Mikroklima-Warmup

Messpunkte, Raumfüllung und einfache Modellkritik

Worum es geht

Dieser Reader begleitet das Warmup für Studierende ohne Programmiererfahrung. Öffnen Sie das Projekt und führen Sie das vorhandene Skript abschnittsweise aus. Im Mittelpunkt steht ein basales Skript: Es wird geöffnet, blockweise ausgeführt und so gelesen, dass nachvollziehbar wird, welche Daten eingelesen werden, welche räumlichen Operationen stattfinden und welche Ergebnisdatensätze daraus entstehen. Die Studierenden sollen hier nicht selbst einen vollständigen Forschungsworkflow entwickeln, sondern trainieren, ein vorhandenes Skript kontrolliert zu nutzen, seine Zwischenschritte zu verstehen und die Ergebnisse fachlich zu lesen.

Die Messungen liegen an einzelnen Stationen vor. Das Skript liest Stationen und Höhenraster ein, wählt eine Temperaturspalte und berechnet vier flächenhafte Schätzungen: Voronoi, IDW, ein lineares Höhenmodell und Random Forest. Karten und Leave-One-Out-Fehler dienen zum Vergleich der Verfahren.

Führen Sie zuerst das unveränderte Skript aus. Prüfen Sie danach eine andere Temperaturspalte. Sie müssen dafür keinen Algorithmus schreiben. Für die weitere Kursarbeit üben Sie hier das Laden von Daten, relative Pfade, blockweises Ausführen und die Kontrolle von Rasterergebnissen.

Projekt herunterladen und öffnen

Für die Übung wird das komplette RStudio-Projekt als ZIP-Datei bereitgestellt:

Projekt-ZIP herunterladen

Die ZIP-Datei muss zuerst heruntergeladen und entpackt werden. Danach wird im entpackten Projektordner die `.Rproj`-Datei geöffnet. Erst dann liegen Skript, Datenordner und relative Pfade in der erwarteten Projektstruktur.

Das Skript

Das vollständige R-Skript kann hier heruntergeladen werden:

microclimate_warmup_minimal.R herunterladen

Skript anzeigen

Fachliche Idee: Mikroklima

Mikroklima beschreibt kleinräumige Unterschiede in Temperatur, Feuchte, Wind, Strahlung oder thermischer Belastung. Solche Unterschiede entstehen nicht zufällig. Sie hängen unter anderem mit Geländeform, Höhe, Hanglage, Vegetation, Verschattung, Bodenbedeckung, Bebauung, Kaltluftbildung und Luftaustausch zusammen.

Die Übung überträgt die Stationswerte mit vier unterschiedlichen Modellannahmen auf ein Höhenraster.

Zwischen Höhe, Luftdruck und Temperatur besteht ein grundlegender physikalischer Zusammenhang: Mit zunehmender Höhe nimmt der Luftdruck ab, Luft dehnt sich bei Aufstieg aus und kühlt dabei in der Regel ab. Deshalb ist Höhe häufig ein sinnvoller erster Prädiktor für räumliche Temperaturunterschiede.

Verwendet werden daher neben den Temperaturmessungen an einzelnen Stationen ein digitales Höhenmodell. Die Übung nutzt Höhe als einfache, fachlich begründbare Hilfsvariable ein, um zu zeigen, wie Punktmessungen mit einem flächenhaften Raster verbunden und in eine räumliche Ergebnisfläche übersetzt werden können.

1. Temperaturmessungen an mehreren Stationen.
2. Höhe aus einem digitalen Höhenmodell.

Das Ergebnis ist daher ein modellierter räumlicher Datensatz. Je nachdem, ob Nähe, Stationszuständigkeit, Höhe oder ein datengetriebenes Muster als wichtig angenommen wird, entsteht eine andere räumliche Struktur.

Methodische Idee: Raum füllen

Messstationen liefern punktuelle Werte. Eine räumliche Aussagefläche braucht aber Werte für viele Rasterzellen. Zwischen diesen beiden Ebenen liegt die zentrale methodische Aufgabe. Genau diese Aufgabe beantwortet das basale Skript Schritt für Schritt:

> Punktwerte müssen in eine flächenhafte Aussage übersetzt werden.

