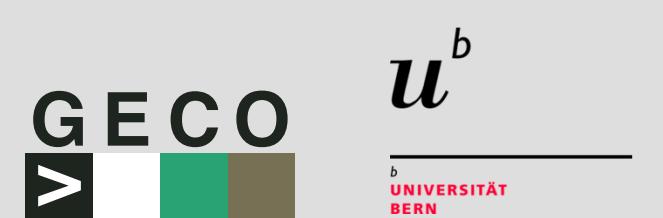




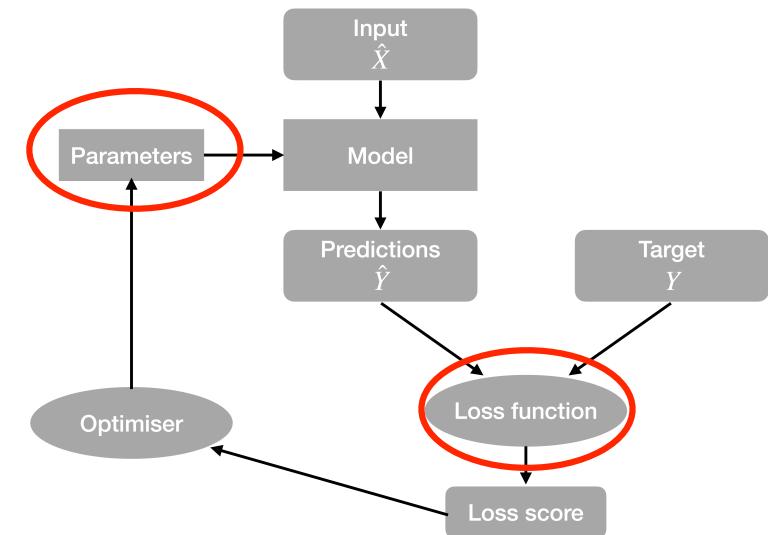
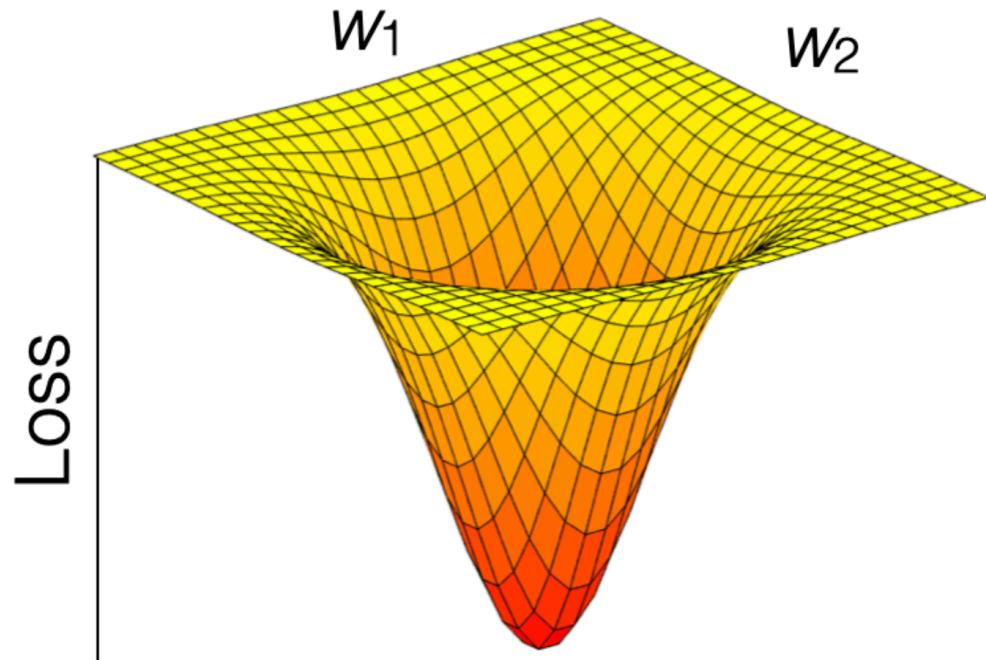
# Applied Geodata Science I

# Session 11

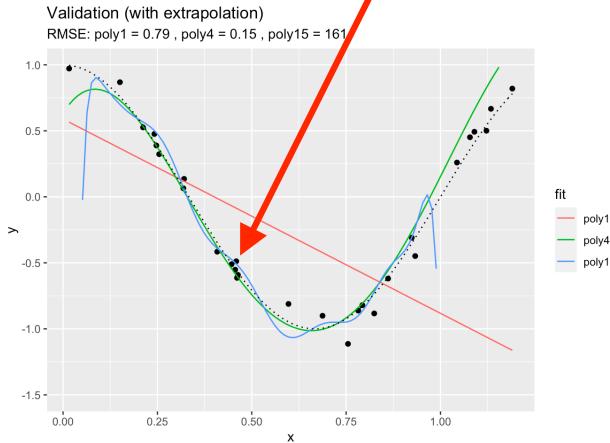
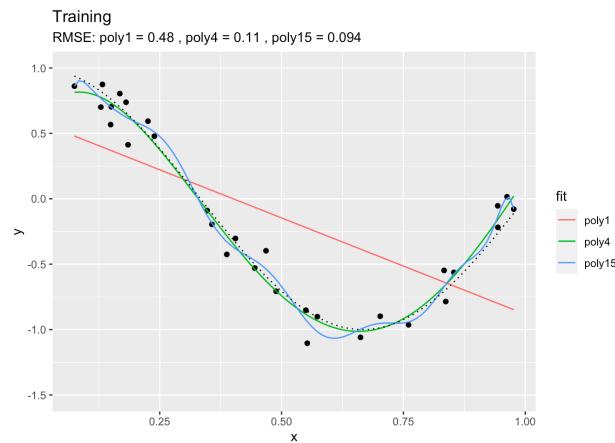
Prof. Dr. Benjamin Stocker  
05.05.2025



# Loss as a function of the model parameters (coefficients)



# Predicting to new data



These are not the same data points as used for training.  
New data!

