis, as National Statistics Day: a reminder that sound data are the basis of sound decisions.

Descriptive and inferential statistics

Descriptive Summarises the data in hand: tables, graphs, averages, spread, percentages. It makes no claim beyond the observed data.

Inferential Uses a sample to learn about the population it came from: estimates, confidence intervals, tests and predictions, each with a statement of uncertainty.



Both at work: *World Health Statistics 2021*

- In 2021 the World Health Organization published *World Health Statistics 2021: Monitoring Health for the SDGs*.
- The report examines global trends in population health, country by country.
- It uses descriptive statistics to show *what is happening* and inferential methods to say *what is likely to happen*, from the data available.

who.int/publications/i/item/9789240027053

Descriptive side reported indicators summarised in tables, maps and trend charts.

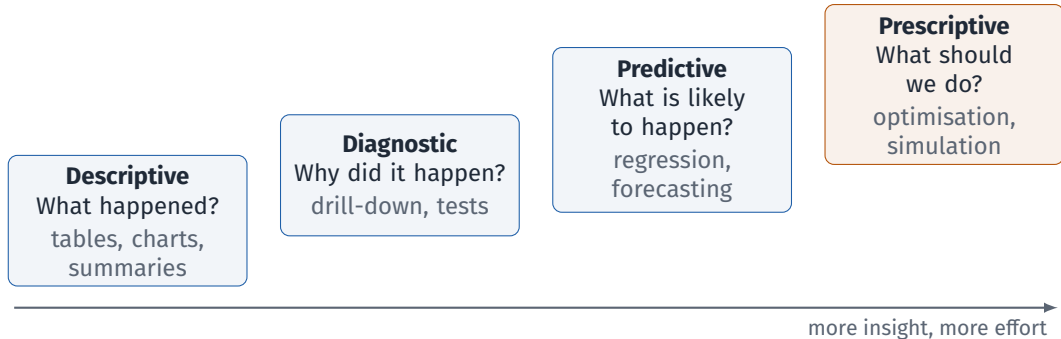
Inferential side estimates for countries with incomplete data and projections of future values, each with an uncertainty range.

What is data analytics?

- **Data analytics** is the process of extracting meaningful patterns from data to support decisions.
- It rests on three foundations: statistics, computer programming and machine learning.
- Results are communicated through visualisations of trends and key metrics.
- Organisations use it to explain, predict and improve their performance.

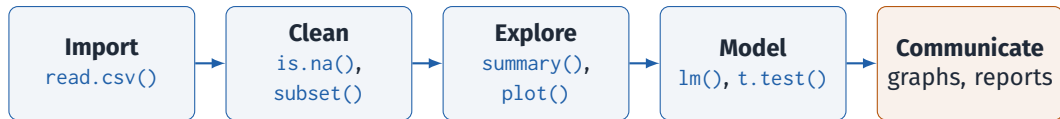
Division of labour Statistics tells us how the data are structured and how far a pattern can be trusted; analytics turns trusted patterns into actionable insight and communicates it to decision makers.

Four types of data analytics



Each type builds on the one before.

Where R fits in



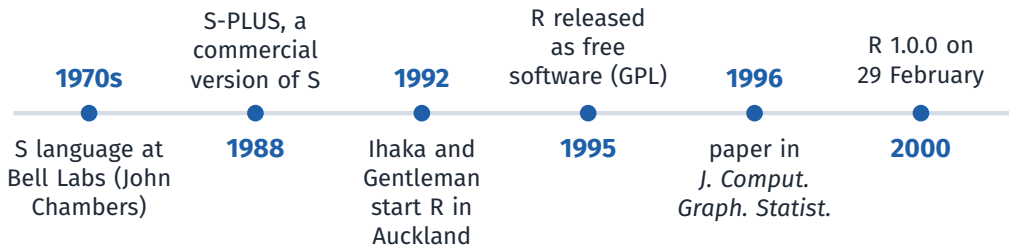
- One free tool covers every step, so the whole analysis is a script that anyone can re-run and check.
- Today we work on the first three steps.

Reproducibility A script records exactly what was done to the data, which a sequence of mouse clicks does not.

