# Statistical Techniques using R

## Session 1: An Introduction to R

Dr. Kapil Kumar

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

22 June 2018

One-month Faculty Induction Training Programme for faculty of universities, colleges and institutes (14 June – 14 July 2018)

Pandit Madan Mohan Malaviya National Mission on Teachers and Teaching (PMMMNMTT), Central University of Haryana, Mahendergarh

**Keynote lecture**

# Plan of the session

1. Getting started with R and RStudio
2. Vectors, lists and data frames
3. Graphics
4. Comparing groups: $t$-tests and ANOVA
5. Correlation, regression and functions of your own

No programming experience is assumed. Every line of code on these slides runs as a plain script, from top to bottom.

**By the end you can** create and summarise data in R, import a CSV file, draw basic graphs, run $t$-tests and a one-way ANOVA, fit a regression line, and write a small function of your own.

**Session 2** builds on this one: simple linear regression modelling in depth.

# Outline


1. Getting started
2. Vectors and data frames
3. Graphics
4. Comparing groups
5. Regression and functions

# What is R?

- A language and environment for statistical computing and graphics, in one integrated suite
- Created by Ross Ihaka and Robert Gentleman (University of Auckland); the name comes partly from their first names
- Conceived in 1992, first released in 1995; version 1.0.0 (the first stable release) in February 2000; now maintained by the R Core Team
- Free and open source; extended by thousands of packages on CRAN

**Why it suits a teacher** Free for you and your students. Every analysis is a script that can be rerun, checked and shared, and the graphics are of publication quality.

**RStudio** A free integrated development environment (IDE) that makes R easier to use. Install R first, then RStudio.

# RStudio: four panes

**Source (editor)**
write and save your script;
Ctrl+Enter runs the current line

**Environment / History**
the objects you have created
and the commands you have run

**Console**
code is executed here
and results are printed

**Files / Plots / Packages / Help**
browse the folder, view graphs,
install packages, read help

**Habit** Type code in the *script*, not in the console: the script is your record of the analysis.

# Your first project

- Make a folder for the course, e.g. `lecture/`, with a sub-folder `data/`
- RStudio: *File* → *New Project* → *Existing Directory*. The folder becomes the working directory
- *File* → *New File* → *R Script*; save it in the folder
- In Excel, save your data as CSV (comma-separated values) inside `data/`
- Read files with relative paths such as `"data/D1.csv"`, never with a full path that exists only on your own computer

**Folder layout**

```
lecture/
  intro.R
  data/
    D1.csv
```

**Literate files** The handout for this session was a Sweave (`.Rnw`) file, in which text and R code live together. These slides are made the same way.

# Packages used in this session

- Base R and its built-in data sets (`trees`, `CO2`) need no installation
- **`arm`**: `display()` prints a compact regression summary
- A package is *installed* once, but *loaded* with `library()` in every new session

```
# install once (needs internet)
# install.packages("arm")
library(arm)    # also loads MASS
```

# Getting help

```
help(cor)                # or ?cor
help(package = "MASS") # a package index
example("aov")           # run the examples
help(CO2)                # about a data set
```

Every help page has the same parts: *Description*, *Usage*, *Arguments*, *Value*, *Examples*. Start reading at *Examples*.

```
dim(CO2)          # rows, columns

## [1] 84  5

head(CO2, 3)      # first rows

## Grouped Data: uptake ~ conc | Plant
##   Plant   Type  Treatment conc uptake
## 1   Qn1 Quebec nonchilled   95   16.0
## 2   Qn1 Quebec nonchilled  175   30.4
## 3   Qn1 Quebec nonchilled  250   34.8
```

`dim()`, `head()` and `str()` are the first three things to run on any new data set.