- Model is trained on data  
→ yields model object: mod

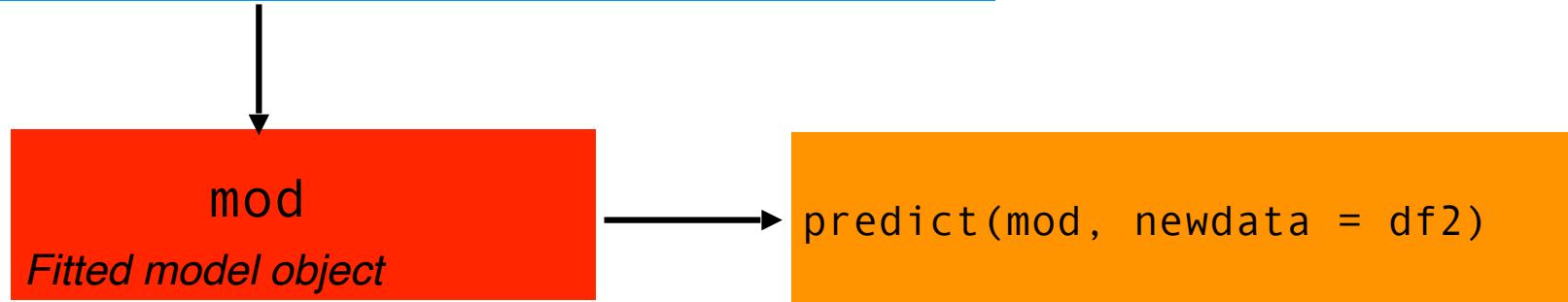
- Trained model is applied to new data:  
`predict(mod, newdata = ...)`

# Predicting to new data

```
mod <- caret::train(..., data = df1)  
or: mod <- lm(..., data = df1)  
or...
```

*Model training*

```
df1  
variables: y, x1, x2, x3
```



- Contains information about model structure (model type, formula) and coefficients (parameters)
- df2 must contain variables (columns) used in the formula.

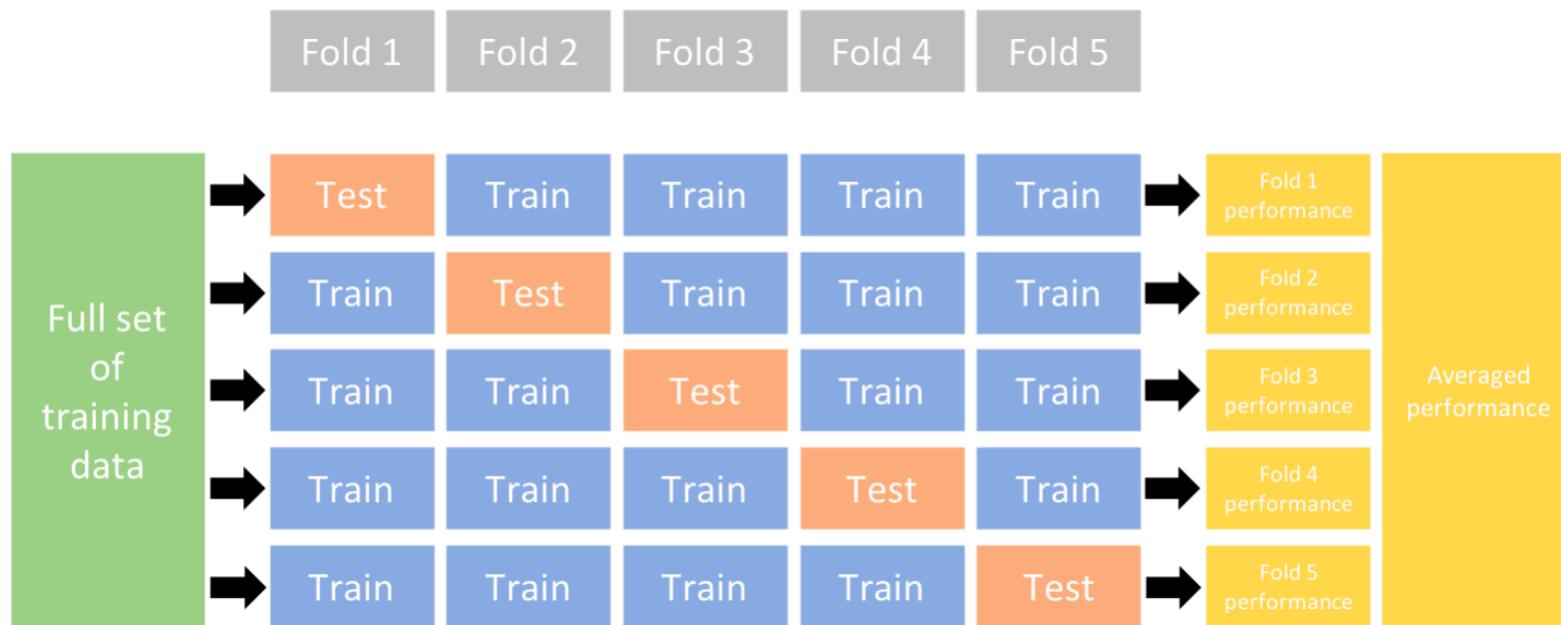
# Predicting to new data

```
eval_model <- function(mod, df_train, df_test, return_metrics = FALSE){  
  require(magrittr) # part of dplyr, required for the "old" pipe  
  # add predictions to the data frames----  
  df_train <- df_train %>%  
    drop_na() %>% # magrittr pipe necessary here for the dot ('.') evaluation  
    mutate(fitted = predict(mod, newdata = .))  
  
  df_test <- df_test %>%  
    drop_na() %>%  
    mutate(fitted = predict(mod, newdata = .))
```

