R Cheat Sheet

Computational Cognitive Science (INFR10054) | for students who already know Python or Java

1. Python / Java -> R

Python / Java	R
`x = 5`	`x <- 5  # 'x = 5' also works`
`[1, 2, 3]`	`c(1, 2, 3)  # a vector`
`double[] a = new double[n]`	`a <- numeric(n)`
`range(n)  (0 .. n-1)`	`seq_len(n)  (1 .. n)`
`x[0]  x[-1]  x[0:3]`	`x[1]  x[length(x)]  x[1:3]`
(slice end is excluded)	(both ends included; `x[-1]` DROPS the 1st)
`True  False  None`	`TRUE  FALSE  NULL` (`NA` = missing)
`a and b / a or b / not a`	`a && b / a || b / !a` (scalars)
(numpy: `a & b / a | b`)	`a & b / a | b` (elementwise, vectors)
`x**2  x//2  x%2`	`x^2  x %/% 2  x %% 2`
`print(f"n = {n}")`	`cat("n =", n, "\n")`
`f"{a}_{b}"`	`paste0(a, "_", b)`
`def f(x, p=2): return x**p`	`f <- function(x, p = 2) x^p`
`for i in range(n): ...`	`for (i in seq_len(n)) { ... }`
`if / elif / else`	`if / else if / else` (braces)
`d = {'a': 1};  d['a']`	`d <- list(a = 1);  d$a  d[["a"]]`
`[f(v) for v in xs]`	`sapply(xs, f)` / `lapply(xs, f)`
`import numpy as np`	`library(pkg)` (install once: `install.packages`)
`np.random.normal(0, 1, 10)`	`rnorm(10, mean = 0, sd = 1)`
`len(x)  x.append(v)`	`length(x)  x <- c(x, v)` (slow)
indentation = structure	`{ }` = structure; indent is only style

2. Vectors: the core data type

No scalars: `5` is a length-1 vector. Arithmetic and comparisons are **vectorised** (elementwise), so avoid loops where you can. Indices start at **1**.

```
x <- c(2, 4, 6, 8)
1:5;  seq(0, 1, by = 0.25);  seq(0, 1, length.out = 5)
rep(0, 3);  rep(1:2, times = 2);  rep(1:2, each = 2)
x[2];  x[-1];  x[c(1, 3)];  x[x > 4]   # pick/drop/filter
x * 2;  x + x;  x > 3                  # elementwise
sum(x); mean(x); sd(x); var(x); median(x)
length(x); rev(x); sort(x); order(x); cumsum(x); diff(x)
which(x > 4)        # positions where TRUE
which.max(x)        # position of the maximum
any(x > 5);  all(x > 0);  4 %in% x
```

3. Lists and data frames

```
p <- list(name = "Ada", n = 3)      # like a dict; mixed types
p$name;  p[["n"]];  p["n"]      # [[ ]] extracts; [ ] keeps list
df <- data.frame(id = 1:3, rt = c(.4, .5, .6))
df$rt;  df[2, ];  df[df$rt > .45, ]   # column; row; filter
df$acc <- c(1, 0, 1)                # add a column
nrow(df); ncol(df); head(df); str(df); summary(df)
d <- read.csv("data.csv")           # write.csv(d, "out.csv")
```

4. Matrices

```
m <- matrix(0, nrow = 3, ncol = 4)    # filled column by column
m[2, 3] <- 1;  m[1, ];  m[, 2];  m[1, , drop = FALSE]
dim(m);  t(m);  m %*% t(m)            # %*% = matrix product
rowSums(m);  colMeans(m);  apply(m, 1, max)  # 1=rows 2=cols
cbind(a, b);  rbind(a, b)             # stack as columns / rows
```

5. Functions, control flow, loops

```
f <- function(x, p = 2) {
  x^p                             # last value is returned
}
f(3);  f(3, p = 3)                # positional or named args
if (x > 0) "pos" else "neg"       # if is an expression
ifelse(x > 0, "pos", "neg")       # vectorised version
for (i in seq_len(n)) { ... }     # also: while (cond) { ... }
break;  next                      # next = Python's continue
sapply(1:5, function(i) i^2)      # -> vector (lapply: list)
sapply(1:5, \(i) i^2)             # \(i) shorthand, R >= 4.1
replicate(1000, mean(rnorm(10)))     # repeat an expression
1:5 |> rev() |> head(2)           # pipe: x |> f(y) is f(x, y)
```

6. Probability distributions and sampling

Each distribution has four functions: `d` density / mass, `p` cumulative P(X<=q), `q` quantile (inverse CDF), `r` random draws.

