An Introduction to R

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

ICSSR-sponsored Research Methodology Course:
A Comprehensive Research Methodology Course for Research Scholars of Social Science (18–27 February 2025)

Centre for International Trade and Development, School of International Studies, Jawaharlal Nehru University, New Delhi

26 February 2025 · Session 1 of 2

Plan of the session

1. Getting started
2. R basics: objects and vectors
3. Importing and exporting data
4. The apply family
5. Graphics with base R

By the end you can install R and RStudio, work with vectors and data frames, read and write CSV, text and Excel files, summarise data without loops, and draw and annotate the standard graphs.

Session 2 applies these tools to a real air-quality data set from Delhi.

PART 1

Getting started

What is R?

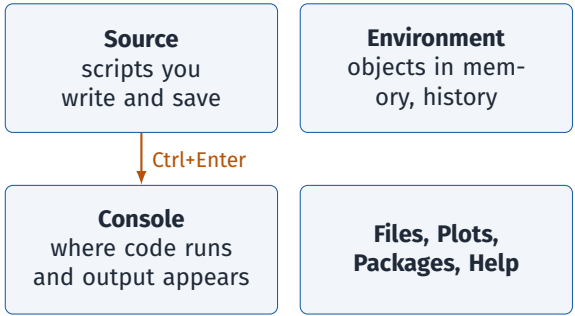
- A language *and* an environment for data, calculation and graphics
- Free, open source, on every major platform
- Grew out of **S** (Bell Labs, 1970s); begun by Ross Ihaka and Robert Gentleman in Auckland; version 1.0.0 on 29 February 2000
- CRAN listed 22,079 packages on 20 February 2025, grouped by field in Task Views

Why researchers use it A script is a complete, re-runnable record of the analysis: examiners and co-authors can reproduce every table and figure.

Trade-off Commands instead of menus: a steeper start, but nothing is hidden.

Installing R and RStudio

1. Download and install R from CRAN:
   `cran.r-project.org`
2. Then install RStudio Desktop (free) from Posit:
   `posit.co/download/rstudio-desktop`
3. Start RStudio, not R: it finds R by itself



Getting help

```
?mean                    # help page of a function
help("cor")              # the same, as a function
??correlation             # search all help pages
help.search("regression")
help(package = "MASS")   # what a package offers
example(mean)             # run the examples
```

- Every help page has the same parts: Description, Usage, Arguments, Value, Examples. Read Usage and Examples first
- Manuals: RStudio menu *Help* → *R Help* ("An Introduction to R")
- Web: `rseek.org` searches R sites only; Springer's *Use R!* book series covers many fields

Packages and colours used in this session

- A **package** adds functions and data; install once, load in every session
- **`readr`**, **`data.table`**: fast reading of text files
- **`readxl`**, **`writexl`**: Excel files, no Excel needed
- Graphics today use base R; two colours are defined once

```r
# install once
# install.packages(c("readr", "data.table",
#                    "readxl", "writexl"))
library(readr)
library(data.table)
library(readxl)
library(writexl)
col_main <- "#1F5FA8"   # blue
col_alt  <- "#B45309"   # amber
```

PART 2

R basics: objects and vectors

R as a calculator

```
2 + 2; 3 + 2 * 5; (2 + 3) * 2; 5 / 2

## [1] 4
## [1] 13
## [1] 10
## [1] 2.5

5^2; 7 %/% 2; 7 %% 2; log(100, base = 10)

## [1] 25
## [1] 3
## [1] 1
## [1] 2
```

Order: `^` then `*` `/` then `+` `-`; brackets first.
`%/%` integer division, `%%` remainder

- `x <- 5` stores a value in the object `x`
- Names start with a letter (or a dot not followed by a digit) and may contain letters, digits, `.` and `_`
- R is case sensitive: `x` and `X` differ

Creating a data vector

```
x <- c(2, 4, 3, 5, 7, 9, 15)  # c() combines
c(n = length(x), sum = sum(x), mean = mean(x),
  median = median(x), min = min(x), max = max(x))

##      n    sum   mean median    min    max
##  7.000 45.000  6.429  5.000  2.000 15.000

summary(x)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    3.50    5.00    6.43    8.00   15.00

sort(x, decreasing = TRUE)

## [1] 15  9  7  5  4  3  2
```

- A **vector** holds values of one type: numbers, text or logical
- Summary functions take the whole vector at once
- # starts a comment: R ignores the rest of the line

Reading it Mean 6.43 $>$ median 5: the single large value 15 pulls the mean up.