# K-fold cross validation

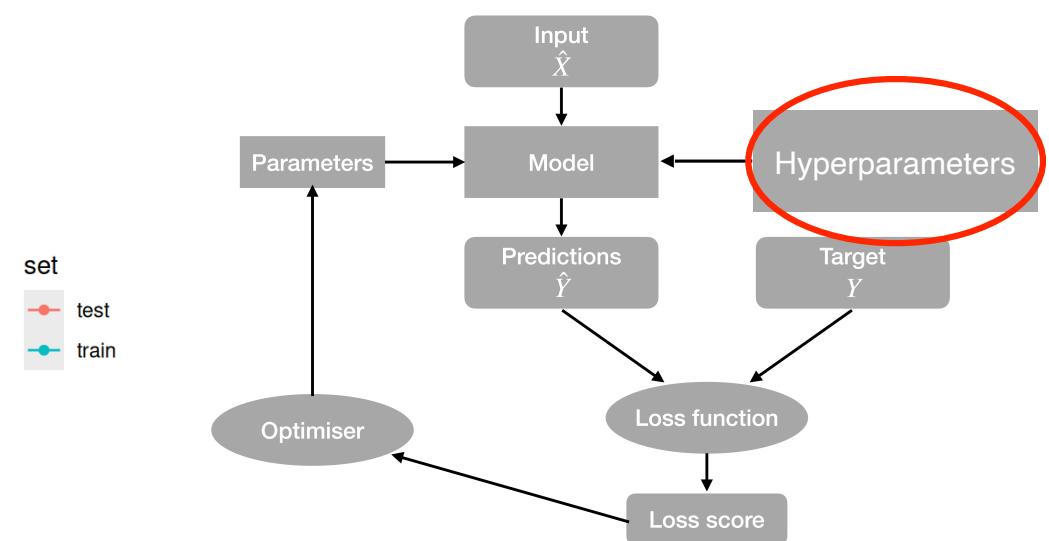
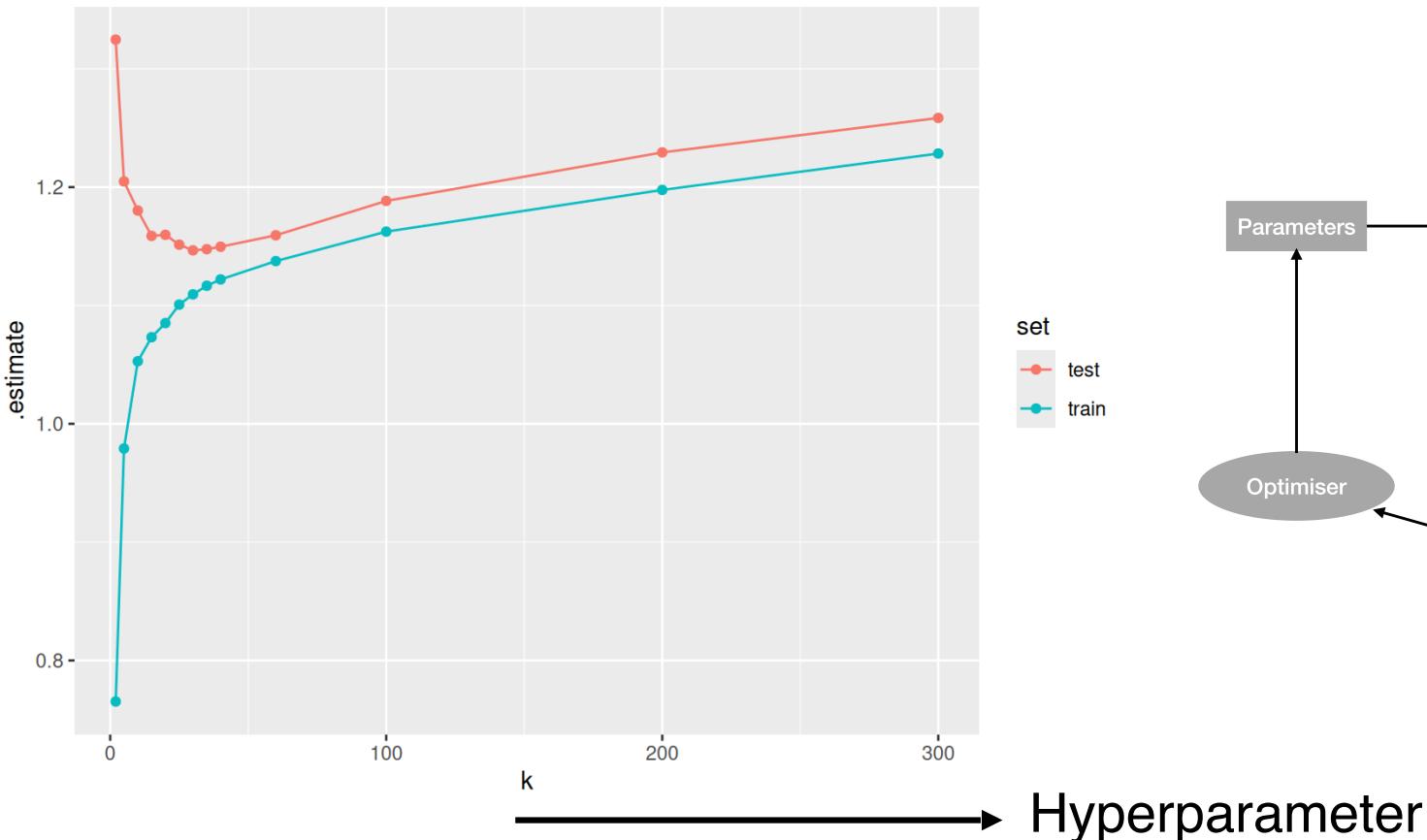