Distribution	Functions (+ parameters)
Normal	`dnorm pnorm qnorm rnorm` (mean, sd)
Binomial / Bernoulli	`dbinom ...` (size, prob); size = 1
Uniform	`dunif ...` (min, max)
Poisson / Exponential	`dpois` (lambda); `dexp` (rate)
Beta / Gamma	`dbeta` (shape1, shape2); `dgamma` (shape, rate)
Student t / Chi-sq.	`dt` (df); `dchisq` (df)

```
dnorm(0, mean = 0, sd = 1)          # density at 0
dbinom(7, size = 10, prob = 0.5)    # P(X = 7)
pnorm(1.96);  qnorm(0.975)          # CDF; inverse CDF
pnorm(2, lower.tail = FALSE)        # P(X > 2)
dnorm(x, 0, 1, log = TRUE)          # log density (use this)
set.seed(42)                        # reproducible randomness
rnorm(5, mean = 100, sd = 15)
sample(1:6, 10, replace = TRUE)     # ten die rolls
sample(c("L", "R"), 20, replace = TRUE,
       prob = c(.3, .7))            # weighted choice
```

7. Likelihood, fitting, model comparison

Fit by minimising the **negative log-likelihood** with `optim()`. Sum log-probabilities: multiplying many probabilities underflows to 0.

```
nll <- function(par, x) {          # par = c(mu, sigma)
  -sum(dnorm(x, mean = par[1], sd = par[2], log = TRUE))
}
fit <- optim(par = c(0, 1), fn = nll, x = x,
             method = "L-BFGS-B", lower = c(-Inf, 1e-6))
fit$par;  fit$value;  fit$convergence   # 0 = converged
k <- length(fit$par);  n <- length(x)
AIC <- 2 * k + 2 * fit$value            # lower = better
BIC <- k * log(n) + 2 * fit$value
```

- `optim` minimises (default Nelder-Mead, no bounds). Try several starting values to avoid local optima.
- Extra arguments (`x = x`) are passed on to `fn`; add `hessian = TRUE` for curvature.

Bayes on a grid

```
theta <- seq(0, 1, length.out = 101)   # candidate values
prior <- dbeta(theta, 1, 1)
lik   <- dbinom(7, size = 10, prob = theta)   # 7 of 10
post  <- prior * lik / sum(prior * lik)       # normalise
theta[which.max(post)]                        # posterior mode
```

Information theory (surprisal, entropy)

```
surprisal <- -log2(p)                 # bits; p = P(event)
H  <- -sum(p[p > 0] * log2(p[p > 0]))  # entropy of p
kl <- sum(p * log2(p / q))            # KL(p||q); p, q > 0
```

Least squares and model weights

```
rmsd <- function(obs, pred) sqrt(mean((obs - pred)^2))
aic <- c(m1 = 210.3, m2 = 205.9)      # AIC of each model
delta <- aic - min(aic)
w <- exp(-delta / 2) / sum(exp(-delta / 2))   # Akaike weights
```

8. Worked example: fit a forgetting curve

Made-up data: proportion recalled after a delay. Model: exponential decay `a * exp(-b * t)`, fitted by minimising RMSD.

```
tt  <- c(1, 3, 6, 9, 12, 18)             # delay
obs <- c(.82, .70, .55, .47, .40, .31)   # proportion recalled
pred <- function(par, tt) par[1] * exp(-par[2] * tt)
loss <- function(par) sqrt(mean((obs - pred(par, tt))^2))
fit <- optim(c(1, 0.1), loss)            # start at a=1, b=0.1
fit$par;  fit$value                      # best (a, b); RMSD
plot(tt, obs, pch = 19)                  # data as points
curve(pred(fit$par, x), add = TRUE)      # fitted curve on top
```

9. Simulation pattern (random walk / drift-diffusion)

```
nreps <- 1000;  nsamples <- 500
latencies <- numeric(nreps)                  # preallocate
evidence  <- matrix(0, nreps, nsamples + 1)  # col 1 = time 0
for (i in seq_len(nreps)) {
  steps <- rnorm(nsamples, mean = drift, sd = sdrw)
  evidence[i, ] <- cumsum(c(0, steps))
  latencies[i] <- which(abs(evidence[i, ]) > criterion)[1]
}                                            # NA: no crossing
mean(latencies, na.rm = TRUE)
```

10. Data from many participants

```
d <- read.csv("data.csv")   # cols: subject, cond, rt, acc
aggregate(rt ~ subject + cond, data = d, FUN = mean)
tapply(d$rt, d$cond, mean)              # mean rt per cond.
by_s <- split(d, d$subject)             # one data frame each
sapply(by_s, function(s) mean(s$rt))    # one per subject
fits <- lapply(by_s, function(s) {
  optim(c(0, 1), nll, x = s$rt)         # fit each participant
})
do.call(rbind, lapply(fits, function(f) f$par))  # table
```

11. Handy modelling snippets

```
normalise <- function(w) w / sum(w)      # weights -> probs
softmax <- function(v, beta = 1) {       # choice rule
  e <- exp(beta * (v - max(v)))          # max-shift: stable
  e / sum(e)
}
sim <- exp(-sens * abs(x1 - x2))         # similarity
sample(seq_along(p), 1, prob = p)        # draw index from p
outer(a, b, function(u, v) abs(u - v))   # pairwise distances
```