Vector arithmetic works element by element

```
x + 1; 2 * x; x^2; round(log(x), 2)

## [1]  3  5  4  6  8 10 16
## [1]  4  8  6 10 14 18 30
## [1]   4  16   9  25  49  81 225
## [1] 0.69 1.39 1.10 1.61 1.95 2.20 2.71

round(100 * x / sum(x), 1)   # percentage shares

## [1]  4.4  8.9  6.7 11.1 15.6 20.0 33.3

c(1, 2, 3, 4) + c(10, 20)    # recycling

## [1] 11 22 13 24
```

- No loop is needed: `2 * x` doubles every element
- Functions such as `log()`, `sqrt()`, `round()` are *vectorised*
- Shares of a total in one line: `x / sum(x)`

Recycling A shorter vector is reused; R warns only when the longer length is not a multiple of the shorter one.

Sequences and repetition

```
3:6; seq(3, 6, by = 0.5); seq(3, 5, length.out = 5)

## [1] 3 4 5 6
## [1] 3.0 3.5 4.0 4.5 5.0 5.5 6.0
## [1] 3.0 3.5 4.0 4.5 5.0

rep(x = c(2, 3), times = c(5, 4))

## [1] 2 2 2 2 2 3 3 3 3

rep(1:3, times = 2); rep(1:3, each = 2)

## [1] 1 2 3 1 2 3
## [1] 1 1 2 2 3 3
```

- `a:b` counts in steps of 1
- `seq()` sets the step (`by`) or the number of points (`length.out`)
- `rep()` repeats a whole vector (`times`) or each element (`each`)

Typical uses: questionnaire item numbers, years of a panel, group labels for an experiment.

Selecting and naming elements

```
x[2]; x[c(2, 4)]; x[-2]  # by position

## [1] 4
## [1] 4 5
## [1]  2  3  5  7  9 15

x[x > 5]                 # by a condition

## [1]  7  9 15

names(x) <- LETTERS[1:7] # labels A to G
x[c("A", "C")]

## A C
## 2 3
```

- `[ ]` is the extraction operator; positions start at 1
- A negative index *removes* elements
- A logical condition keeps the elements where it is `TRUE`
- `names()` attaches labels, so elements can be picked by name

Combining vectors: `c()`, `cbind()`, `rbind()`

```
x1 <- c(1, 2, 3, 4, 5, 6, 7)
y1 <- c(2, 1, 4, 3, 2, 5, 6)
c(x1, y1)             # one long vector

##  [1] 1 2 3 4 5 6 7 2 1 4 3 2 5 6

rbind(x1, y1)         # two rows: a matrix

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

dim(cbind(x1, y1))    # 7 rows, 2 columns

## [1] 7 2
```

- `c()` joins end to end
- `cbind()` and `rbind()` bind as columns or rows and return a **matrix**
- `dim()` gives the numbers of rows and columns

> **Objects you will meet** vector, matrix, data frame, list, factor.