PART 2

Getting to know R

A short history of R



S was designed as a programming language for statistics and later earned Chambers the ACM Software System Award. R is “not unlike S” but free; its name comes partly from the first names of its two authors, Ross Ihaka and Robert Gentleman.

Why R?

- Free and open source; runs on Windows, macOS and Linux.
- One suite for data handling, calculation and graphics.
- A full programming language: conditionals, loops, user-defined functions, input and output.
- [CRAN](#), a worldwide network of servers, keeps identical copies of R and its packages: 18,299 packages and 39 Task Views on 27 June 2022.

An environment R is best seen as an environment in which statistical techniques are implemented, not as a fixed menu-driven package.

Limitations Objects live in memory, so very large data need care.

Installing R and RStudio

1. Install R from CRAN:
`cran.r-project.org` → your operating system → *base*.
2. Then install RStudio Desktop (free):
`posit.co/download/rstudio-desktop/`
3. Open RStudio; it finds R automatically.

RStudio A free, open-source integrated development environment (IDE) for R. The company behind it (now Posit) was founded by J. J. Allaire in 2009.

Import wizard *File* → *Import Dataset* reads CSV, Excel, SAS, SPSS and Stata files and shows the R code it used.

The four panes of RStudio

Source (script)

write and save code;
run a line with
Ctrl+Enter

Environment / History

objects in memory:
data, vectors, functions

Console

code is executed here
and output appears

Files / Plots / Packages / Help

graphs, packages,
documentation

Script, not console

Console input is hard to
trace later. Type code in
a script, save it as a .R
file and run it from there.

Check After reading
data, look in the
Environment pane.

Packages, help and our colours

- A **package** adds functions and data; install it once, load it in each session.
- **lattice** ships with R; we use it for one 3-D plot.
- **?mean** opens a help page; **??correlation** searches all pages.
- **example(mean)** runs the examples on a help page.

```
# install.packages("lattice") # once
library(lattice)              # each session

# ?mean                       help on one function
# ??correlation               search the help system
# help(package = "MASS")

# colours used in our plots
col_main <- "#1F5FA8"
col_alt  <- "#B45309"
```

PART 3

First steps in R

R as a calculator

```
2 + 2
3 + 2 * 5      # * before +
(3 + 2) * 5    # brackets first
5 / 2
17 %/% 5       # integer division
17 %% 5        # remainder

## [1] 4
## [1] 13
## [1] 25
## [1] 2.5
## [1] 3
## [1] 2
```

- Order of operations: ^ first, then * and /, then + and -. Use brackets when in doubt.
- # starts a comment: R ignores the rest of the line.
- [1] in the output is the position of the first value on that line.

Assignment `a <- 5` stores 5 under the name `a`; typing `a` prints it.

Generating sequences and repetitions

```
3:6                                # same as c(3, 4, 5, 6)
seq(from = 3, to = 10, by = 0.5)  # step of 0.5
x2 <- seq(from = 3, to = 5, length.out = 20) # 20 equally spaced values
round(x2, 2)
rep(c(2, 3), times = c(5, 4))     # 2 five times, 3 four times
rep(c(1, 2, 3), each = 2)

## [1] 3 4 5 6
## [1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0
## [1] 3.00 3.11 3.21 3.32 3.42 3.53 3.63 3.74 3.84 3.95 4.05 4.16 4.26 4.37 4.47 4.58
## [17] 4.68 4.79 4.89 5.00
## [1] 2 2 2 2 2 3 3 3 3
## [1] 1 1 2 2 3 3
```

Combining vectors

```
x1 <- c(1, 2, 3, 4, 5, 6, 7)
y1 <- c(2, 1, 4, 3, 2, 5, 6)
cbind(x1, y1)      # side by side
```

```
##      x1 y1
## [1,]  1  2
## [2,]  2  1
## [3,]  3  4
## [4,]  4  3
## [5,]  5  2
## [6,]  6  5
## [7,]  7  6
```

```
c(x1, y1)          # one long vector
rbind(x1, y1)       # one above the other
```

```
##  [1] 1 2 3 4 5 6 7 2 1 4 3 2 5 6
##    [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## x1    1    2    3    4    5    6    7
## y1    2    1    4    3    2    5    6
```

One type per vector `c(1:3, "R")`
turns every element into text.