# 5-fold cross-validation



Boehmke & Greenwell (2019) *Hands On Machine Learning in R*

# Loss profile



# Hyperparameter tuning

- Hyperparameters determine the model structure and training setup.
- Hyperparameters are algorithm-specific.
  - $k$  in KNN
  - min.node.size, mtry, splitrule in Random Forest (implementation using {ranger} with {caret})
  - Not to confuse with model parameters/coefficients.

# Hyperparameter tuning

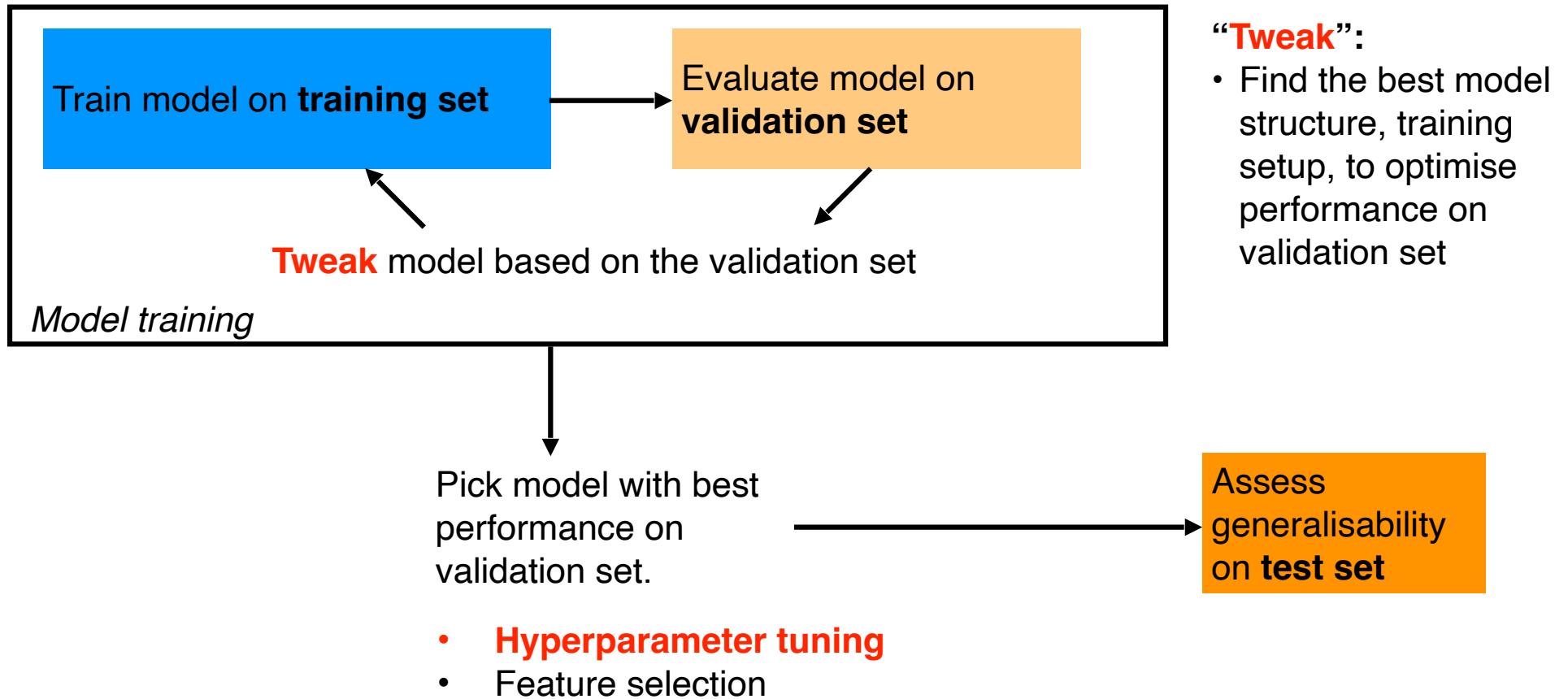


Figure adopted from [Google Machine Learning Crash Course](#)