Data frames and factors: the `swiss` data

```
str(swiss[, 1:4], vec.len = 2)

## 'data.frame': 47 obs. of  4 variables:
##  $ Fertility  : num  80.2 83.1 92.5 85.8 76.9 ...
##  $ Agriculture: num  17 45.1 39.7 36.5 43.5 ...
##  $ Examination: int  15 6 5 12 17 ...
##  $ Education  : int  12 9 5 7 15 ...

swiss$majority <- factor(ifelse(swiss$Catholic > 50,
                                "Catholic", "Protestant"))
summary(swiss$majority)

##   Catholic Protestant
##         18         29
```

- Built-in classic data: fertility and socio-economic indicators for 47 French-speaking Swiss provinces, about 1888; two more columns are `Catholic` and `Infant.Mortality`
- A **data frame**: one row per case, one column per variable; columns may differ in type
- A **factor** stores a categorical variable with its levels

PART 3

Importing and exporting data

Where R reads and writes

- Keep the script and a `data/` folder together in one RStudio **project**
- Use relative paths such as `"data/ito.csv"`; drive-letter paths break on every other computer
- `getwd()` shows the current working folder

File type	Read / write with
CSV	`read.csv()`, `readr::read_csv()`
Text	`read.table()`, `data.table::fread()`
Excel	`readxl::read_excel()`
Write CSV	`write.csv()`, `readr::write_csv()`
Write Excel	`writexl::write_xlsx()`

RStudio's *Import Dataset* menu writes the same code for you.

Reading a CSV file

```
jnu <- read.csv("data/data_jnu.csv",
                fileEncoding = "UTF-8-BOM")
dim(jnu); colMeans(jnu)

## [1] 10  4
##    A    B    C    D
## 33.7 26.2 27.7 44.4

head(jnu, 3)

##    A  B  C  D
## 1 10 12 15 20
## 2 20 10 14 18
## 3 15 25 58 47
```

- `read.csv()` returns a data frame; line 1 holds the column names
- Illustrative data made for the course: 10 rows, columns A–D
- `fileEncoding` strips the invisible byte-order mark of Excel's "CSV UTF-8" files

> **Always check** `dim()`, `names()` and `head()` right after reading.

The same file with `readr`

```
jnu_tbl <- read_csv("data/data_jnu.csv",
                    show_col_types = FALSE)
print(jnu_tbl, n = 3)

## # A tibble: 10 x 4
##       A     B     C     D
##   <dbl> <dbl> <dbl> <dbl>
## 1    10    12    15    20
## 2    20    10    14    18
## 3    15    25    58    47
## # i 7 more rows
```

- `read_csv()` returns a **tibble**: a data frame that prints compactly and shows each column's type (`<dbl>` = number)
- Faster on large files; handles the byte-order mark itself
- `spec(jnu_tbl)` lists the column types it chose

> **Note the underscore** `read.csv()` is base R; `read_csv()` is **readr**.

Text files: `read.table()` and `fread()`

```
scores <- read.table("data/data3.txt",
                     header = TRUE)
str(scores, vec.len = 2)

## 'data.frame': 10 obs. of  2 variables:
##  $ score: num  20 20.5 13 16 11 ...
##  $ group: chr  "batchA" "batchA" ...

write.table(scores, "data/scores.txt", sep = "\t",
            row.names = FALSE)
scores_dt <- fread("data/scores.txt")
class(scores_dt); dim(scores_dt)

## [1] "data.table" "data.frame"
## [1] 10  2
```

- `data3.txt`: test scores of two batches of five students, separated by spaces (data from the lecture)
- `write.table()` with `sep = "\t"` saves a tab-separated copy
- `fread()` detects the separator and column types by itself, and reads files with millions of rows in seconds

Excel workbooks

```
write_xlsx(scores, "data/scores.xlsx")
excel_sheets("data/scores.xlsx")

## [1] "Sheet1"

sc_xl <- read_excel("data/scores.xlsx",
                    sheet = 1)
head(sc_xl, 3)

## # A tibble: 3 x 2
##   score group
##   <dbl> <chr>
## 1  20   batchA
## 2  20.5 batchA
## 3  13   batchA
```

- `read_excel()` reads `.xls` and `.xlsx`; choose the sheet by number or name, and a cell block with `range = "B2:D20"`
- `write_xlsx()` writes a data frame (or a named list of them, one per sheet)

Tip Merged cells, colours and notes in a spreadsheet do not survive import: keep one rectangular table per sheet.