A data vector and its summaries

```
x <- c(2, 4, 3, 5, 7, 9, 15)
length(x); mean(x); median(x)
quantile(x)
summary(x)
order(x)           # positions in sorted order

## [1] 7
## [1] 6.429
## [1] 5
##   0%   25%   50%   75%  100%
##  2.0   3.5   5.0   8.0  15.0
##   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.00   3.50   5.00   6.43   8.00   15.00
## [1] 1 3 2 4 5 6 7
```

Reading it The mean (6.43) exceeds the median (5): the single large value 15 pulls the mean up, a sign of right skew.

- `quantile()` gives the minimum, quartiles and maximum.
- `order()` returns positions, so `x[order(x)]` is the same as `sort(x)`.

Vector arithmetic works element by element

```
x + 1
2 * x
round(sqrt(x), 3)
round(log(x), 3)           # natural logarithm
round(log(x, base = 10), 3)
gamma(x[1:4])              # gamma(n) = (n - 1)!
```



```
## [1]  3  5  4  6  8 10 16
## [1]  4  8  6 10 14 18 30
## [1] 1.414 2.000 1.732 2.236 2.646 3.000 3.873
## [1] 0.693 1.386 1.099 1.609 1.946 2.197 2.708
## [1] 0.301 0.602 0.477 0.699 0.845 0.954 1.176
## [1]  1  6  2 24
```

Vectorisation Each operation is applied to every element at once, so no loop is needed. This makes R code short and fast.

Other functions that work the same way: `exp()`, `abs()`, `sin()`, `cos()`, `factorial()`.

Naming and extracting elements

```
names(x) <- LETTERS[1:7] # "A" to "G"
x
x[c(2, 4)] # second and fourth
x[-2] # all but the second
x[x > 5] # a logical condition

## A B C D E F G
## 2 4 3 5 7 9 15
## B D
## 4 5
## A C D E F G
## 2 3 5 7 9 15
## E F G
## 7 9 15
```

- `[ ]` is the extraction operator: by position, by name, or by a condition. A negative index drops elements.

Naming rules Object names may contain letters, digits, “.” and “_” and must start with a letter (or a dot not followed by a digit). Names are case sensitive: `x` and `X` are different objects.

Reading data from a CSV file

```
book <- read.csv("data/Book1.csv")
head(book, 3)
mean(book$Marks)
book[book$Marks > 17, ]
```

```
##   S.No. Name Roll_No. Marks
## 1     1   A1       10     18
## 2     2   A2       11     20
## 3     3   A3       12     15
## [1] 17.6
##   S.No. Name Roll_No. Marks
## 1     1   A1       10     18
## 2     2   A2       11     20
## 4     4   A4       13     19
```

- `read.csv()` is `read.table()` with `header = TRUE` and `sep = ","`.
- `book$Marks` picks a column; `book[rows, ]` picks rows.
- Other readers: `readLines()` for text, `source()` for R code, `load()` for a saved workspace.

Paths Keep data in a `data/` folder beside the script. A path such as `D:/...` works on one computer only.

Missing values, and writing data back

```
dat <- read.csv("data/data1.csv")
dat
colMeans(dat, na.rm = TRUE) # ignore NAs
```

##	A	B	C	D
## 1	10	12	1	14
## 2	25	24	2	36
## 3	36	36	5	69
## 4	14	85	9	58
## 5	10	12	9	NA

##	A	B	C	D
## 19.00	33.80	5.20	44.25	

Reading it Column D has 1 missing value, so `colMeans(dat)` would give NA for it; with `na.rm = TRUE` its mean uses the 4 recorded values. Ask why a value is missing before ignoring it.

```
write.csv(dat, "data/data1_copy.csv",
          row.names = FALSE)
dim(read.csv("data/data1_copy.csv"))

## [1] 5 4
```

PART 4

Programming in R

Writing your own function

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

- square is the name, `function()` creates it, number is its argument.
- The body sits between { and }; `return()` says what comes back.
- Keep functions in a file such as `myfunctions.R`; load it with `source()`.

A typical mistake `return(number` without its closing bracket gives an “unexpected” syntax error; RStudio marks the line before you run it.

