Spatial data with terra :: CHEAT SHEET

raster + vector analysis with SpatRaster & SpatVector

SPATRASTER · SPATVECTOR · RASTER ↔ VECTOR · GENERAL

The classes

SpatRaster — grid of equal-size cells with one or more layers. Can be file-backed: huge files are read in chunks, not loaded into RAM.
SpatVector — points, lines or polygons + attribute table (like a data.frame with geometries).
SpatExtent — rectangular bounding box. **ext(xmin, xmax, ymin, ymax)**
sds(r1, r2) — SpatRasterDataset: sub-datasets (variables × time, e.g. NetCDF).
sprc(list) — SpatRasterCollection: rasters whose geometry does not align.
svc(v1, v2) — SpatVectorCollection.

Spat* objects hold C++ pointers: **wrap(x)** / **unwrap(x)** before .rds or parallel workers (saveRDS wraps for you).

Create & read

r <- rast("dem.tif") — read any GDAL source: GeoTIFF, NetCDF, COG, "/vsicurl/https://..."
rast("x.nc", subds="tmp", lyrs=1:12) — pick sub-dataset / layers
rast(nrows=18, ncols=36, xmin=-180, xmax=180, ymin=-90, ymax=90, crs="EPSG:4326") — from scratch
rast(matrix), **rast(array)**, **rast(df, type="xyz")** — from R data
rast(r) — template: same geometry, no values
rast(v, res=100) — template from a SpatVector's extent
vrt(files) — virtual mosaic of many tiles
describe("f.tif") — file metadata (gdalinfo)
writeRaster(r, "out.tif", overwrite=TRUE) — write; set **datatype="INT2S"**, **gdal=c("COMPRESS=LZW")**
writeCDF(r, "out.nc") — write NetCDF

Inspect

nrow(r); ncol(r); nlyr(r); ncell(r) — dimensions & n layers
res(r); ext(r); origin(r) — resolution, extent, origin
crs(r, describe=TRUE) — coordinate reference system
names(r) <- c("a","b") — layer names; also **varnames()**, **units()**, **time()**
minmax(r) — min/max per layer (**setMinMax(r)** to compute)
summary(r), **unique(r)**, **freq(r)** — value counts
inMemory(r); sources(r) — in RAM or on disk?
datatype(r) — on-disk data type (e.g. INT2S, FLT4S)
head(r); **tail(r)** — peek at values
compareGeom(r1, r2) — same extent / res / crs?

Cells, coordinates & values

values(r) — all values as matrix; **values(r) <- x** to set
r[42]; r[3, 5]; r[1:2,] — index by cell or row, col
extract(r, cbind(x, y)) — values at coordinates
cells(r, v) — cell numbers covered by a SpatVector
xyFromCell(r, cell) ↔ **cellFromXY(r, xy)**
rowColFromCell(r, cell) ↔ **cellFromRowCol(r, row, col)**
crds(r) — coordinates of (non-NA) cells
spatSample(r, 100, "random") — also "regular", "stratified"; **as.points=TRUE**, **na.rm=TRUE** (works on SpatVector/SpatExtent too)

Subset & combine layers

r[[2]]; r[[c("a","b")]]; r$a — select layers
subset(r, 1:3) — by index or name; **negate=TRUE** to drop
c(r1, r2) — stack layers (must align); **add(r) <- s** in place
rep(r, 3) — replicate layers

Map algebra

r * 10 + s; sqrt(r); log(r) — cell-wise arithmetic & math
r > 3; r == 1; r %in% c(1,3) — comparisons → boolean raster
ifel(r > 3, 1, NA) — if-else
mean(r); sum(r1, r2); min(r); which.max(r) — across layers
app(r, fun=sum, na.rm=TRUE) — apply function over layers, per cell
lapp(c(r, s), \(x, y) (x - y) / (x + y)) — layers as function arguments
tapp(r, index="months", fun=mean) — by group of layers (e.g. time)
xapp(r, s, fun=cor) — function over two SpatRasters
roll(r, n=5, "mean") — rolling; **diff(r)**, **cumsum(r)** — over layers
weighted.mean(r, w=1:12); **rowSums(r)**; **colMeans(r)**

Change cell values

classify(r, rcl=cbind(from, to, becomes)) — reclassify ranges
subst(r, from=1, to=100) — substitute specific values
clamp(r, 0, 1) — truncate to range; **thresh(r)** — auto-threshold
scale(r); scale_linear(r, 0, 1); stretch(r) — rescale
mask(r, m) — NA where m is NA; **inverse=TRUE**, **updatevalue=0**
cover(r, s) — fill NA cells of r with values of s
init(r, fun=runif) — fill with values