# Hyperparameter tuning

## The model training workflow

- Read and wrangle data (no pre-processing!)
- Split data into train and test
- Specify formula and pre-processing recipe (allocate “roles” to variables)
- *Hyperparameter loop*
  - *Resample loop*
    - Split into validation and (remaining) train set
    - Apply pre-processing (transform train and validation set based on parameters determined on train set)
    - Fit model on train set
    - Predict on validation set
    - Get loss from prediction on validation set
    - Average loss across resamples
  - Pick best hyperparameter based on average loss across resamples

```
# Read and wrangle data -----
# (no pre-processing!). Here, no wrangling done (cleaning, variable selection, etc.)
df <- read.csv(paste0(here(),"data/df_daily_exercise_supervisedmlm.csv"))

# Split data into train and test -----
# Specify formula and pre-processing recipe -----
# (for predict roles)
nam_target <- "GPP_NT_VUT_REF"
nams_predictors <- c("TA_F", "SW_IN_F", "VPD_F")

# Hyperparameter loop -----
vec_k <- (2, 5, 10, 15, 20, 25, 30, 35, 40, 60, 100, 200, 300)

# specify number of cross-validation folds
n_folds <- 5

# initialise loss across resamples, to be kept for each hyperparameter choice
loss_avg <- c()

for (i_k in seq(length(vec_k))){

  # re-shuffle rows in data frame (not required)
  df <- df[sample(nrow(df),]

  # create folds:
  # determine row indices to be allocated to each fold
  # i.e. n_folds total folds, total number of rows
  nrows_per_fold <- ceiling(nrow(df) / n_folds)
  idx <- rep(seq(1:n_folds), each = nrows_per_fold)
  resample_folds <- split(1:nrow(df), idx[1:nrow(df)]) # yields a list of vectors containing the row indices

  # resample loop -----
  # initialise vector of loss per resample
  loss_vec <- c()

  for (i_resample in seq(n_folds)){

    # Split into validation and (remaining) train set -----
    # validation set
    df_valid <- df[resample_folds[[i_resample]], c(nam_target, nams_predictors)]

    ## remaining training set
    df_train <- df[-resample_folds[[i_resample]], c(nam_target, nams_predictors)]

    ## Apply pre-processing -----
    # set sample size based on training data parameters
    mean_byvar <- c()
    sd_byvar <- c()
    df_train.cs <- df_train %>%
      mutate_all(funs(. - mean(.), . / sd(.)))
    df_valid.cs <- df_valid %>%
      mutate_all(funs(. - mean(.), . / sd(.)))

    # center and scale each predictor
    for (ivar in nams_predictors){

      # determine mean and sd for centering and scaling
      mean_byvar[ivar] <- mean(df_train[,ivar], na.rm = TRUE)
      sd_byvar[ivar] <- sd(df_train[,ivar], na.rm = TRUE)

      # center and scale training data
      df_train.cs[,ivar] <- df_train[,ivar] - mean_byvar[ivar]
      df_train.cs[,ivar] <- df_train[,ivar] / sd_byvar[ivar]

      # center and scale validation data
      # important: use parameters (mean and sd) determined on training data
      df_valid.cs[,ivar] <- df_valid[,ivar] - mean_byvar[ivar]
      df_valid.cs[,ivar] <- df_valid[,ivar] / sd_byvar[ivar]

    }

    # add unmodified target variable
    df_valid.cs[,nam_target] <- df.valid[,nam_target]
    df_train.cs[,nam_target] <- df.train[,nam_target]

    # Fit model on train set -----
    # train using the scaled training data
    mod <- caret::knnreg(df_train.cs[,nams_predictors],
                         df_train.cs[,nam_target],
                         k = vec_k[i_k])
  }
}
```

*Switch to:*

add\_material/full\_workflow\_no\_caret.R

# Specifying recipes

## The model training workflow

- Read and wrangle data (**no pre-processing!**)
- Split data into train and test
- Specify formula and pre-processing **recipe**
- *Hyperparameter loop*
  - *Resample loop*
    - **Apply pre-processing**  
(transform train and validation set based on parameters determined on train set)
    - Fit model on train set
    - Predict on validation set
    - Get loss from prediction on validation set
  - Average loss across resamples
- Pick best hyperparameter based on average loss across resamples

*How many times is pre-processing transformation applied here?*

```
pp <- recipes::recipe(GPP_NT_VUT_REF ~ SW_IN_F + VPD_F + TA_F,  
                      data = df) |>  
  recipes::step_center(all_numeric(), -all_outcomes()) |>  
  recipes::step_scale(all_numeric(), -all_outcomes())  
  
mod <- caret::train(  
  pp,  
  data = df %>%  
    drop_na(),  
  method = "ranger",  
  metric = "RMSE",  
  trControl = trainControl(  
    method = "cv",  
    number = 5,  
    savePredictions = "final"  
  ),  
  tuneGrid = expand.grid(  
    .mtry = seq(from = 3, to = 19, by = 2),  
    .min.node.size = 5:9,  
    .splitrule = "variance"  
  ),  
  # arguments specific to "ranger" method  
  replace = FALSE,  
  sample.fraction = 0.5,  
  num.trees = 12,  
  seed = 1982  
)
```

# Specifying recipes

```
pp <- recipes::recipe(GPP_NT_VUT_REF ~ SW_IN_F + VPD_F + TA_F,  
  . . . . . . . . . . . data = df) |>  
recipes::step_center(all_numeric(), -all_outcomes()) |>  
recipes::step_scale(all_numeric(), -all_outcomes())
```

```
> pp  
Recipe  
  
Inputs:  
  
  role #variables  
  outcome      1  
  predictor    3  
  
Operations:  
  
Centering for all_numeric(), -all_outcomes()  
Scaling for all_numeric(), -all_outcomes()
```

## From Chapter 10 *Supervised ML I*:

To actually transform the data, we first have to “prepare” the recipe:

```
pp_prep <- recipes::prep(pp, training = daily_fluxes_train)
```

Finally we can actually transform the data. That is, “juice” the prepared recipe.