A function for the standard error of the mean

```
set.seed(2022)
y1 <- rnorm(100, mean = 5, sd = 1)
se.mean <- function(x) {
  n <- sum(!is.na(x))           # non-missing count
  sqrt(var(x, na.rm = TRUE) / n)
}
se.mean(y1)
y1[50] <- NA                   # one value goes missing
sqrt(var(y1) / length(y1))     # naive formula fails
se.mean(y1)

## [1] 0.102
## [1] NA
## [1] 0.103
```

$$SE(\bar{x}) = \frac{s}{\sqrt{n}}, \text{ where}$$

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

Why is.na() **length()**
counts NA as a value and has
no na.rm option;
sum(!is.na(x)) counts only
the 99 recorded values.

A function can return several results

```
normal_mle <- function(x) {  
  n <- length(x)  
  sigma2 <- var(x) * (n - 1) / n      # divisor n  
  list(mean_mle = mean(x), sd_mle = sqrt(sigma2))  
}  
set.seed(29)  
z <- rnorm(100, mean = 5, sd = 2)  
normal_mle(z)  
  
## $mean_mle  
## [1] 4.761  
##  
## $sd_mle  
## [1] 2.004
```

MLEs for $N(\mu, \sigma^2)$:

$$\hat{\mu} = \bar{x},$$

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

Reading it The estimates 4.76 and 2.00 are close to the true 5 and 2. `list()` returns both by name.

A loop inside a function: a double sum

```
series_sum <- function(x) {  
  s <- 0  
  for (i in seq_along(x)) {  
    s <- s + sum((1 / x[i] - 1 / x[-i])^2)  
  }  
  s  
}  
series_sum(1:10)  
# check without a loop  
d <- outer(1 / (1:10), 1 / (1:10), "-")  
sum(d^2)  
  
## [1] 13.84  
## [1] 13.84
```

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

- `x[-i]` drops the i -th value, which gives exactly the terms with $j \neq i$.
- `outer()` builds all differences at once; the diagonal is zero, so both answers agree.

Control structures: if and else

```
set.seed(1)
u <- runif(1, min = 0, max = 10)
if (u > 3) {
  v <- 10
} else {
  v <- 0
}
c(u = u, v = v)
# vectorised version for many values
ifelse(c(1, 5, 8) > 3, "high", "low")

##      u      v
## 2.655 0.000
## [1] "low"  "high" "high"
```

- if/else: test a condition, act on it
- for: repeat a fixed number of times
- while: repeat while a condition holds
- break, next: leave a loop, skip one pass

Layout matters Put else on the same line as the closing }; on a new line at the top level it is a syntax error.

for and while loops

```
for (i in 10:13) {  
  print(i^2)  
}  
k <- 0  
while (k^2 < 50) k <- k + 1  
k      # smallest k with k^2 >= 50  
  
## [1] 100  
## [1] 121  
## [1] 144  
## [1] 169  
## [1] 8
```

```
A2 <- matrix(1:6, nrow = 2)  
# seq_len(0) is empty; 1:0 is not  
for (i in seq_len(nrow(A2))) {  
  for (j in seq_len(ncol(A2))) {  
    cat("A2[", i, ",", j, "] =", A2[i, j], "\n")  
  }  
}  
  
## A2[ 1 , 1 ] = 1  
## A2[ 1 , 2 ] = 3  
## A2[ 1 , 3 ] = 5  
## A2[ 2 , 1 ] = 2  
## A2[ 2 , 2 ] = 4  
## A2[ 2 , 3 ] = 6
```

The apply family: rows and columns of a matrix

```
A1 <- matrix(1:20, nrow = 5, byrow = TRUE)
A1
```

```
apply(A1, 1, mean)  # MARGIN = 1: rows
apply(A1, 2, mean)  # MARGIN = 2: columns
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
## [1]  2.5  6.5 10.5 14.5 18.5
## [1]  9 10 11 12
```

```
apply(X, MARGIN, FUN, ...)
```

- Applies FUN to each row (MARGIN = 1) or column (MARGIN = 2) of a matrix.
- Replaces an explicit loop with one readable line.
- Any function works, including our own: `apply(A1, 2, se.mean)`.

lapply() and sapply(): working through a list

```
L <- list(A1 = A1, B1 = 1:5)
lapply(L, mean)      # result is a list
sapply(L, mean)      # simplified to a vector

## $A1
## [1] 10.5
##
## $B1
## [1] 3
##
##      A1      B1
## 10.5  3.0
```

Summary `apply()`: over the margins of a matrix.