Exporting results

```
batch_means <- aggregate(score ~ group,
                         data = scores, FUN = mean)
batch_means

##    group score
## 1 batchA  16.1
## 2 batchB  46.6

write.csv(batch_means, "data/output.csv",
          row.names = FALSE)
read.csv("data/output.csv")      # verify

##    group score
## 1 batchA  16.1
## 2 batchB  46.6
```

- `aggregate()` summarises a variable by group (formula `score ~ group`)
- `write.csv()` saves a table for Excel or a manuscript; `row.names = FALSE` avoids an extra column
- Reading the file back confirms what was written

A first look at a real data set

```
ito <- read.csv("data/ito.csv")
dim(ito); class(ito$Date)

## [1] 659  11
## [1] "character"

head(ito[, 1:5], 3); colSums(is.na(ito[, 2:5]))

##         Date  PM25  PM10    NO   NO2
## 1 01-01-2023 138.7 167.3 142.8 47.21
## 2 02-01-2023 292.1 361.4 139.3 52.09
## 3 03-01-2023 284.1 366.8 103.1 36.83
## PM25 PM10   NO  NO2
##   20   20   17   17
```

- Daily air-pollution readings at the ITO monitoring station, Delhi (Central Pollution Control Board)
- `Date` arrives as *text*, not as a date
- Some days are missing

Session 2 turns these 659 rows into a research analysis: dates, missing values, exploration, tests and a model.

PART 4

The apply family

`apply()`: rows or columns of a matrix

```
mat <- matrix(1:12, nrow = 3, ncol = 4)
mat

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

apply(mat, 1, mean); apply(mat, 2, mean)

## [1] 5.5 6.5 7.5
## [1]  2  5  8 11
```

`apply(X, MARGIN, FUN, ...)`
`X`: matrix or data frame; `MARGIN`: 1 rows, 2 columns; `FUN`: the function to apply

Row means first, then column means. On a data frame, `apply(swiss[, 1:6], 2, median)` gives every indicator's median in one call; `rowMeans()` and `colMeans()` are faster shortcuts for means.

`lapply()`, `sapply()` and `vapply()`: over a list

```
lst <- list(a = 1:3, b = 4:6)
str(lapply(lst, function(v) v^2))  # a list

## List of 2
##  $ a: num [1:3] 1 4 9
##  $ b: num [1:3] 16 25 36

sapply(lst, sum); vapply(lst, length, numeric(1))

##  a  b
##  6 15
## a b
## 3 3
```

- `lapply()` applies a function to each element and always returns a **list**
- `sapply()` simplifies the result to a vector or matrix when it can
- `vapply()` also states the result type (`numeric(1)`: one number), so it stops with an error rather than returning something unexpected
- A data frame is a list of columns: `sapply(swiss, mean)` gives column means

`tapply()`: a summary for each group

```
marks  <- c(85, 90, 78, 92, 88, 76)
groups <- factor(c("A", "A", "B", "B", "A", "B"))
tapply(marks, groups, mean)

##     A     B
## 87.67 82.00

round(tapply(swiss$Fertility, swiss$majority,
             mean), 1)

##   Catholic Protestant
##       76.5       66.2
```

`tapply(X, INDEX, FUN)`: split `X` by the factor `INDEX`, then apply `FUN` to each piece

Reading it Mostly Catholic provinces had a higher average fertility index (76.5) than mostly Protestant ones (66.2): a description, not yet a test.

`mapply()`: several vectors in parallel

```r
a <- 1:4
b <- 5:8
mapply(sum, a, b)        # 1+5, 2+6, ...

## [1]  6  8 10 12

mapply(function(n, p) round(n * p),
       c(200, 150, 90), c(0.42, 0.35, 0.60))

## [1] 84 52 54
```

- `mapply()` takes the first elements of all vectors, then the second ones, and so on
- Second example: expected number of “yes” answers in three survey strata from their sizes and proportions
- For simple arithmetic, `a + b` or `n * p` is shorter: vectorised operators come first

Which one to use?