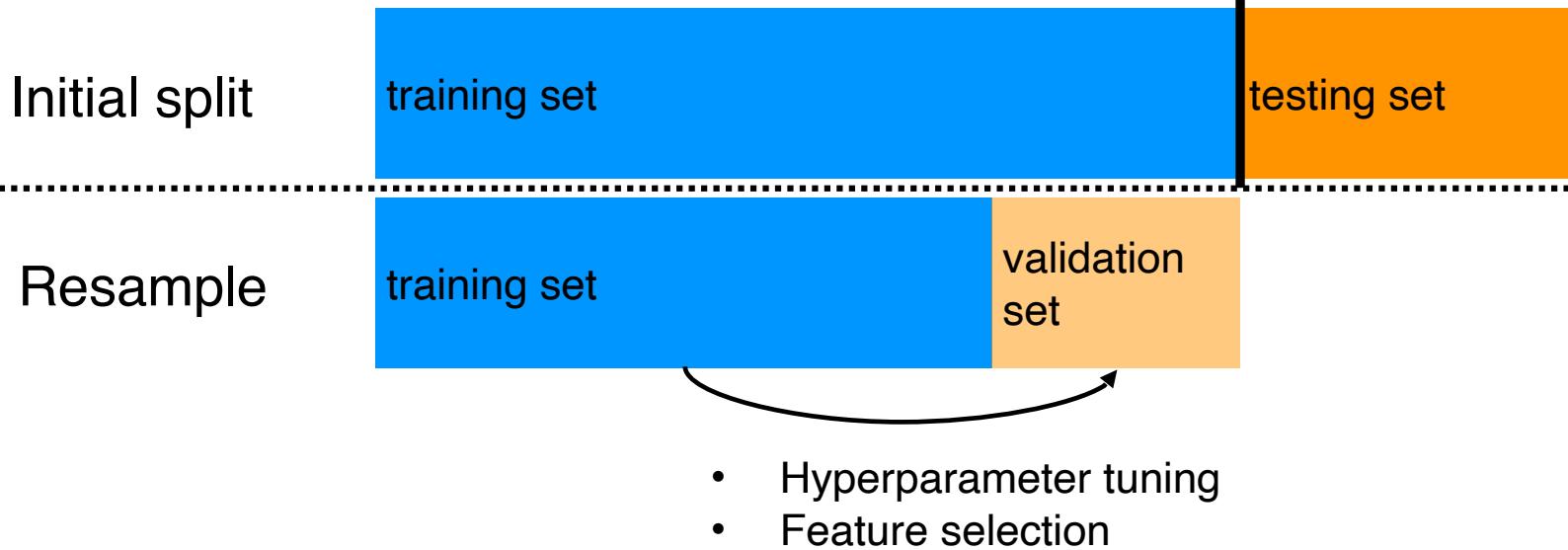
```
daily_fluxes_juiced <- recipes::juice(pp_prep)
```

Note, if we are to apply the prepared recipe to *new* data, we’ll have to `bake()` it.

```
daily_fluxes_baked <- recipes::bake(pp_prep, new_data = daily_fluxes_train)  
  
# confirm that juice and bake return identical objects when given the same data  
all_equal(daily_fluxes_juiced, daily_fluxes_baked)
```

```
## [1] TRUE
```