`lapply()`: over each element of a list; returns a list.

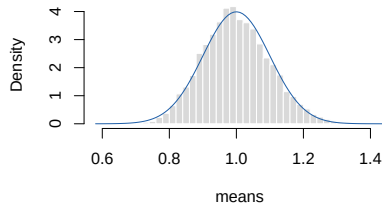
`sapply()`: like `lapply()`, but simplifies the result where it can.

`unlist(lapply(L, mean))` gives the same vector as `sapply()`.

A simulation function: how does the sample mean behave?

```
sim1 <- function(n) mean(rexp(n))      # Exp(1)
set.seed(2022)
means <- replicate(10000, sim1(100))
c(mean = mean(means), sd = sd(means))
par(mar = c(4, 4, 1, 1))
hist(means, freq = FALSE, breaks = 40, main = "",
     col = "grey85", border = "white")
curve(dnorm(x, 1, 0.1), add = TRUE, col = col_main)

##    mean    sd
## 1.0000 0.1005
```



Reading it Exp(1) is skewed, yet the means have mean 1.000, SD 0.101 (theory 1, 0.1) and look normal: the CLT.

PART 5

Basic statistics and graphics

Mean, trimmed mean and median

```
x <- c(2, 4, 3, 5, 7, 9, 15)
mean(x)
mean(x, trim = 0.2) # 20% off each end
mean(x, trim = 0.5) # gives the median
median(x)

## [1] 6.429
## [1] 5.6
## [1] 5
## [1] 5
```

Syntax: `mean(x, trim = 0, na.rm = FALSE)`,
`median(x, na.rm = FALSE)`.

$\bar{x} = \frac{1}{n} \sum x_i$. The α -trimmed mean averages what is left after removing a fraction α of the values from each end.

Reading it With $n = 7$, `trim = 0.2` removes one value from each end and gives 5.6, between the mean and the median. `trim` lies in $[0, 0.5]$; 0.5 or more returns the median.

Mode, coefficient of variation and standard error

```
# base R's mode() gives the storage type
mode_value <- function(x) {
  u <- unique(x)
  u[which.max(tabulate(match(x, u)))]
}
A3 <- c(2, 1, 2, 3, 1, 2, 3, 4, 1, 5, 5, 3, 2, 3)
table(A3)
mode_value(A3)

## A3
## 1 2 3 4 5
## 3 4 4 1 2
## [1] 2
```

```
cv <- function(x) 100 * sd(x) / mean(x)
cv(A3) # CV in %
sd(A3) / sqrt(length(A3)) # SE of mean

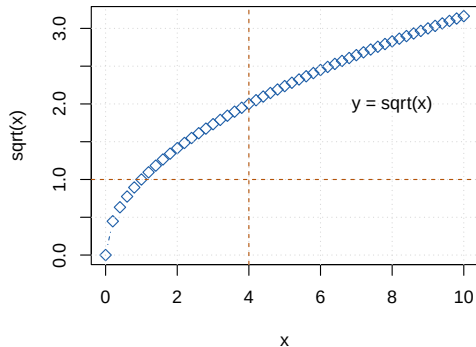
## [1] 50.56
## [1] 0.3571
```

Look at the table 2 and 3 both occur four times: A3 is bimodal, but `which.max()` returns only the first maximum.

High-level and low-level graphics

```
x <- seq(0, 10, by = 0.2)
y <- sqrt(x)
par(mar = c(4, 4, 1, 1))
plot(x, y, type = "b", pch = 5, lty = 4,
      col = col_main, ylab = "sqrt(x)")
grid()
abline(h = 1, v = 4, col = col_alt, lty = 2)
text(8, 2, "y = sqrt(x)")
```

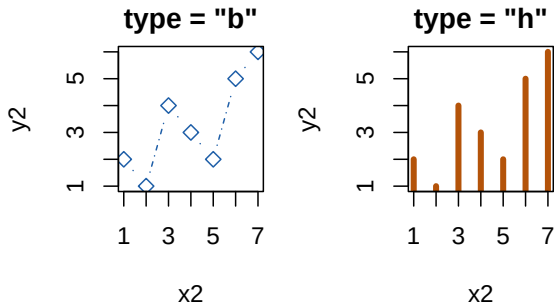
- High level (new plot): `plot()`, `hist()`, `boxplot()`, `barplot()`
- Low level (add to it): `lines()`, `points()`, `abline()`, `text()`, `grid()`, `legend()`