# R is a functional language

- Everything you do is a call to a function, $y = f(x)$
- The name `f` is followed by parentheses `()`
- Several arguments are separated by commas, $y = f(x_1, x_2)$
- Arguments can be named, and many have defaults
- Store a result with <- and pass it on to the next function

```
sqrt(2)

## [1] 1.414

round(sqrt(2), digits = 3)

## [1] 1.414

log(100, base = 10)

## [1] 2

args(log)    # arguments and defaults

## function (x, base = exp(1))
## NULL
```

# Outline


1. Getting started
2. Vectors and data frames
3. Graphics
4. Comparing groups
5. Regression and functions

# A data vector

```
x <- c(2, 4, 3, 5, 7, 9)  # scores of 6 students
x

## [1] 2 4 3 5 7 9

length(x); sum(x); mean(x); median(x)

## [1] 6
## [1] 30
## [1] 5
## [1] 4.5

quantile(x); summary(x)

##   0%  25%  50%  75% 100%
## 2.00 3.25 4.50 6.50 9.00
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    3.25    4.50    5.00    6.50    9.00
```

- `c()` combines values into a vector
- <- assigns: read it as “gets”
- # starts a comment
- R is case-sensitive: `x` and `X` are different objects
- `summary()` gives the five-number summary and the mean

# Sorting and ordering

```
sort(x)

## [1] 2 3 4 5 7 9

sort(x, decreasing = TRUE)  # = rev(sort(x))

## [1] 9 7 5 4 3 2

order(x)

## [1] 1 3 2 4 5 6

x[order(x)]                 # same as sort(x)

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

**sort() versus order()** `sort()` returns the values in order. `order()` returns their *positions*: the first entry is the position of the smallest value.

`order()` is what you use to sort a whole data set by one column: `df[order(df$score), ]`.

# Vector arithmetic

```
x + 1; 2 * x; x^2      # scalars are recycled

## [1]  3  5  4  6  8 10
## [1]  4  8  6 10 14 18
## [1]  4 16  9 25 49 81

round(log(x), 3); round(log(x, base = 10), 3)

## [1] 0.693 1.386 1.099 1.609 1.946 2.197
## [1] 0.301 0.602 0.477 0.699 0.845 0.954

gamma(x)               # (x - 1)!

## [1]     1     6     2    24   720 40320

x + c(0, 100)          # recycled 3 times

## [1]   2 104   3 105   7 109
```

Element by element:

$\mathbf{x} + c = (x_1 + c, \ldots, x_n + c)$

$\log \mathbf{x} = (\log x_1, \ldots, \log x_n)$

**Recycling rule** When two vectors have different lengths, the shorter one is repeated. R warns if the longer length is not a multiple of the shorter.

# Extracting elements and naming them

```
x[2]; x[c(2, 4)]       # 2nd; 2nd and 4th

## [1] 4
## [1] 4 5

x[-c(1, 3)]            # drop 1st and 3rd

## [1] 4 5 7 9

x[x > 4]; which(x > 4) # values > 4; positions

## [1] 5 7 9
## [1] 4 5 6
```

```
names(x) <- c("A", "B", "C", "D", "E", "F")
x

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

x[c("A", "F")]; names(x)[x == max(x)]

## A F
## 2 9
## [1] "F"
```

**The extraction operator `[ ]`** A positive index keeps, a negative one drops, a condition keeps the positions where it is `TRUE`. Names travel with the values.

# Lists

```
LD1 <- list(x = c(2, 5, 4), Name = c("X", "Y", "Z"),
            Median = 4)
str(LD1)

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

dump("LD1", file = "data/LD1.txt")  # save as R code
rm(LD1)                              # remove it ...
source("data/LD1.txt")             # ... and restore
LD1$Name

## [1] "X" "Y" "Z"
```

**List** A collection of objects of any type and length, each with a name. Extract a component with `$` or `[[ ]]`.

The result of a statistical test is a list; we will use this to pull out $p$-values and confidence intervals.

# Data frames

```r
x1 <- c(20, 20.5, 13, 16, 11)  # batch A scores
x2 <- c(40, 42, 38, 56, 57)    # batch B scores
score <- c(x1, x2)
group <- gl(2, 5, labels = c("batchA", "batchB"))
DF1 <- data.frame(score, group)
str(DF1, vec.len = 1)          # structure

## 'data.frame': 10 obs. of  2 variables:
##  $ score: num  20 20.5 ...
##  $ group: Factor w/ 2 levels "batchA","batchB": 1 1 ...