Function	Works over	Returns
`apply(X, MARGIN, FUN)`	rows or columns of a matrix	vector or matrix
`lapply(X, FUN)`	elements of a list or vector	list
`sapply(X, FUN)`	elements of a list or vector	vector or matrix if possible
`vapply(X, FUN, FUN.VALUE)`	elements of a list or vector	vector of a stated type
`tapply(X, INDEX, FUN)`	groups defined by a factor	one value per group
`mapply(FUN, ...)`	several vectors in parallel	vector or list

Rule of thumb First look for a vectorised function (`colMeans()`, `x / sum(x)`); then the apply family; write an explicit `for` loop only when each step depends on the previous one.

PART 5

Graphics with base R

The base graphics toolbox

Question	Graph	Function
Do two variables relate?	scatter	`plot(x, y)`
How is one variable spread?	histogram	`hist()`
Do groups differ?	box plot	`boxplot(y ~ g)`
Change over time?	line plot	`plot(type = "l")`
How many per category?	bar chart	`barplot()`
Many named cases?	dot chart	`dotchart()`

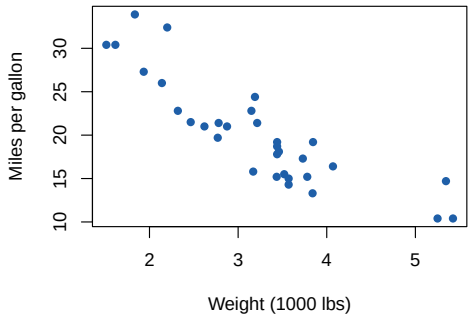
Two kinds of function
High-level functions draw a new graph.
Low-level functions add to the current one: `points()`, `lines()`, `abline()`, `text()`, `legend()`.

Data: built-in `mtcars`, 32 cars (1973–74 models).

Scatter plot

```r
par(mar = c(4, 4, 1, 1))
plot(mtcars$wt, mtcars$mpg,
     pch = 16, col = col_main,
     xlab = "Weight (1000 lbs)",
     ylab = "Miles per gallon")
```

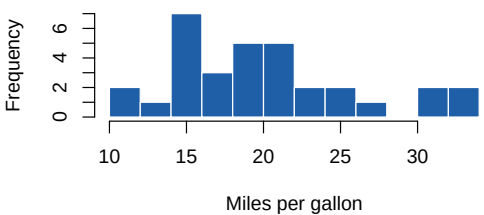
- `pch = 16`: solid circles
- `xlab`, `ylab`: always label axes with units



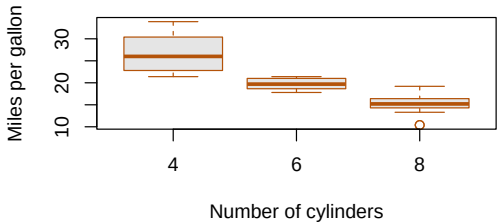
Heavier cars travel fewer miles per gallon: a strong negative relationship, $r = -0.87$.

Histogram and box plot

```r
par(mar = c(4, 4, 1, 1))
hist(mtcars$mpg, breaks = 10, main = "",
     col = col_main, border = "white",
     xlab = "Miles per gallon",
     ylab = "Frequency")
```




```r
par(mar = c(4, 4, 1, 1))
boxplot(mpg ~ cyl, data = mtcars,
        col = "grey90", border = col_alt,
        xlab = "Number of cylinders",
        ylab = "Miles per gallon")
```



Reading the two graphs

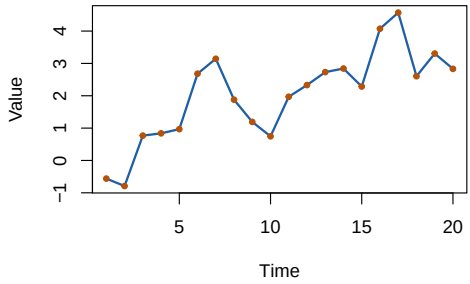
Histogram Fuel economy ranges from 10.4 to 33.9 mpg and is slightly right-skewed: a few very economical cars. `breaks = 10` is only a suggestion; R picks “pretty” class limits near it.