Change geometry

crop(r, e) — cut to extent of any object; **mask=TRUE** clips to polygon
extend(r, e) — add (NA) rows/cols; **trim(r)** — drop outer NA margins
merge(r1, r2) — combine rasters, first has priority
mosaic(r1, r2, fun="mean") — combine, summarizing overlaps
aggregate(r, fact=10, fun="mean") — to coarser cells
disagg(r, fact=10, method="bilinear") — to finer cells
resample(r, template, method="bilinear") — transfer to another grid
project(r, "EPSG:3035") — reproject; use **method="near"** for categorical data; **project(r, y)** matches y's grid
shift(r, dx=1); flip(r, "vertical"); rotate(r); t(r)
align(ext(v), r) — snap an extent to r's grid
is.rotated(r); **rectify(r)** — fix a rotated raster
divide(r, n=4) — subdivide into parts
impose(sprc(r1, r2), r) — force a collection onto one grid

Focal (neighborhood)

focal(r, w=3, fun="mean", na.policy="omit") — moving window
focal(r, w=focalMat(r, 1000, "circle"), fun=sum) — weights matrix ("circle", "Gauss", "rectangle")
focal3D(r, c(3,3,3), fun=mean) — window across rows, cols & layers
focalPairs(r, w=3, "pearson"), **focalReg(r, 3)** — two-layer windows
focalValues(r, w=3) — neighborhood values as a matrix

Terrain & hydrology

terrain(r, v=c("slope","aspect"), unit="radians") — also "TPI", "TRI", "roughness", "flowdir"
shade(slope, aspect, angle=45, direction=315) — hillshade
viewshed(r, loc=c(x, y)) — visible cells; **surfArea(r)** — 3-D area
flowAccumulation(flowDir(r)); **watershed(fd, pp)** — catchments
pitfinder(fd); **pitfiller(r)** — find / fill depressions

Distance, patches & cost

distance(r) — from NA cells to nearest non-NA; **distance(r, v)** — to a SpatVector
direction(r) — direction to nearest non-NA cell
costDist(r, target=0) — cost-weighted distance
gridDist(r) — distance moving over the grid, NA = barrier
patches(r, directions=8) — label connected cell clumps
sieve(r, threshold=10) — remove small patches
boundaries(r) — detect edge cells
buffer(r, width=500) — TRUE within distance of non-NA cells

Summaries

global(r, "mean", na.rm=TRUE) — one statistic per layer
zonal(r, z, fun="mean") — statistics per zone (raster or polygons)
freq(r); **crosstab(c(r, s))** — (cross-)tabulate counts
expanse(r, unit="km") — area of non-NA cells; **cellSize(r)** — cell-area raster (geodesic-aware)
layerCor(r, "cor") — between-layer correlation
quantile(r); **autocor(r)** — Moran's I

Categorical rasters

levels(r) <- data.frame(id=1:3, cover=c("forest","water","urban"))
cats(r); **activeCat(r) <- 2** — get / choose category table
catalyze(r) — factor → numeric layers; **as.factor(r)**
segregate(r) — one 0/1 layer per class (one-hot)
concats(r, s) — combine two categorical layers

Time series

time(r) <- as.Date("2026-01-01") + 0:364 — assign layer times
has.time(r); **timeInfo(r)**; **depth(r)** — third dimension
tapp(r, "yearmonths", mean) — aggregate over time
fillTime(r) — insert missing time steps as NA layers
approximate(r) — interpolate NA cells across layers
mergeTime(sds(r1, r2)) — merge overlapping timelines
clamp_ts(r) — clamp a time series

Colors & RGB

RGB(r) <- c(3, 2, 1) — declare which layers are the R, G, B channels; **has.RGB(r)**
colorize(r, "col") — RGB layers → single layer + color table; also "hsv", "hsl", "rgb"
coltab(r) <- data.frame(value=1:3, col=c("red","gold","navy")) — color table for categorical rasters; **has.colors(r)**

terra :: vectors, raster ↔ vector, plotting

SPATRASTER · SPATVECTOR · RASTER ↔ VECTOR · GENERAL

Create & read

`v <- vect("parcels.gpkg")` — any GDAL/OGR source (.shp, .gpkg, GeoJSON, PostGIS, ...)
`vect("f.gpkg", query="SELECT … WHERE …", extent=e)` — partial read; `proxy=TRUE` reads no geometries yet
`vect(cbind(x, y), atts=df, crs="EPSG:4326")` — points from coords
`vect("POLYGON ((0 0, 5 0, 5 5, 0 0))")` — from WKT
`vect(sf_object)` — from sf; `sf::st_as_sf(v)` goes back
`writeVector(v, "out.gpkg", overwrite=TRUE)`
`vector_layers("f.gpkg")` — list layers in a file