DF1[c(1, 6), ]                 # rows 1 and 6

##   score  group
## 1    20 batchA
## 6    40 batchB
```

**Data frame** A list of equal-length columns shown as a table: one row per unit, one column per variable. Columns may be of different types.

`gl(k, n)` generates a factor with $k$ levels, each repeated $n$ times. `DF1[rows, columns]` extracts; an empty index means "all".

# Wide and long layouts

```
wide <- data.frame(batchA = x1, batchB = x2)
wide

##   batchA batchB
## 1   20.0     40
## 2   20.5     42
## 3   13.0     38
## 4   16.0     56
## 5   11.0     57

long <- stack(wide)
names(long) <- c("score", "group")
head(long, 3)

##   score  group
## 1  20.0 batchA
## 2  20.5 batchA
## 3  13.0 batchA
```

- Wide: one column per group, the way data are often typed into Excel
- Long: one column of values and one column saying which group each value belongs to
- Model formulas such as `score` $\sim$ `group` need the long layout; `stack()` converts wide to long

# Saving and reading a data frame

```
write.table(DF1, file = "data/DF1.txt")
DF1 <- read.table("data/DF1.txt", header = TRUE,
                  stringsAsFactors = TRUE)
str(DF1, vec.len = 1)

## 'data.frame': 10 obs. of  2 variables:
##  $ score: num  20 20.5 ...
##  $ group: Factor w/ 2 levels "batchA","batchB": 1 1 ...
```

- `write.table()` / `read.table()`: plain text files
- `write.csv()` / `read.csv()`: CSV files, the usual bridge to Excel

**Text or factor?** Since R 4.0.0, text columns are read as character. Use `stringsAsFactors = TRUE` when they are groups.

# Importing a CSV file

```
D1 <- read.csv("data/D1.csv")
D1

##   obese lean
## 1  9.21 7.53
## 2 11.51 7.48
## 3 12.79 8.08
## 4  8.79 8.11

apply(D1, 2, mean); apply(D1, 2, sd)

## obese  lean
## 10.57  7.80
##  obese   lean
## 1.9000 0.3415
```

- `D1.csv`: energy expenditure of four obese and four lean women (a small illustrative data set)
- No missing values; all values between 7.48 and 12.79
- `apply(X, MARGIN, FUN)` applies `FUN` to each row (`MARGIN = 1`) or each column (`MARGIN = 2`)

# Writing your own functions

```
cv <- function(x) {sd(x) / mean(x) * 100}
se <- function(x) {sd(x) / sqrt(length(x))}
CV <- apply(D1, 2, cv)
SE <- apply(D1, 2, se)
out1 <- rbind(CV, SE)      # bind as rows
round(out1, 2)

##    obese lean
## CV 17.97 4.38
## SE  0.95 0.17

out2 <- cbind(CV, SE)      # bind as columns
round(out2, 2)

##          CV   SE
## obese 17.97 0.95
## lean   4.38 0.17
```

$$\text{CV} = 100\, s/\bar{x}\ \%$$
$$\text{SE}(\bar{x}) = s/\sqrt{n}$$

**Anatomy of a function** `name <- function(arguments) { body }`. The value of the last line in the body is returned.

# Reading the summary table

- Mean energy expenditure is 10.57 for the obese and 7.80 for the lean women
- The coefficient of variation is 18% in the obese group but only 4.4% in the lean group: relative to its mean, the obese group is about 4 times as variable
- The standard error measures how precisely each mean is estimated: 0.95 against 0.17

**Keep in mind** With four women per group these numbers describe the sample only. Whether the groups really differ is a question for a test with a confidence interval, as in Section 4.

# Outline


1. Getting started
2. Vectors and data frames
3. Graphics
4. Comparing groups
5. Regression and functions

# Two kinds of graphics functions

| **High level**: new plot | **Low level**: add to it |
|---|---|
| `plot()`, `hist()` | `points()`, `lines()` |
| `boxplot()`, `barplot()` | `abline()`, `text()` |
| `dotchart()`, `curve()` | `legend()`, `title()` |
| `pairs()` | `curve(..., add = TRUE)` |

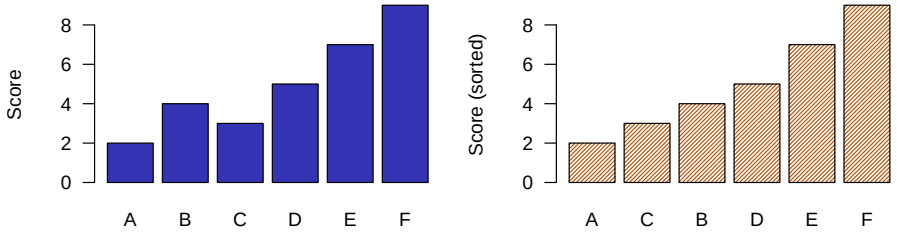
`par()` sets the layout: `mfrow = c(1, 2)` for two panels side by side, `mar` for the margins.

**Build in layers** Draw with one high-level function, then add points, lines and a legend with low-level ones.