Box plot The formula `mpg ~ cyl` draws one box per group. Median mpg falls from 26 (4 cylinders) to 15.2 (8 cylinders), and the groups barely overlap.

Box plot anatomy The box spans the quartiles Q_1 to Q_3, the line inside is the median, whiskers reach the most extreme values within $1.5 \times \mathrm{IQR}$ of the box, and points beyond are drawn individually.

Line plot

```r
set.seed(123)
walk <- cumsum(rnorm(20))  # a random walk
par(mar = c(4, 4, 1, 1))
plot(walk, type = "l", lwd = 2,
     col = col_main, xlab = "Time",
     ylab = "Value")
points(walk, pch = 20, col = col_alt)
```



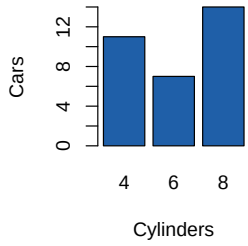
`type = "l"` joins points in order; `points()` adds the observations; `set.seed()` makes the simulation reproducible.

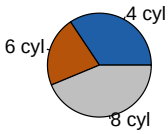
Bar chart and pie chart

```
cyl_counts <- table(mtcars$cyl); cyl_counts

##
##  4  6  8
## 11  7 14

par(mfrow = c(1, 2), mar = c(4, 4, 1, 1))
barplot(cyl_counts, col = col_main,
        xlab = "Cylinders", ylab = "Cars")
pie(cyl_counts, labels = paste(
      names(cyl_counts), "cyl"),
    col = c(col_main, col_alt, "grey75"))
```

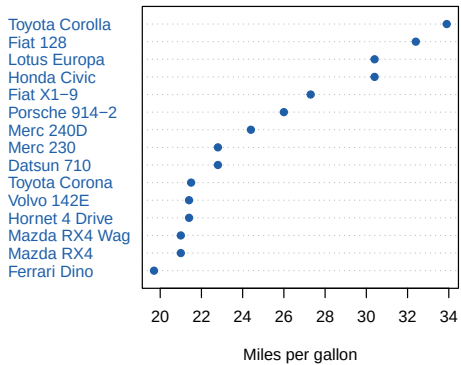




Prefer the bar chart Lengths are judged more accurately than angles; keep pies for a few parts of one whole, never in 3-D.

Dot chart

```
mt <- mtcars[order(mtcars$mpg), ]
top15 <- tail(mt, 15)    # 15 highest mpg
par(mar = c(4, 1, 1, 1))
dotchart(top15$mpg,
         labels = row.names(top15),
         cex = 0.8, pch = 19,
         col = col_main,
         xlab = "Miles per gallon")
```

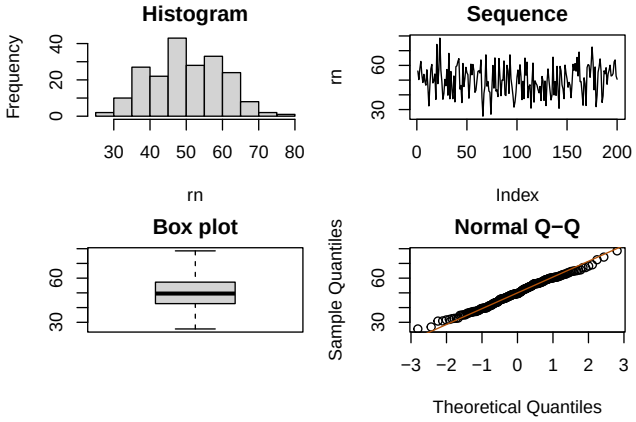


A Cleveland dot chart replaces bars by dots, so many named cases fit. Sorting first with `order()` turns it into a ranking.