Das Skript zeigt vier einfache Wege, diese Übersetzung vorzunehmen. Sie stehen nicht für „richtig“ oder „falsch“, sondern für unterschiedliche Annahmen über räumliche Übertragung.

Voronoi

Voronoi ordnet jede Rasterzelle der nächstgelegenen Messstation zu. Der Wert dieser Station wird direkt auf die Zelle übertragen.

Kurzlogik:

> Nächste Station gewinnt.

Damit entsteht ein Ergebnisraster mit harten Grenzen. Jede Station erhält einen Zuständigkeitsbereich. Innerhalb dieses Bereichs ist der Wert überall gleich, an der Grenze zur nächsten Station springt er abrupt.

Voronoi ist methodisch einfach und gut lesbar. Es glättet nichts und erzeugt keine Zwischenwerte. Gleichzeitig ist die Annahme sehr grob: Temperatur ändert sich in der Realität selten sprunghaft an geometrischen Grenzen zwischen Stationen.

Code-Snippet: Voronoi / nearest station

```
vor_df <- gstat::idw(
  temp ~ 1,
  locations = pts,
  newdata = grid_sf,
  nmax = 1
)

map_vor <- make_map(vor_df$var1.pred, "Voronoi")
```

IDW

IDW bedeutet inverse distance weighting. Das Verfahren schätzt den Wert einer Rasterzelle aus nahegelegenen Stationen. Nahe Stationen erhalten ein höheres Gewicht, weiter entfernte Stationen ein geringeres Gewicht.

Kurzlogik:

> Nähe wirkt, aber nicht hart. Nahe Messpunkte zählen stärker als entfernte.

Im Skript wird `nmax = 4` verwendet. Jede Rasterzelle wird also nur aus den vier nächsten Stationen geschätzt. Dadurch bleibt die Schätzung lokal. Das Ergebnis wird glatter als Voronoi, aber weniger global geglättet als eine IDW-Variante, bei der alle Stationen in jede Schätzung eingehen.

IDW ist eine typische Einstiegsform der Interpolation. Es nutzt räumliche Nähe als zentrales Argument. Fachlich muss man aber prüfen, ob Nähe im konkreten Fall tatsächlich der wichtigste Zusammenhang ist. Bei Mikroklima kann das zutreffen, muss aber nicht ausreichen, weil Gelände, Schatten, Vegetation oder Kaltluft ebenfalls wirksam sein können.

Code-Snippet: IDW

```
idw_df <- gstat::idw(
  temp ~ 1,
  locations = pts,
  newdata = grid_sf,
  nmax = 4
)

map_idw <- make_map(idw_df$var1.pred, "IDW")
```

LM altitude

Das lineare Höhenmodell nutzt nicht primär die räumliche Nähe zwischen Stationen, sondern den Zusammenhang zwischen Temperatur und Höhe.

Kurzlogik:

> Temperatur wird über Höhe erklärt und auf das Höhenraster übertragen.

Das Modell schätzt eine lineare Beziehung zwischen Stationshöhe und Temperatur. Anschließend wird diese Beziehung auf jede Rasterzelle des DEM angewendet. Dadurch folgt das Ergebnis stark dem Höhenmuster.

Dieses Modell ist methodisch anders als Voronoi und IDW. Es füllt den Raum nicht über Stationsnachbarschaft, sondern über eine erklärende Variable. Wenn Höhe fachlich relevant ist, kann das sinnvoll sein. Wenn die gemessenen Temperaturunterschiede stärker durch Schatten, Vegetation oder lokale Kaltluft geprägt sind, kann ein reines Höhenmodell zu kurz greifen.