```
# colours used in all our plots
col_main <- "#3333B3"
col_alt  <- "#B45309"
```

# Bar charts of the named scores

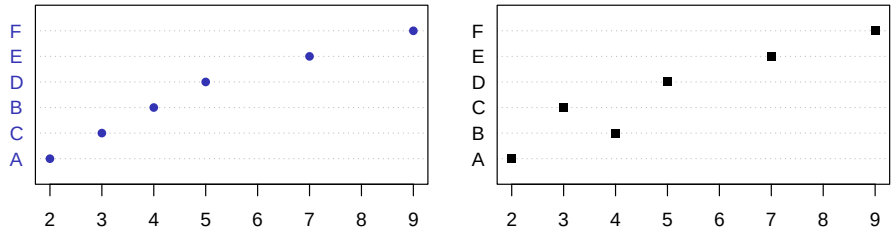
```
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1))
barplot(x, las = 1, col = col_main, ylab = "Score")
barplot(sort(x), las = 1, col = col_alt, density = 40, ylab = "Score (sorted)")
```




The names A–F label the bars automatically. Sorting the bars makes the ranking visible at a glance; `las = 1` keeps axis labels horizontal.

# Dot charts

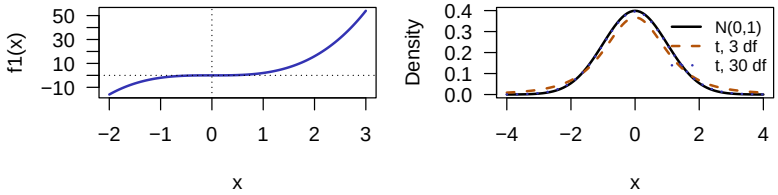
```
par(mfrow = c(1, 2), mar = c(4, 2, 1, 1))
dotchart(sort(x), pch = 16, col = col_main, xlab = "Score: sorted by score")
dotchart(x[order(names(x))], pch = 15, xlab = "Score: sorted by name")
```




A dot chart shows the same information as a bar chart with far less ink, and values are easier to compare along a common scale.

# Plotting functions with `curve()`

```
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), las = 1)
f1 <- function(x) 2 * x^3
curve(f1, from = -2, to = 3, lwd = 2, col = col_main); abline(v = 0, h = 0, lty = 3)
curve(dnorm(x), from = -4, to = 4, lwd = 2, ylab = "Density")
curve(dt(x, df = 3), add = TRUE, lwd = 2, lty = 2, col = col_alt)
curve(dt(x, df = 30), add = TRUE, lwd = 2, lty = 3, col = col_main)
legend("topright", c("N(0,1)", "t, 3 df", "t, 30 df"), lty = 1:3, lwd = 2,
       col = c("black", col_alt, col_main), bty = "n", cex = 0.8)
```




The $t$ density has heavier tails; with 30 degrees of freedom it is almost the $N(0, 1)$ curve.

# Saving a graph