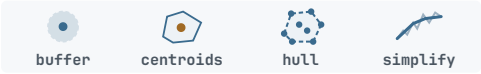
Attributes

`values(v)` / `as.data.frame(v)` — attribute table; `v$pop`, `v[["pop"]]`
`v$dens <- v$pop / expanse(v, "km")` — add / compute a column
`merge(v, df, by="id")` — join a data.frame
`subset(v, v$pop > 1e5, select=c("name","pop"))` — or `v[v$pop > 1e5, 1:2]`
`sort(v, "pop")`; `unique(v)`; `na.omit(v)`; `head(v)`
`as.data.frame(v, geom="WKT")` — attributes + geometry as text

Geometry properties

`geom(v)` — vertices matrix; `crds(v)` — coordinates
`geomtype(v)`; `is.points(v)`, `is.lines(v)`, `is.polygons(v)`
`nrow(v)` — n geometries; `nseg(v)` — n segments
`expanse(v, unit="km")` — polygon area (geodesic for lon/lat)
`perim(v)` — perimeter / line length
`is.valid(v)`; `makeValid(v)` — check / repair geometries

Derive geometries

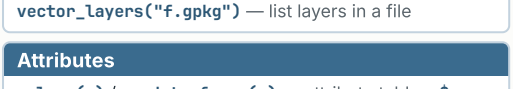


`buffer(v, width=100)` — geodesic-aware buffer (m for lon/lat)
`centroids(v, inside=TRUE)` — point guaranteed inside
`hull(v, "convex")` — also "rectangle", "circle", "concave_ratio"
`voronoi(v)`; `delaunay(v)`
`simplifyGeom(v, tolerance=.01)` — fewer nodes; `densify(v, 100)` — more
`fillHoles(v)`; `gaps(v)` — holes & slivers between polygons
`disagg(v)` — multipart → single parts; `aggregate(v, "region")` — dissolve by attribute
`snap(v, tol=.1)` — topology cleanup; `removeDupNodes(v)`

Transform & topology

`shift(v, dx=100, dy=0)` — move; `spin(v, angle=30)` — rotate; `rescale(v, .5)` — shrink / grow around a center
`flip(v, "vertical")`; `t(v)` — mirror / swap x-y
`normalize.longitude(v)`; `rotate(v)` — fix dateline-crossing data / 0–360° longitudes
`mergeLines(v)`; `makeNodes(v)` — connect lines, nodes at intersections
`elongate(v, length=10)`; `densify(v, 100)` — extend lines, add vertices

Overlay



`intersect(v1, v2)` — intersection, attributes of both
`union(v1, v2)` — all pieces; `symdif(v1, v2)` — all but overlap
`erase(v1, v2)` — v1 minus v2; `erase(v)` — remove self-overlaps
`crop(v, e)` — clip to extent / polygons; `mask(v, m)`
`cover(v1, v2)` — replace overlapping parts
`v[p, ]` — subset geometries that intersect p

Spatial relations

`relate(v1, v2, "intersects")` — matrix; also "touches", "within", "contains", "overlaps", "crosses", or a DE-9IM pattern
`is.related(v1, v2, "intersects")` — one logical per geometry
`adjacent(v)` — neighboring polygons
`nearest(v1, v2)`; `nearby(v, distance=1000)`
`distance(v1, v2)` — geodesic distance matrix (m)
`sharedPaths(v)` — shared borders

Points, density & routes

`thin(pts, 1000)` — subset points by a minimum distance
`agitate(pts)` — jitter overlapping points
`snapTo(pts, lns)` — snap points onto the nearest line / edge
`dots(polys, "pop", size=1000)` — dot-density map: one dot per 1000
`cartogram(polys, "pop", "nc")` — area ∝ variable; also "circles"
`spatSample(polys, 100, "random")` — random points in polygons
`furdist(v1, v2)` — distance to the furthest location
`n <- netw(lns)`; `shortestPath(n, from, to)` — network routing

CRS & projection

`crs(x)` — WKT; `crs(x, describe=TRUE)$code` — EPSG code
`crs(x) <- "EPSG:4326"` — declare / overwrite. Does NOT transform!
`project(x, "EPSG:3035")` — transform raster or vector; accepts EPSG, WKT, proj4, or a template object
`same.crs(x, y)`; `is.lonlat(x)`

Many terra functions (distance, expanse, buffer, cellSize...) compute geodesically for lon/lat data — often no need to project first.