Code-Snippet: LM altitude

```
fit_lm <- lm(temp ~ altitude, data = st_drop_geometry(pts))

map_lm <- predict(dem, fit_lm)
names(map_lm) <- "LM_altitude"
```

RF warning

Random Forest ist ein flexibles, datengetriebenes Modell. Im Skript ist die Zielvariable weiterhin die gemessene Temperatur:

```
temp
```

Das Modell versucht also, Temperaturwerte vorherzusagen. Dafür erhält es drei Prädiktoren:

```
x
y
altitude
```

Damit kann RF Temperaturunterschiede aus räumlicher Position und Höhe ableiten. Das Modell fragt nicht direkt nach einem physikalischen Prozess wie Kaltluft, Verschattung oder Vegetationswirkung. Es schätzt die Temperatur anhand der Kombinationen aus Lage und Höhe in den Trainingsdaten.

Kurzlogik:

> RF lernt aus wenigen Stationen, welche Bereiche des Raums eher zu höheren oder niedrigeren Temperaturwerten passen.

Genau deshalb wird RF hier als Warnbeispiel geführt. Bei wenigen Messpunkten kann das Modell die vorhandenen Temperaturwerte scheinbar gut rekonstruieren, weil es die Punktlage und die Höhe flexibel aufteilt. Daraus entsteht ein Ergebnisraster, das plausibel aussehen und in der Validierung einen niedrigen Fehler haben kann.

Das bedeutet aber nicht automatisch, dass RF eine robuste mikroklimatische Erklärung gefunden hat. Das Modell kann Temperaturmuster erzeugen, die stark an die konkrete Lage der wenigen Messstationen gebunden sind. Eine gute Rückschätzung einzelner Stationen bedeutet deshalb nicht, dass die Ergebnisfläche als Prozessdarstellung fachlich überzeugend ist.

RF zeigt hier eine zentrale methodische Spannung:

> Das Modell sagt Temperatur vorher, aber es erklärt nicht automatisch, warum diese Temperatur räumlich so verteilt sein sollte.

Code-Snippet: RF warning

```
xy <- st_coordinates(pts)
pts$x <- xy[, 1]
pts$y <- xy[, 2]

fit_rf <- randomForest(
  temp ~ x + y + altitude,
  data = st_drop_geometry(pts),
  ntree = 200
)

rf_pred <- predict(fit_rf, newdata = grid)
map_rf <- make_map(rf_pred, "RF_warning")
```