Plot types, symbols and line styles

```
x2 <- 1:7
y2 <- c(2, 1, 4, 3, 2, 5, 6)
par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))
plot(x2, y2, type = "b", pch = 5, lty = 4,
     col = col_main, main = 'type = "b"')
plot(x2, y2, type = "h", lwd = 4,
     col = col_alt, main = 'type = "h"')
```

type: "p" points, "l" lines, "b" both,
"h" spikes, "s" steps, "n" axes only.

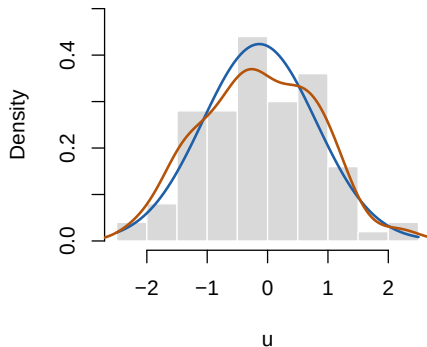


pch: symbol (0–25); lty: line type (1–6); lwd:
width; col: a name (colors() lists 657) or a hex
code.

Histogram with a fitted normal curve

```
set.seed(10)
u <- rnorm(100)
par(mar = c(4, 4, 1, 1))
hist(u, freq = FALSE, breaks = 15, ylim = c(0, 0.5),
     main = "", col = "grey85", border = "white")
curve(dnorm(x, mean(u), sd(u)), add = TRUE,
      col = col_main, lwd = 2) # normal fit
lines(density(u), col = col_alt, lwd = 2) # density
```

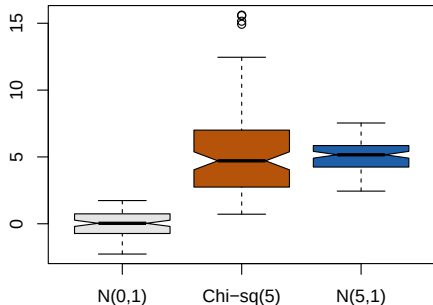
Two overlays `freq = FALSE` puts the bars on the density scale. Blue: normal curve with the sample mean and SD. Amber: a kernel density estimate.



Comparing distributions with box plots

```
set.seed(3)
u1 <- rnorm(100)           # N(0, 1)
u2 <- rchisq(100, df = 5)  # mean 5
u3 <- rnorm(100, 5, 1)     # N(5, 1)
par(mar = c(4, 4, 1, 1))
boxplot(u1, u2, u3, notch = TRUE,
        names = c("N(0,1)", "Chi-sq(5)", "N(5,1)"),
        col = c("grey90", col_alt, col_main))
```

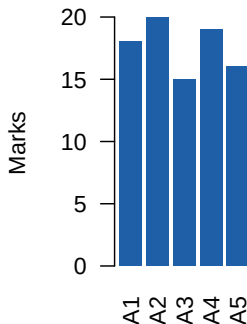
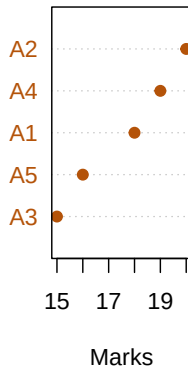
Reading it The chi-square sample is right-skewed: a long upper whisker and points beyond it. Notches that do not overlap suggest different medians.



Bar charts and dot charts

```
marks <- book$Marks
names(marks) <- book$Name
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1))
barplot(marks, col = col_main, border = NA,
        las = 2, ylab = "Marks")
dotchart(sort(marks), pch = 19,
         col = col_alt, xlab = "Marks")
```

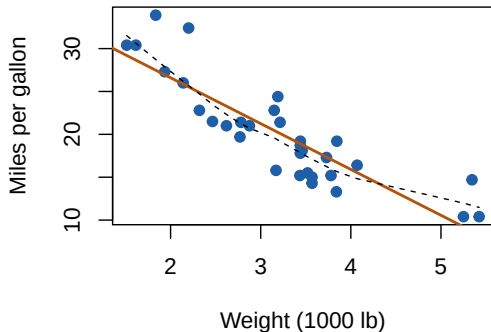
Data: Book1.csv, read earlier (students coded A1–A5). A dot chart of sorted values is easier to read than bars when the categories have no natural order.