```r
pdf("scores.pdf", width = 6, height = 4)
barplot(x, las = 1, col = col_main)
dev.off()      # close the file

png("scores.png", width = 1200,
    height = 800, res = 200)
barplot(x, las = 1, col = col_main)
dev.off()
```

In RStudio the *Plots* pane also has an *Export* button.

**Devices** `pdf()` (vector, for LaTeX and print) or `png()` (for slides and the web) opens a file; `dev.off()` closes it.

**Good habits** Label axes with units, keep labels horizontal, sort bars by value, and avoid 3-D effects.

# Outline


1. Getting started
2. Vectors and data frames
3. Graphics
4. Comparing groups
5. Regression and functions

# Do two batches differ?

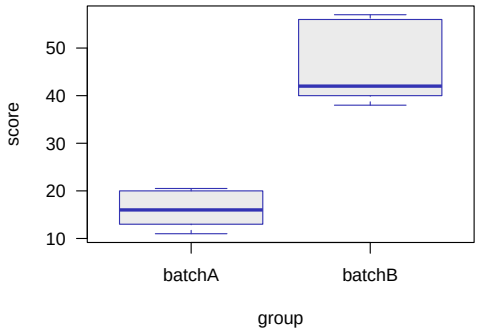
Scores of five students in batch A and five in batch B (illustrative data).

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

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

Look first: batch B scores are higher and also more spread out (a longer box).

```
par(mar = c(4, 4, 1, 1), las = 1)
plot(score ~ group, data = DF1,
     col = "grey92", border = col_main)
```



# Welch two-sample $t$-test

```
t.test(x1, x2)          # variances are not assumed equal

##
##  Welch Two Sample t-test
##
## data:  x1 and x2
## t = -6.8, df = 5.6, p-value = 0.0007
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -41.71 -19.29
## sample estimates:
## mean of x mean of y
##      16.1      46.6
```

Welch's test is R's default; its Welch–Satterthwaite degrees of freedom (5.6) need not be whole numbers.

# Pooled $t$-test and the result object

```
fit.t2 <- t.test(x1, x2, var.equal = TRUE)
names(fit.t2)

##  [1] "statistic"   "parameter"   "p.value"
##  [4] "conf.int"    "estimate"    "null.value"
##  [7] "stderr"      "alternative" "method"
## [10] "data.name"

fit.t2$p.value; fit.t2$conf.int

## [1] 0.0001415
## [1] -40.88 -20.12
## attr(,"conf.level")
## [1] 0.95
```

$$s_p^2 = \frac{(n_A - 1)s_A^2 + (n_B - 1)s_B^2}{n_A + n_B - 2}$$

$$t = \frac{\bar{x}_A - \bar{x}_B}{s_p\sqrt{1/n_A + 1/n_B}}$$

$t \sim t_{n_A+n_B-2}$ under $H_0$

The pooled test assumes equal variances. A test result is a list, and `$` extracts any part for a report.

# Reading the two $t$-tests

- Mean score: batch A 16.1, batch B 46.6; difference $\bar{x}_A - \bar{x}_B = -30.5$ marks
- Welch: $t = -6.77$ on 5.6 df, $p = 0.0007$; 95% CI for $\mu_A - \mu_B$: $(-41.7,\ -19.3)$
- Pooled: the same $t$ (equal group sizes) but 8 df and a slightly narrower interval
- Both reject $H_0$ at the 5% level: batch B scores higher, by roughly 20 to 42 marks

**Interpreting $p$** The $p$-value is the probability of a difference at least this large *if* $H_0$ were true. It is not the probability that $H_0$ is true. The confidence interval shows how large the difference plausibly is.

# Paired $t$-test: when the same students are measured twice

```
t.test(x1, x2, paired = TRUE)

##
##  Paired t-test
##
## data:  x1 and x2
## t = -5.8, df = 4, p-value = 0.004
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
##  -45.09 -15.91
## sample estimates:
## mean difference
##           -30.5
```

$$d_i = x_{Ai} - x_{Bi}$$
$$t = \frac{\bar{d}}{s_d/\sqrt{n}} \sim t_{n-1}$$

**Design decides** Pair only if each A score belongs with one B score (same student, before/after). Never choose the test by its $p$-value.

# The formula interface and equal variances

```
t.test(score ~ group, data = DF1)$p.value

## [1] 0.0006718

t.test(score ~ group, data = DF1,
       var.equal = TRUE)$p.value

## [1] 0.0001415

var.test(score ~ group, data = DF1)$p.value

## [1] 0.1591
```

- `score ~ group` reads "score explained by group" and uses the long data frame
- `var.test()` is the $F$-test of $H_0 : \sigma_A^2 = \sigma_B^2$

**Which $t$-test?** The variance ratio is 0.21, yet with five students per batch the $F$-test cannot detect it ($p = 0.16$). Welch's test is the safer default: it loses little when the variances are equal.

# One-way ANOVA: adding batch C