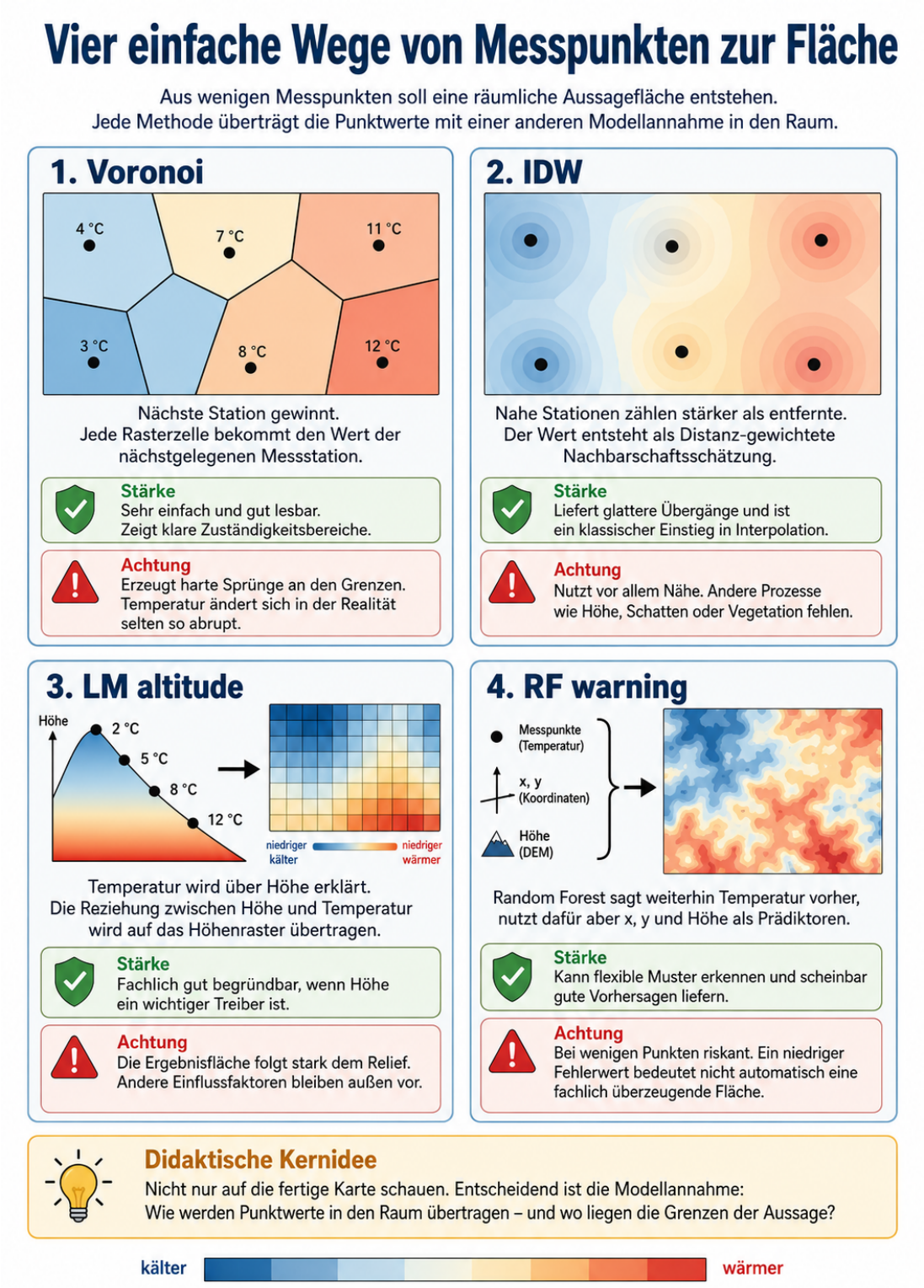


Figure 1: Vier einfache Wege von punktuellen Temperaturmessungen zu einer räumlichen Ergebnisfläche. Die Grafik zeigt, dass Voronoi, IDW, ein lineares Höhenmodell und Random Forest dieselben Messpunkte mit unterschiedlichen Modellannahmen in den Raum übertragen. Entscheidend ist daher nicht nur das fertige Kartenbild, sondern die Frage, welche Annahme die räumliche Aussage trägt und wo ihre fachliche Grenze lieg

Gültige Fläche

Das Skript begrenzt die Ausgabe auf den unmittelbaren Messraum. Dazu wird aus den Stationen eine konvexe Hülle gebildet und um 20 m gepuffert:

> Convex Hull der Stationen plus 20 m Puffer.

Diese Fläche ist keine perfekte Validitätsgrenze. Sie ist aber eine bewusste Einschränkung der Aussagefläche. Die Modelle werden nicht großflächig über das gesamte DEM ausgegeben, sondern nur im Umfeld der vorhandenen Messpunkte.

Das ist fachlich wichtig. Interpolation ist am stärksten dort vertretbar, wo Messpunkte den Raum tatsächlich stützen. Je weiter man sich vom Stationsnetz entfernt, desto stärker wird aus Interpolation eine spekulative Extrapolation.

Validierung

Die Modelle werden mit Leave-One-Out-Cross-Validation geprüft. Dabei wird jede Station einmal ausgelassen. Das Modell wird mit den übrigen Stationen berechnet und sagt anschließend den Wert der ausgelassenen Station voraus.

Der Ablauf ist:

1. Eine Station wird ausgelassen.
2. Das Modell wird mit den übrigen Stationen berechnet.
3. Die ausgelassene Station wird vorhergesagt.
4. Der Fehler zwischen Messwert und Vorhersage wird gespeichert.
5. Der Vorgang wird für alle Stationen wiederholt.