Scatter plot with fitted lines

```
par(mar = c(4, 4, 1, 1))
plot(mpg ~ wt, data = mtcars, pch = 19,
     col = col_main, xlab = "Weight (1000 lb)",
     ylab = "Miles per gallon")
fit <- lm(mpg ~ wt, data = mtcars)
abline(fit, col = col_alt, lwd = 2)
lines(lowess(mtcars$wt, mtcars$mpg), lty = 2)
```

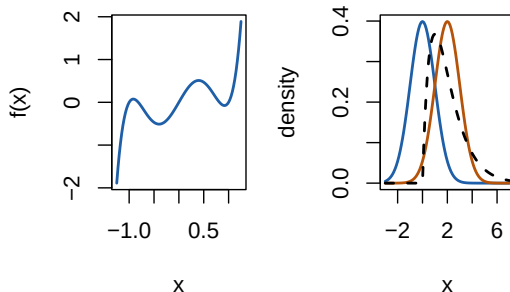
Reading it The fitted line is $\widehat{\text{mpg}} = 37.29 - 5.34 \text{ wt}$: about 5.34 fewer mpg per extra 1000 lb.



The dashed `lowess()` curve flattens for the heaviest cars.

Drawing functions and densities with `curve()`

```
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1))
curve(3 * x^5 - 5 * x^3 + 2 * x, -1.25, 1.25,
      lwd = 2, col = col_main, ylab = "f(x)")
curve(dnorm(x), -3, 7, lwd = 2,
      col = col_main, ylab = "density")
curve(dnorm(x, mean = 2), add = TRUE,
      lwd = 2, col = col_alt)
curve(dgamma(x, shape = 2, scale = 1),
      add = TRUE, lwd = 2, lty = 2)
```

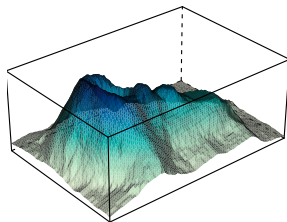


`curve()` plots an expression in x . Right:
 $N(0, 1)$ blue, $N(2, 1)$ amber, $\text{Gamma}(2, 1)$
dashed.

A 3-D surface with lattice

```
dim(volcano)      # heights on a 10 m grid
wireframe(volcano, shade = TRUE, zoom = 1.15,
  aspect = c(61/87, 0.4), zlab = "",
  xlab = "", ylab = "",
  par.settings = list(axis.line = list(col = NA)))

## [1] 87 61
```



Data volcano, built into R, holds heights (m) of Maunga Whau in Auckland, the city where R was born.

Key take-aways

1. **Describe before you infer:** descriptive statistics summarise the data; inferential statistics generalise from a sample with a measure of uncertainty.
2. **Analytics climbs a ladder:** descriptive, diagnostic, predictive, prescriptive.
3. **Work in a script** inside an RStudio project, with relative paths to data/.
4. **Think in vectors:** arithmetic, summaries and `apply()` act on whole objects without loops.
5. **Write small functions** and test them, including on missing values.
6. **Plot first:** a histogram, box plot or scatter plot often shows what a summary number hides.

References

- Dalgaard, P. (2008). *Introductory Statistics with R*, 2nd ed. Springer.
- Ihaka, R. and Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3), 299–314.
- Jones, O., Harden, S. and Crawley, M. J. (2022). *The R Book*, 3rd ed. Wiley.
- Verzani, J. (2014). *Using R for Introductory Statistics*, 2nd ed. CRC Press.
- Wickham, H., Çetinkaya-Rundel, M. and Grolemund, G. (2023). *R for Data Science*, 2nd ed. O'Reilly. Free online: r4ds.hadley.nz
- R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna. <https://www.R-project.org>
- Web: CRAN <https://cran.r-project.org>; RStudio Desktop <https://posit.co/download/rstudio-desktop/>

Thank you

Questions and discussion