12. Quick statistics

```
t.test(rt ~ cond, data = d)              # two-sample t-test
cor(x, y);  cor.test(x, y)               # correlation
m <- lm(rt ~ cond + age, data = d)        # linear regression
summary(m);  coef(m);  predict(m, newdata = nd)
glm(acc ~ cond, family = binomial, data = d)   # logistic
chisq.test(table(d$cond, d$acc))         # independence
```

13. Plotting

```
hist(x, breaks = 20, freq = FALSE, main = "", xlab = "RT")
curve(dnorm(x, 0, 1), add = TRUE, col = "red")   # overlay
plot(x, y, type = "l", col = "blue", lwd = 2,
     xlab = "Time", ylab = "Evidence", main = "Title")
lines(x, y2);  points(x, y);  abline(h = 0, lty = 2)
legend("topright", legend = c("A", "B"), col = 1:2, lty = 1)
par(mfrow = c(1, 2))          # 1 x 2 panels (reset: c(1, 1))
matplot(t(evidence[1:5, ]), type = "l", lty = 1)  # 5 walks
barplot(table(responses))
png("fig.png", 800, 600);  plot(x, y);  dev.off()   # save
```

ggplot2 (works on data frames)

```
library(ggplot2)
ggplot(df, aes(x = t, y = ev, colour = cond)) +
  geom_line() + labs(x = "Time", y = "Evidence") +
  theme_minimal()
```

14. Strings and other odds and ends

```
paste("a", "b");  paste0("a", 1:3);  sprintf("%.2f", pi)
nchar(s);  substr(s, 1, 3);  toupper(s)
strsplit(s, " ")[[1]]     # split into words
gsub("a", "b", s);  grepl("ab", s);  as.numeric("3.5")
table(x);  unique(x);  round(x, 2);  cut(x, breaks = 3)
```

15. Common gotchas

- `1:0` is `c(1, 0)`, not empty. Loop over `seq_len(n)`.
- `<-` assigns, `==` compares. Use `TRUE` / `FALSE`, not `T` / `F`.
- `if` needs **one** TRUE/FALSE: use `&& ||` there, and `& |` on vectors.
- `NA` spreads: `mean(c(1, NA))` is NA. Use `na.rm = TRUE` and `is.na(x)` (never `x == NA`). `which(cond)[1]` gives NA if nothing matches.
- `[` keeps the type, `[[` extracts one element. `m[1, ]` drops to a vector; add `drop = FALSE` to keep a matrix.
- Recycling: `1:6 + 1:2` silently repeats the short vector; if lengths do not divide you only get a warning.
- Matrices fill by **column** (`byrow = TRUE` for rows). `*` is elementwise, `%*%` is the matrix product.
- Copy-on-modify: `y <- x; y[1] <- 0` leaves `x` alone. A function cannot change its arguments in place; return the result.
- Growing a vector in a loop is slow; preallocate with `numeric(n)` or `matrix(0, r, c)`.
- `5 / 2` is 2.5 (always double); `5 %/% 2` is 2. Compare floats with `isTRUE(all.equal(a, b))`, not `==`.
- At top level, `else` must sit on the same line as the closing `}`.

16. Debugging, help, workflow

- **Read the first line of the error.** `object not found` = typo, or the line was not run yet; `non-numeric argument` = wrong type; `subscript out of bounds` = index too large.
- Inspect: `str(x) class(x) head(x) length(x) summary(x)`. Add `print()` or `cat()` calls freely.
- `stopifnot(n > 0)` checks assumptions; `traceback()` after an error; `browser()` inside a function steps through it.
- `tryCatch(expr, error = function(e) NA)` handles errors.
- Help: `?mean ??"regression" example(plot)`.
- Scripts: `source("file.R")`, `getwd()` / `setwd()`. `pkg::fn()` calls a function without `library`.
- **Reproducibility:** `set.seed(1)` at the top. Before submitting, restart R and re-run the whole script from a clean session.

RStudio shortcuts

Action	Windows/Linux (Mac)
Run line / selection	Ctrl+Enter (Cmd+Enter)
Run whole script	Ctrl+Alt+R (Cmd+Opt+R)
Insert `<-`	Alt+- (Opt+-)
Comment / uncomment	Ctrl+Shift+C (Cmd+Shift+C)
Restart R session	Ctrl+Shift+F10 (Cmd+Shift+0)
Help for function	F1 with cursor on its name

17. Packages and further reading

```
install.packages("ggplot2")   # once per machine
library(ggplot2)              # once per session
.libPaths()                   # folders packages install into
```

- **R for Data Science** (free online): r4ds.hadley.nz
- **Advanced R** (language details): adv-r.hadley.nz
- Posit cheat sheets: posit.co/resources/cheatsheets
- Course textbook: Farrell and Lewandowsky (2018), *Computational Modeling of Cognition and Behavior*.
- Stuck? Ask on EdStem, and include a minimal example that reproduces the problem.

Generated by Claude, use with caution.