```
x3 <- c(10, 15, 12, 10, 3)
score <- c(x1, x2, x3)
group <- gl(3, 5, labels = c("batchA", "batchB",
                             "batchC"))
DF2 <- data.frame(score, group)
M2 <- aov(score ~ group, data = DF2)
summary(M2)

##             Df Sum Sq Mean Sq F value    Pr(>F)
## group        2   3845    1923    47.7 0.0000019 ***
## Residuals   12    483      40
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

$$y_{ij} = \mu + \tau_i + \varepsilon_{ij}$$
$$\varepsilon_{ij} \sim N(0, \sigma^2), \text{ independent}$$
$$H_0 : \tau_1 = \tau_2 = \tau_3 = 0$$
$$F = \frac{\text{MS}_{\text{between}}}{\text{MS}_{\text{within}}} \sim F_{k-1,\, N-k}$$

$F = 47.7$ on 2 and 12 df, $p < 0.001$: the three batch means are not all equal. But which ones differ?

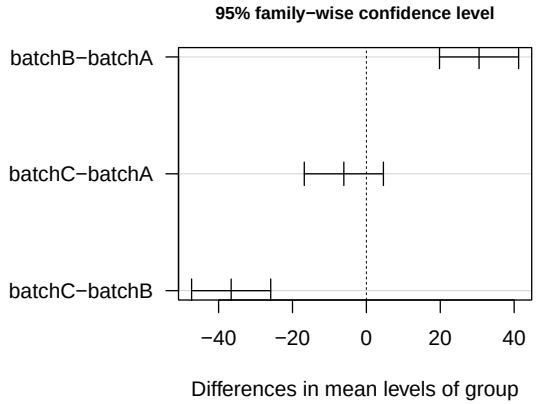
# Group means and Tukey's comparisons

```
model.tables(M2, type = "means",
             se = TRUE)

## Tables of means
## Grand mean
##
## 24.23
##
##  group
## group
## batchA batchB batchC
##   16.1   46.6   10.0
##
## Standard errors for differences of means
##         group
##         4.014
## replic.     5
```

```
TukeyHSD(M2, which = "group")

##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = score ~ group, data = DF2)
##
## $group
##                diff    lwr     upr  p adj
## batchB-batchA  30.5  19.79  41.209 0.0000
## batchC-batchA  -6.1 -16.81   4.609 0.3167
## batchC-batchB -36.6 -47.31 -25.891 0.0000
```

# Tukey intervals

```
par(mar = c(4, 8, 2, 1), las = 1, cex.main = 0.8)
plot(TukeyHSD(M2, which = "group"), las = 1)
```




**Reading it** Intervals that miss 0 indicate a real difference. Batch B is higher than A (by 30.5) and than C (by 36.6). A and C do not differ significantly (adjusted $p = 0.32$).

Tukey's method keeps the *family-wise* error rate at 5% across all three comparisons.

# Checking the ANOVA assumptions

```
bartlett.test(score ~ group, data = DF2)$p.value

## [1] 0.2269

shapiro.test(residuals(M2))$p.value

## [1] 0.6125

kruskal.test(score ~ group, data = DF2)$p.value

## [1] 0.003698
```

- Bartlett: equal variances across groups
- Shapiro–Wilk on the residuals: normal errors
- Kruskal–Wallis: a rank-based alternative that does not assume normality

**With five per group** Neither check rejects ($p = 0.23$ and $0.61$), but such small samples give these tests little power. The rank test agrees with the ANOVA, which is reassuring.