# Testing vs. validation set

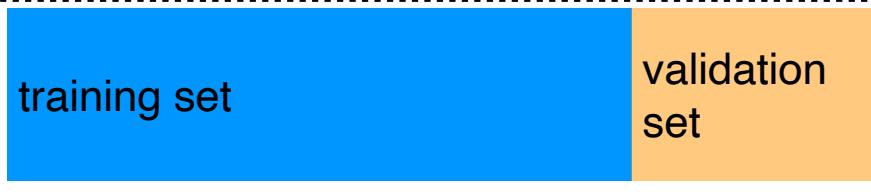


# K-fold cross validation

Initial split



Folds

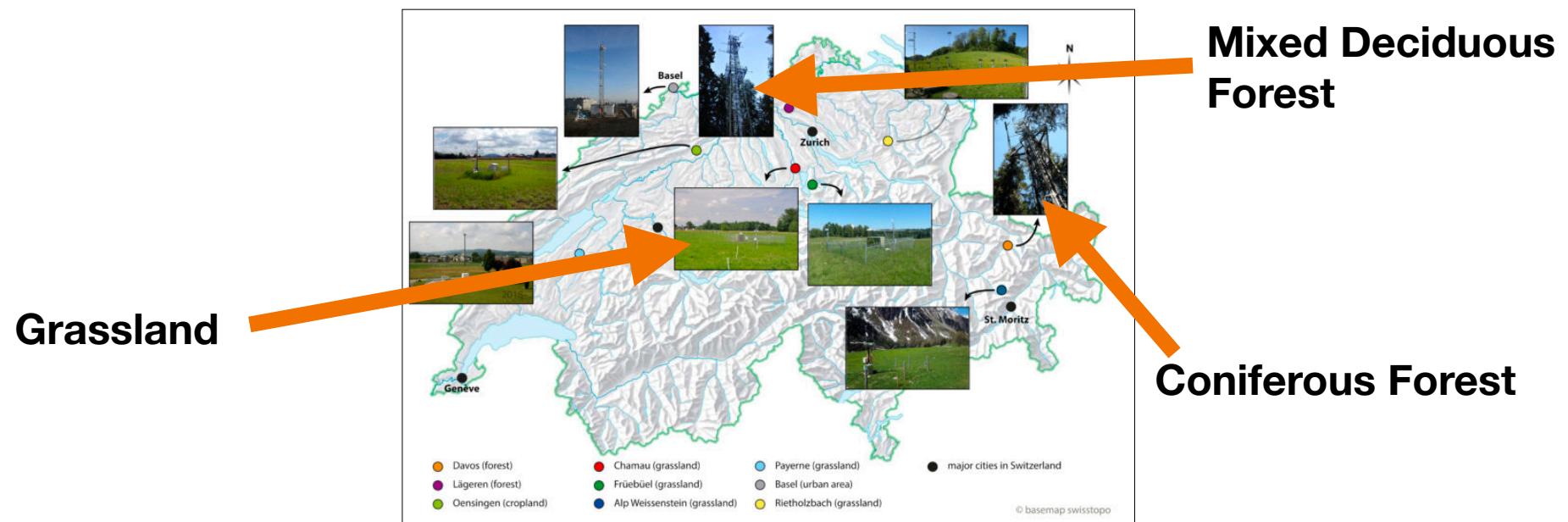


...

# Report Exercise 10.4: Spatial Upscaling

The data used for model training depends on our research question!

- Do we want a model that predicts GPP across Switzerland?
- Do we want a model that captures GPP responses of a deciduous forest?
- ...

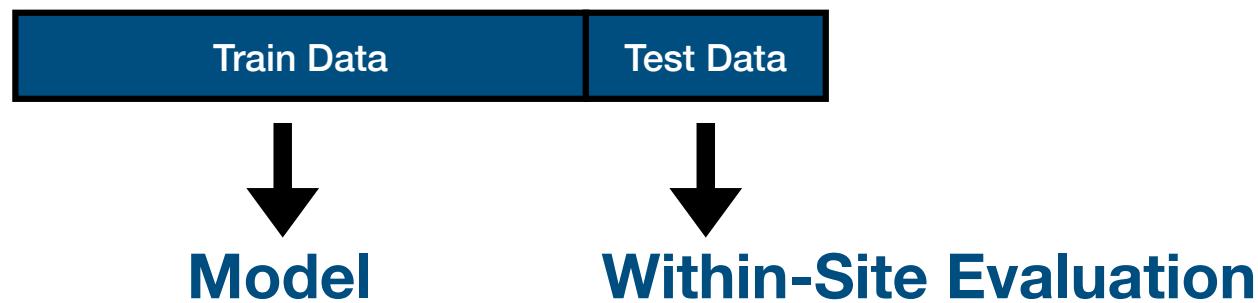


# Report Exercise 10.4: Spatial Upscaling

Training and testing on **one** site (Within-Site Evaluation)

Training on 1 site:

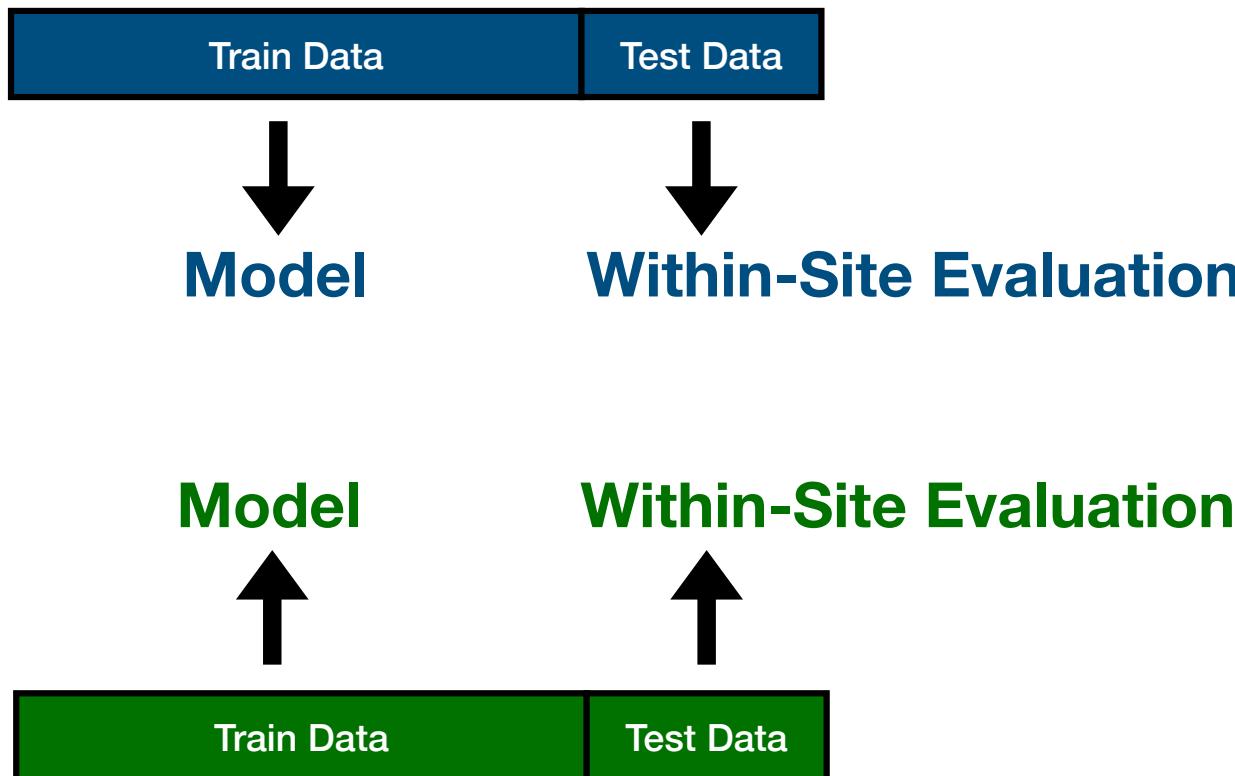
- Target is to predict response of **this site to unseen** environmental conditions
- Model is trained on time-series at that site
- Captures site-specific characteristics



# Report Exercise 10.4: Spatial Upscaling

How generalisable is our model?