Several plots on one page with `par()`

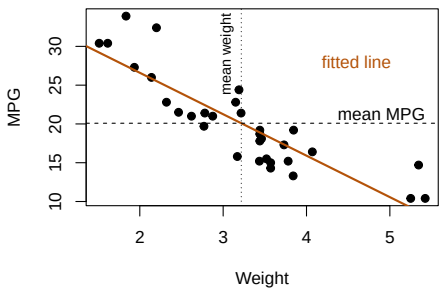
```r
set.seed(2025)
rn <- rnorm(200, mean = 50, sd = 10)
par(mfrow = c(2, 2),
    mar = c(4, 4, 2, 1))
hist(rn, main = "Histogram")
plot(rn, type = "l", main = "Sequence")
boxplot(rn, main = "Box plot")
qqnorm(rn, main = "Normal Q-Q")
qqline(rn, col = col_alt)
```

`mfrow = c(2, 2)`: a 2×2 grid, filled by rows. Q–Q points on the line: normal data.



Adding annotations

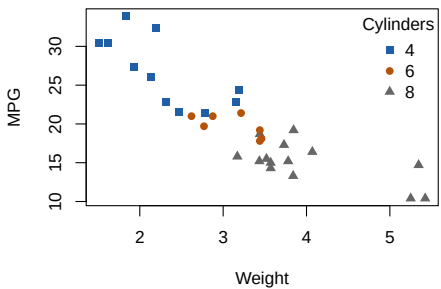
```
fit <- lm(mpg ~ wt, data = mtcars)
par(mar = c(4, 4, 1, 1))
plot(mtcars$wt, mtcars$mpg, pch = 19,
     xlab = "Weight", ylab = "MPG")
abline(fit, col = col_alt, lwd = 2)
abline(h = mean(mtcars$mpg), lty = 2)
abline(v = mean(mtcars$wt), lty = 3)
text(4.6, 28, "fitted line", col = col_alt)
text(4.9, 21.3, "mean MPG")
text(3.05, 29, "mean weight", srt = 90,
     cex = 0.8)
```



`abline()` adds fitted or reference lines (`lty`: 2 dashed, 3 dotted); `text()` writes at (x, y); `srt = 90` rotates.

Groups, colours and a legend

```r
grp  <- factor(mtcars$cyl)
cols <- c(col_main, col_alt, "grey40")
shps <- c(15, 16, 17)
par(mar = c(4, 4, 1, 1))
plot(mtcars$wt, mtcars$mpg,
     pch = shps[grp], col = cols[grp],
     xlab = "Weight", ylab = "MPG")
legend("topright", title = "Cylinders",
       legend = levels(grp), pch = shps,
       col = cols, bty = "n")
```



Indexing `shps` and `cols` by the factor gives each group its own symbol and colour; `legend()` must reuse the *same* vectors.

Plot options and saving a graph

Argument	Controls
`type`	`"p"`, `"l"`, `"b"` (both)
`pch`	symbol (1–25)
`lty`, `lwd`	line type and width
`col`	colour; see `colors()`
`main`, `xlab`	title, axis label
`xlim`, `ylim`	axis ranges
`cex`	text and symbol size

```
pdf("weight_mpg.pdf", width = 6, height = 4)
plot(mtcars$wt, mtcars$mpg, pch = 19)
dev.off()               # closes the file

png("weight_mpg.png", width = 1800,
    height = 1200, res = 300)
plot(mtcars$wt, mtcars$mpg, pch = 19)
dev.off()
```

PDF scales without blurring (best for a thesis); a 300 dpi PNG suits Word and slides.

Key take-aways

1. **Work in scripts and projects**: relative paths and saved code make the analysis reproducible.
2. **Think in vectors**: arithmetic, summaries and selection act on whole vectors.
3. **Check every import** with `dim()`, `head()` and `str()`: types, dates and missing values.
4. **Prefer vectorised functions and the apply family** to hand-written loops; `tapply()` gives group summaries in one line.
5. **Choose the graph from the question**, label axes with units, and keep legends consistent with the plot.

Next session Application of R for data analysis in social science research: the ITO air-quality data from research question to model.

References

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

Thank you

Questions and discussion