# Outline


1. Getting started
2. Vectors and data frames
3. Graphics
4. Comparing groups
5. Regression and functions

# Correlation between three courses

```
DF3 <- data.frame(batchA = x1, batchB = x2,
                  batchC = x3,
                 row.names = c("A", "B", "C", "D", "E"))
write.table(DF3, file = "data/DF3.txt")
DF3 <- read.table("data/DF3.txt", header = TRUE)
round(cor(DF3), 3)

##        batchA batchB batchC
## batchA  1.000 -0.478  0.696
## batchB -0.478  1.000 -0.699
## batchC  0.696 -0.699  1.000

cor(DF3$batchA, DF3$batchB, method = "spearman")

## [1] -0.3
```

Now suppose the three columns are the marks of the *same* five students (A–E) in three courses.

$$r = \frac{s_{xy}}{s_x \, s_y}$$

$s_{xy}$: sample covariance; $s_x, s_y$: standard deviations. Spearman's $\rho$ is Pearson's $r$ computed on the ranks.

# Testing a correlation

```
cor.test(DF3$batchA, DF3$batchB)

##
##  Pearson's product-moment correlation
##
## data:  DF3$batchA and DF3$batchB
## t = -0.94, df = 3, p-value = 0.4
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.9568  0.6992
## sample estimates:
##     cor
## -0.4778
```

$$H_0 : \rho = 0$$
$$t = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}} \sim t_{n-2}$$

**Reading it**

$r = -0.48$ but $p = 0.42$ and the 95% CI runs from $-0.96$ to $0.7$. With $n = 5$, $|r|$ must exceed 0.878 to be significant at 5%.

# Simple linear regression: the `trees` data

```
data(trees)                    # 31 black cherry trees
head(trees, 3)

##   Girth Height Volume
## 1   8.3     70   10.3
## 2   8.6     65   10.3
## 3   8.8     63   10.2

M3 <- lm(Volume ~ Girth, data = trees)
display(M3, detail = TRUE)   # from package arm

## lm(formula = Volume ~ Girth, data = trees)
##             coef.est coef.se t value Pr(>|t|)
## (Intercept) -36.94     3.37  -10.98    0.00
## Girth         5.07     0.25   20.48    0.00
## ---
## n = 31, k = 2
## residual sd = 4.25, R-Squared = 0.94
```

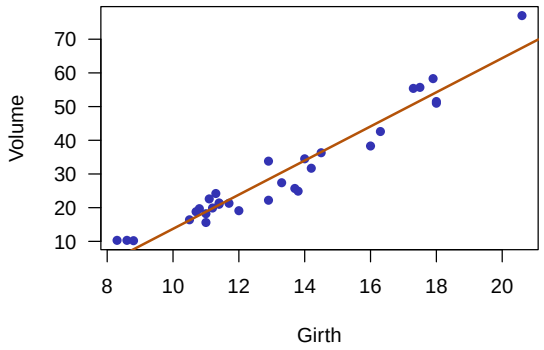
Five students are too few for a regression, so we switch to a built-in data set.

- `Girth`: in fact the tree *diameter* (inches)
- `Height` (feet), `Volume` of timber (cubic feet)

$$\text{Volume}_i = \beta_0 + \beta_1 \, \text{Girth}_i + \varepsilon_i$$

# The fitted line

```
par(mar = c(4, 4, 1, 1), las = 1)
plot(Volume ~ Girth, data = trees, pch = 16,
     col = col_main)
abline(M3, col = col_alt, lwd = 2)
```



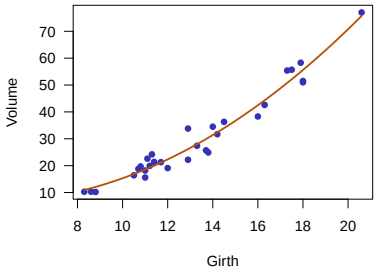

**Reading it** Fitted volume $= -36.94 + 5.07 \times$ Girth. Each extra inch of diameter adds about 5.1 cubic feet of timber; $R^2 = 0.94$ and the residual SD is 4.25 cubic feet.

**But** The intercept (a tree of zero diameter) is meaningless, and the points bend upwards at both ends: a straight line misses some curvature.

# Adding a quadratic term

```
M3q <- lm(Volume ~ Girth + I(Girth^2), data = trees)
display(M3q)

## lm(formula = Volume ~ Girth + I(Girth^2), data = trees)
##             coef.est coef.se
## (Intercept) 10.79    11.22
## Girth       -2.09     1.65
## I(Girth^2)   0.25     0.06
## ---
## n = 31, k = 3
## residual sd = 3.33, R-Squared = 0.96
```

```
par(mar = c(4, 4, 1, 1), las = 1); b <- coef(M3q)
plot(Volume ~ Girth, data = trees, pch = 16, col = col_main)
curve(b[1] + b[2] * x + b[3] * x^2, add = TRUE,
      col = col_alt, lwd = 2)
```




`I()` "insulates" ^ so that it means a power, not a formula operator.