Aus den Fehlern wird der RMSE berechnet.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

Dabei ist `y_i` der gemessene Temperaturwert an Station `i`, `ŷ_i` der vorhergesagte Temperaturwert für die ausgelassene Station und `n` die Anzahl der Stationen.

Der RMSE gibt an, wie groß der typische Vorhersagefehler bei dieser Rückschätzung ist. Ein kleinerer RMSE bedeutet, dass die ausgelassenen Stationen im Durchschnitt besser getroffen wurden.

Für die Interpretation des flächenhaften Ergebnisses reicht das aber nicht. Der RMSE bewertet Punktvorhersagen an bekannten Stationsorten. Er sagt nicht automatisch, ob die gesamte

Ergebnisfläche fachlich plausibel modelliert wurde. Besonders bei wenigen und räumlich eng liegenden Punkten muss das Ergebnis zusätzlich nach seiner Modelllogik gelesen werden.

Technische Minimalleistung des basalen Skripts

Das Skript ersetzt keinen vollständigen Forschungsworkflow. Es zeigt die kleinste sinnvolle Kette, mit der aus Messpunkten, einem Höhenraster und einfachen Modellannahmen räumliche Ergebnisdatensätze entstehen. Genau deshalb eignet es sich als Trainingsskript: Die Studierenden können den Arbeitsweg nachvollziehen, ohne dass Rohdatenimport, Paketmanagement, Fehlerbehandlung oder ein vollständiger Analyseviewer den Kern überdecken.

1. Daten laden

Benötigt werden zwei Dateien: `data/climdata.rds` und `data/DEM1.tif`. `climdata.rds` enthält die Messstationen mit Geometrie und Temperaturspalten. `DEM1.tif` enthält die Höhe als Raster.

Code-Snippet: Daten laden

```
library(sf)
library(terra)
library(gstat)
library(randomForest)
library(leaflet)

m   <- readRDS("data/climdata.rds")
dem <- rast("data/DEM1.tif")
names(dem) <- "altitude"
```

2. Einen Zeitpunkt auswählen

Die Messdaten enthalten mehrere Temperaturspalten. Für den Warmup wird ein Zeitpunkt herausgegriffen. Damit arbeiten alle Modelle mit derselben Zielvariable.

Code-Snippet: Temperaturspalte setzen

```
m$temp <- m[["A20230830"]]
```

3. Koordinatensystem angleichen

Stationsdaten und DEM müssen im selben Koordinatensystem liegen. Das ist Voraussetzung für korrekte Höhenextraktion, Distanzen und Rasterüberlagerung.

Code-Snippet: CRS angleichen

```
m <- st_transform(m, crs(dem))
```

4. Höhe an den Stationen extrahieren

Die Stationshöhen werden direkt aus dem verwendeten DEM gezogen. Dadurch nutzen Stationsmodell und Ergebnisraster dieselbe Höhenbasis.

Code-Snippet: Höhe extrahieren

```
m$altitude <- terra::extract(dem, terra::vect(m))$altitude
```

5. Gültige Messpunkte auswählen

Es bleiben nur Stationen mit Temperaturwert und Höhenwert. Unvollständige Punkte werden aus der Modellierung entfernt.

Code-Snippet: vollständige Punkte wählen

```
pts <- m[!is.na(m$temp) & !is.na(m$altitude), c("temp", "altitude")]
```

6. Aussagefläche erzeugen

Die Stationen definieren den Bereich, in dem die Ergebnisse ausgegeben werden. Das DEM wird auf Convex Hull plus 20 m zugeschnitten und maskiert.

Code-Snippet: Aussagefläche und DEM-Maske

```
area <- st_sf(
  geometry = st_buffer(
    st_convex_hull(st_union(st_geometry(pts))),
    20
  )
)

dem <- crop(dem, vect(area))
dem <- mask(dem, vect(area))
names(dem) <- "altitude"
```

7. Vorhersagegrid erzeugen

Aus dem DEM wird eine Tabelle mit Rasterzellen, Koordinaten und Höhe gebaut. Diese Tabelle ist die technische Brücke zwischen Raster und Modell.