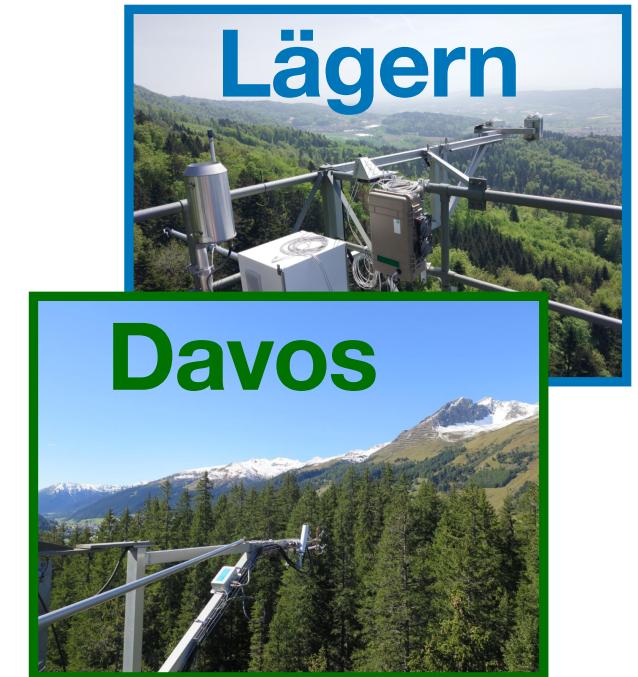
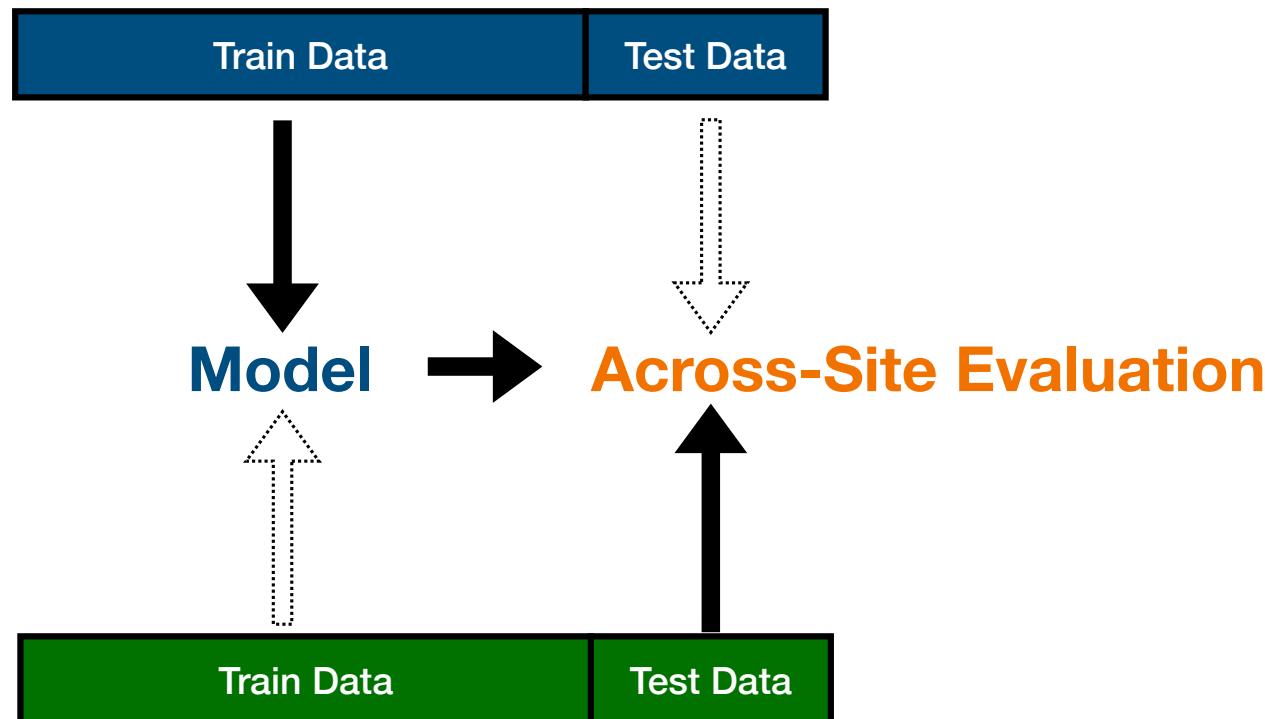
*Compare within-site predictions ...  
considering different metrics*



# Report Exercise 10.4: Spatial Upscaling

How generalisable is our model? → Out-of-sample Prediction

*For across-site predictions, make sure to implement a train and test setup that enables a true out-of-sample prediction test.*



# Report Exercise 10.4: Spatial Upscaling

## Capturing spatial variability - training on **two** sites

*Train a single model with training data pooled from both sites and predict with this single model on the test data of both sites.*