Raster ↔ vector

`extract(r, pts)` — values at points, one column per layer; `bind=TRUE` returns SpatVector
`extract(r, polys, fun=mean, na.rm=TRUE)` — summarize per polygon; `exact=TRUE` weights by cover fraction
`zonal(r, polys, fun="mean")` — fast polygon statistics
`extractAlong(r, line)` — cell values along a line
`rasterize(v, r, field="value", fun="max", background=NA)`
`rasterizeGeom(v, r, "count")` — or "length", "area" per cell
`crop(r, v, mask=TRUE)` — clip raster to polygon outline
`as.points(r)`; `as.lines(r)`; `as.contour(r)`
`as.polygons(r)` — dissolves by value; `aggregate=FALSE` keeps cells

Interpolation & prediction

`interpIDW(r, pts, "z", radius=1e4)` — inverse-distance weighted
`interpNear(r, pts, "z")` — nearest-neighbor interpolation
`interpolate(r, model)` — any spatial model with predict (kriging via gstat, thin-plate splines via fields::Tps, ...)
`predict(r, model)` — glm / randomForest / ... ; layer names must match the model's predictor names
`k_means(r, centers=5)`; `prcomp(r)` — clustering, PCA

Plotting

`plot(r)` — map; `col=map.pal("viridis")`, `breaks=`, `range=`, `legend=`
`plot(r, type="interval")` — or "classes", "continuous"
`plot(v, "pop", type="interval")` — choropleth
`plotRGB(r, 3, 2, 1, stretch="lin")` — satellite composite
`lines(v)`; `points(v)`; `polys(v)`; `text(v, "name")`; `halo(x, y, "lab")` — add to map
`sbar(50, "bottomright")`; `north()` — scale bar, north arrow
`panel(r)` — multi-layer panel, shared legend; `inset()`
`plet(r)` — interactive leaflet map
`hist(r)`; `pairs(r)`; `persp(r)`; `contour(r, add=TRUE)`

Big data & performance

`terraOptions(memfrac=.7, tempdir="…", progress=10)`
`fun(r, …, filename="out.tif", wopt=list(…))` — most functions write directly to disk: safe for huge rasters
`window(r) <- e` — restrict all reads to an extent
`makeTiles(r, y, "tile_.tif")` — split into tiles; `tile_apply(r, fun)` — process tiles in parallel
`app(r, fun, cores=4)` — parallel apply (also predict, ...)
`mem_info(r)`; `free_RAM()`; `tmpFiles(remove=TRUE)`

Map furniture & extras

`g <- graticule(lon=30, lat=30, crs="+proj=robin")` — graticule to plot / add; `add_grid()` — reference grid
`add_legend("topleft", …)`; `legend_cont()` — continuous legend; `add_mtext()`, `add_box()`
`animate(r, pause=.25)` — flip through layers
`barplot(r)`; `boxplot(r, z)` — values of r grouped by z

Extents (SpatExtent)

`e <- ext(5.7, 6.6, 49.4, 50.2)` — xmin, xmax, ymin, ymax
`xmin(e) <- 5.5` — also `xmax()`, `ymin()`, `ymax()`
`union(e1, e2)`; `intersect(e1, e2)` — combine extents
`e * 2; e + 0.5` — grow / shrink; `align(e, r)` — snap to r's grid
`as.polygons(e, crs="EPSG:4326")` — extent → SpatVector
`draw()` — digitize an extent on a plotted map

GDAL & files

`gdal()` — GDAL version; `gdal(drivers=TRUE)` — supported formats
`describe("f.tif")`; `vector_layers("f.gpkg")` — what's in a file?
`setGDALconfig("GDAL_PAM_ENABLED", "FALSE")` — e.g. no .aux.xml files
`gdalCache(2000)` — GDAL cache size (MB)
`libVersion("proj")` — GDAL / PROJ / GEOS versions

Compare & validate

`compareGeom(x, y)` — same extent, resolution, crs?
`all.equal(x, y)`; `identical(x, y)` — geometry + values
`is.empty(e)`; `is.valid(v)`; `hasValues(r)`
`is.lonlat(x)`; `is.rotated(r)`; `is.flipped(r)`

Interactive (on a plotted map)

`click(r)` — query values by clicking; `click(v)`
`draw("polygon")` — digitize a polygon, line, extent or points
`zoom(r)` — zoom to a drawn extent; `sel(v)` — select by drawing

Based on terra 1.9-40 (2026) · terra is by Robert J. Hijmans · community cheat sheet, CC BY 4.0