Code-Snippet: Vorhersagegrid

```
grid <- as.data.frame(dem, xy = TRUE, cells = TRUE, na.rm = FALSE)
names(grid)[4] <- "altitude"
grid <- grid[!is.na(grid$altitude), ]

grid_sf <- st_as_sf(
  grid,
  coords = c("x", "y"),
  crs = st_crs(pts),
  remove = FALSE
)
```

8. Ergebnisraster berechnen

Das Skript erzeugt vier Ergebnisraster. Alle verwenden dieselbe Aussagefläche. Der Unterschied liegt in der Modellannahme.

Code-Snippet: Ergebnisraster berechnen

```
make_map <- function(pred, name) {
  r <- dem
  values(r) <- NA
  values(r)[grid$cell] <- pred
  names(r) <- name
  r
}

xy <- st_coordinates(pts)
pts$x <- xy[, 1]
pts$y <- xy[, 2]

fit_lm <- lm(temp ~ altitude, data = st_drop_geometry(pts))
map_lm <- predict(dem, fit_lm)
names(map_lm) <- "LM_altitude"

vor_df <- gstat::idw(temp ~ 1, locations = pts, newdata = grid_sf, nmax = 1)
map_vor <- make_map(vor_df$var1.pred, "Voronoi")

idw_df <- gstat::idw(temp ~ 1, locations = pts, newdata = grid_sf, nmax = 4)
map_idw <- make_map(idw_df$var1.pred, "IDW")

fit_rf <- randomForest(
  temp ~ x + y + altitude,
  data = st_drop_geometry(pts),
  ntree = 200
)

rf_pred <- predict(fit_rf, newdata = grid)
map_rf <- make_map(rf_pred, "RF_warning")
```

9. Modelle validieren

Die Modelle werden mit Leave-One-Out-Cross-Validation geprüft. Jede Station wird einmal ausgelassen und aus den übrigen Stationen zurückgeschätzt.

Code-Snippet: LOOCV und RMSE

```r
rmse_fun <- function(e) sqrt(mean(e * e, na.rm = TRUE))

lm_cv <- rep(NA, nrow(pts))
rf_cv <- rep(NA, nrow(pts))

for (i in 1:nrow(pts)) {
  train <- pts[-i, ]
  test  <- pts[i, ]

  fit_lm_i <- lm(temp ~ altitude, data = st_drop_geometry(train))
  lm_cv[i] <- predict(fit_lm_i, newdata = st_drop_geometry(test))

  fit_rf_i <- randomForest(
    temp ~ x + y + altitude,
    data = st_drop_geometry(train),
    ntree = 200
  )
  rf_cv[i] <- predict(fit_rf_i, newdata = st_drop_geometry(test))
}

vor_model <- gstat::gstat(
  formula = temp ~ 1,
  locations = pts,
  nmax = 1,
  set = list(idp = 2)
)

vor_cv <- gstat::gstat.cv(vor_model, nfold = nrow(pts))

idw_model <- gstat::gstat(
  formula = temp ~ 1,
  locations = pts,
  nmax = 4,
  set = list(idp = 2)
)

idw_cv <- gstat::gstat.cv(idw_model, nfold = nrow(pts))

rmse <- data.frame(
  model = c("LM altitude", "Voronoi", "IDW", "RF warning"),
  RMSE = c(
    rmse_fun(pts$temp - lm_cv),
    rmse_fun(vor_cv$residual),
    rmse_fun(idw_cv$residual),
    rmse_fun(pts$temp - rf_cv)
  )
)
```

10. Ergebnisraster als statische Plots darstellen

Die vier Ergebnisraster werden zunächst nebeneinander als statische Plots dargestellt. Alle verwenden dieselbe Farbskala, damit die räumlichen Muster vergleichbar bleiben.