# Multiple regression: diameter and height

```
M4 <- lm(Volume ~ Girth + Height, data = trees)
display(M4, detail = TRUE)

## lm(formula = Volume ~ Girth + Height, data = trees)
##             coef.est coef.se t value Pr(>|t|)
## (Intercept) -57.99     8.64   -6.71    0.00
## Girth         4.71     0.26   17.82    0.00
## Height        0.34     0.13    2.61    0.01
## ---
## n = 31, k = 3
## residual sd = 3.88, R-Squared = 0.95
```

**Reading it** Holding height fixed, each extra inch of diameter adds about 4.71 cubic feet; holding diameter fixed, each extra foot of height adds about 0.34 cubic feet. $R^2$ rises from 0.935 to 0.948.

# Comparing nested models with anova()

```
print(anova(M3, M3q), signif.stars = FALSE)

## Analysis of Variance Table
##
## Model 1: Volume ~ Girth
## Model 2: Volume ~ Girth + I(Girth^2)
##   Res.Df RSS Df Sum of Sq    F  Pr(>F)
## 1     29 524
## 2     28 311  1       213 19.1 0.00015
```

```
print(anova(M3, M4), signif.stars = FALSE)

## Analysis of Variance Table
##
## Model 1: Volume ~ Girth
## Model 2: Volume ~ Girth + Height
##   Res.Df RSS Df Sum of Sq    F Pr(>F)
## 1     29 524
## 2     28 422  1       102 6.79  0.014
```

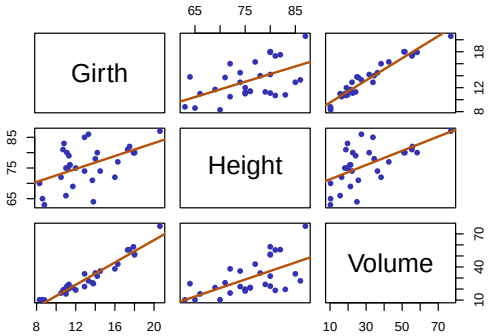
**Reading it** Each test asks whether the extra term reduces the residual sum of squares by more than chance would. Both do: $p = 0.00015$ for Girth$^2$ and $p = 0.014$ for Height. Geometry suggests why: volume grows roughly like diameter$^2 \times$ height.

# A function of your own for `pairs()`

```r
pan1 <- function(x, y) {
  # points and a fitted line in one panel
  m1 <- lm(y ~ x)
  points(x, y, pch = 16, col = col_main)
  abline(m1, col = col_alt, lwd = 2)
}
dump("pan1", file = "data/pan1.txt")
source("data/pan1.txt")
```

`dump()` saves the function as R code, so it can be reused in another session with `source()`.

```r
pairs(trees, panel = pan1)
```



# Key take-aways

1. **R is a functional language:** everything is an object and every action is a function call.
2. **Vectors are the building blocks**; data frames hold data sets and lists hold results.
3. **Work in a project folder** and read data with relative paths; keep data in plain text (CSV).
4. **Look before you test:** box plots, dot charts and scatter plots come first.
5. **The design decides the test:** independent groups (Welch $t$, ANOVA) or pairs (paired $t$).
6. **Report estimates with confidence intervals**; five observations show little.
7. **Write small functions** (`cv`, `se`, `pan1`) to make R work your way.

**Next session:** simple linear regression in depth, and the curvature in `trees`.

# References


- 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.
- Jones, O., Harden, S., and Crawley, M. J. (2022). *The R Book*, 3rd Ed., Wiley, Hoboken.
- Gelman, A., and Hill, J. (2007). *Data Analysis Using Regression and Multilevel/Hierarchical Models*, Cambridge University Press, Cambridge.
- 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 (Posit) `https://posit.co/download/rstudio-desktop/`

# Thank you

Questions and discussion