Code-Snippet: Ergebnisraster darstellen

```
maps <- c(map_vor, map_idw, map_lm, map_rf)
z <- range(c(pts$temp,
             global(maps, "min", na.rm = TRUE)[, 1],
             global(maps, "max", na.rm = TRUE)[, 1]), na.rm = TRUE)
# Kleine Reserve für Rundung bei der Kartenprojektion.
z <- z + c(-0.01, 0.01)

par(mfrow = c(2, 2))

for (i in 1:nlyr(maps)) {
  plot(maps[[i]], range = z, main = names(maps)[i])
  points(pts, pch = 19, cex = 0.8)
}

par(mfrow = c(1, 1))

print(rmse)
```

11. Ergebnisse als Leaflet-Karte darstellen

Am Ende werden die gleichen Ergebnisraster zusätzlich in eine interaktive Leaflet-Karte eingebunden. Die Karte ersetzt die statischen Plots nicht, sondern ergänzt sie: Die Studierenden können zwischen den Modellannahmen umschalten, die Messpunkte sichtbar lassen und dadurch prüfen, wie stark sich die räumliche Aussage je nach Verfahren verändert.

Code-Snippet: Leaflet-Karte

```
pal <- leaflet::colorNumeric(
  palette = hcl.colors(100, "RdYlBu", rev = TRUE),
  domain = z,
  na.color = "transparent"
)

pts_leaflet <- st_transform(pts, 4326)
pts_xy <- st_coordinates(pts_leaflet)

leaflet_map <- leaflet::leaflet()
leaflet_map <- leaflet::addProviderTiles(
  leaflet_map,
  leaflet::providers$CartoDB.Positron
)

for (i in 1:nlyr(maps)) {
  leaflet_map <- leaflet::addRasterImage(
    leaflet_map,
    x = raster::raster(maps[[i]]),
    colors = pal,
    opacity = 0.70,
    group = names(maps)[i]
  )
}

leaflet_map <- leaflet::addCircleMarkers(
  leaflet_map,
  lng = pts_xy[, 1],
  lat = pts_xy[, 2],
  radius = 4,
  stroke = FALSE,
  fillOpacity = 0.9,
  label = paste0("Temperatur: ", round(pts$temp, 2), " °C"),
  group = "Messpunkte"
)

leaflet_map <- leaflet::addLayersControl(
  leaflet_map,
  baseGroups = names(maps),
  overlayGroups = "Messpunkte",
  options = leaflet::layersControlOptions(collapsed = FALSE)
)

leaflet_map <- leaflet::addLegend(
  leaflet_map,
  pal = pal,
  values = z,
  title = "Temperatur",
  opacity = 0.70
)

leaflet_map <- leaflet::hideGroup(leaflet_map, names(maps)[-1])
```

Was nicht enthalten ist

Viele Bestandteile, die in einer umfangreichen Analyse zwingend oder sinnvoll wären, werden bewusst weggelassen:

```
Ecowitt-Rohdatenimport
automatische Paketinstallation
tryCatch-Sicherheitslogik
Backup-Dateien
R*-Tuning
Variogramm-Interpretation
KED
OK
GAM
TPS
große Benchmark-Tabellen
vollständiger Ergebnisviewer
PDF-Report
Batch-Verarbeitung aller Zeitpunkte
```

Diese Elemente können später wichtig werden. Für diesen Warmup würden sie den Kern verdecken: Punktdaten, Raster, Aussagefläche, Interpolation, Modellannahme und erste Validierung.

Ergebnisse sichern

Das heruntergeladene Skript schreibt `output/warmup/temperatur_modelle.tif` und `output/warmup/loocv_rmse.csv`. Öffnen Sie das Raster in QGIS und stellen Sie die vier Bänder mit derselben Temperaturskala dar.

Notieren Sie die verwendete Temperaturspalte und die Zahl gültiger Stationen. Ordnen Sie die vier Bänder den Verfahren Voronoi, IDW, Höhenmodell und Random Forest zu. Vergleichen Sie an einem Stationsort und zwischen zwei Stationen die vorhergesagten Werte. Die RMSE-Tabelle enthält Leave-One-Out-Fehler an Stationsorten; eine unabhängige Flächenvalidierung liegt hier nicht vor.