First Lasso experiments

Set.seed was set to 4 to make results reproducable

prerun this please: code_below is full gitcode from 'heiko_wolfgang_code_test.R"

```r
{
  {
library(magrittr)
library(dplyr)
library(glmnet)
library(purrr)
library(tidyverse)
#?glmnet
library(coefplot) #for extracing non 0 coef
#install.packages("tidyverse")
library(tidyverse)
library(pROC)
library(fhpredict)
library(tidyverse)

library(kwb.flusshygiene)



#if (FALSE)



  #### Laden von Testdaten ###################

  rivers <- c("havel")
  river <- "havel"
  #river_paths <- kwb.flusshygiene::get_paths()[paste0(rivers, "data")]

  river_paths <- list(havel = "Y:/SUW_Department/Projects/FLUSSHYGIENE/Data-Work packages/Daten/Daten_Te

  river_paths <- list(havel = "/Users/heiko.langer/Masterarbeit_lokal/Data_preprocess/Daten_TestPackage_

  river_data <- lapply(river_paths, kwb.flusshygiene::import_riverdata)

  river <- "havel"

  names(river_data) <- rivers


#
  calc_t <- function (datalist=river_data$havel, onlysummer) {
    #heiko
```

```r
#datalist<- river_data1$havel
phy_data <- datalist[-1] # Entfernung der Hygienedaten

if(onlysummer==T){
  hyg_df <- subset(datalist[[1]],
            subset = lubridate::month(datum) %in% 5:9) # Filtern nach Sommer, warum hier 5:9 und be

  data_summer <- lapply(phy_data, function(df){

    df <- subset(df, subset = lubridate::month(datum) %in% 4:9)

  }
  )
}


# z_standardize <- function (x) {

#   y = (x - mean(x, na.rm=T))/sd(x, na.rm=T)

# }

log_transorm_rain <- function(df) { #log transforming rain data

  for (site in names(df)[-1]) { # every col gets treatment

    df2 <- subset(df, select = c("datum", site))

    if (grepl("^r_.*",site)) { # rain gets log-transformed and 1/sigma2

      df2[[site]] <- log(df2[[site]]+1)

      # df2[[site]] <- df2[[site]]/sd(df2[[site]], na.rm=T)

    } #else {

    #   df[[site]] <- z_standardize(df2[[site]]) # standardize

    # }

    df[[site]] <- df2[[site]]

  }

  return(df)

}

data_t <- lapply(data_summer, log_transorm_rain)

result <- append(list(hyg_df), data_t)

names(result) <- names(datalist)
```

```r
  return(result)

}
### Anwenden von calc_t auf Inputliste

river_data_ts <- lapply(river_data, function(river_list){

  river_ts <- calc_t(river_list, onlysummer = T) # use function

  add_meancol <- function (df) { # for rain and i #edit: + ka #2ndedit: + q

    prefix <- unique(sub("([a-z])_.*","\\1",names(df)[-1]))

    for (pre in prefix) {

      df2 <- dplyr::select(df, dplyr::starts_with(pre))

      df[,paste0(pre,"_mean")] <- rowMeans(df2, na.rm=T)

    }



    return(df)

  }

  add_sumcol <- function (df) { # originally for ka, but not used

    prefix <- unique(sub("([a-z])_.*","\\1",names(df)[-1]))

    if (length(df) > 2)

      df[,paste0(prefix,"_sum")] <- rowSums(df[,-1], na.rm=T)

    return(df)

  }



  q_pos <- grep("^q", names(river_ts)[-1])+1


  if (length(q_pos) == 1)

    river_ts[[q_pos]] <- add_meancol(river_ts[[q_pos]])

  ka_pos <- grep("^ka", names(river_ts)[-1])+1

  if (length(ka_pos) == 1)

    river_ts[[ka_pos]] <- add_meancol(river_ts[[ka_pos]])
  i_pos <- grep("^i", names(river_ts)[-1])+1

  if (length(i_pos) == 1)

    river_ts[[i_pos]] <- add_meancol(river_ts[[i_pos]])

  r_pos <- grep("^r", names(river_ts)[-1])+1

  river_ts[[r_pos]] <- add_meancol(river_ts[[r_pos]])

  return(river_ts)

})



rm(river_data,calc_t)

river = "havel"
pattern = "(i_mean|q_mean_mean|r_mean_mean|ka_mean_mean)"
riverdata <- river_data_ts[[river]]

# prepare variables out of all cominations (given by pattern)

# variables for interaction get replaced by q_new (remove q_old)

vars1 <- (riverdata[-1] %>% unroll_physical_data() %>%

            lapply(names) %>% unlist() %>% unique())[-1]

vars2 <- vars1[stringr::str_detect(vars1, pattern)]



# prepare formulas

data <- process_model_riverdata(riverdata, c("log_e.coli", vars2)) %>%

  dplyr::select(-datum)





data <- na.omit(data)

data <- data %>% filter(log_e.coli > log10(15)) #why-heiko?


#Definition of models

# Definition of null and full models
#stepwise models
```

```r
  null <- lm(log_e.coli ~ 1, data = data) #model with only 1 variable

  full <- lm(log_e.coli ~ .^2, data = data)

  #heiko models


}
  {
    #heiko
    {


      get_coef_1se_cv <- function(df){
        tmp_coeffs <- coef(df, s = "lambda.1se")
        a <- data.frame(name = tmp_coeffs@Dimnames[[1]][tmp_coeffs@i + 1], coefficient = tmp_coeffs@x)
        return(a)
      }
      get_coef_min_cv <- function(df){
        tmp_coeffs <- coef(df, s = "lambda.min")
        a <- data.frame(name = tmp_coeffs@Dimnames[[1]][tmp_coeffs@i + 1], coefficient = tmp_coeffs@x)
        return(a)
      }



      get_coef_fixed_lambda <- function(df,lambda){
        tmp_coeffs <- coef(df, s = lambda)
        a <- data.frame(name = tmp_coeffs@Dimnames[[1]][tmp_coeffs@i + 1], coefficient = tmp_coeffs@x)
        return(a)
      }


    }
    get_formula_variable_names <- function(formula_a,df){
      mf <- model.frame(formula_a, data=df)
      mt <- attr(mf, "terms")
      predvarnames <- attr(mt, "term.labels")
      predvarnames
    }

    #lasso
    #build/integrate here into folds to train with same cross validation
    #fold1<-train_rows[[1]]

    #training_heiko<-data[fold1,]

    part1<-names(data)[1]
    form<-formula(paste(part1," ~ (.)^2"))
    get_formula_variable_names(form,data)
        #training_heiko_features <- (training_heiko%>% select(-log_e.coli))
    #sparse.model.matrix(form, training_heiko)
    #form <- log_e.coli ~ (.)^2
    #training_heiko_features_matrix <- (data.frame.2.sparseMatrix(training_heiko_features))
    train_sparse <- sparse.model.matrix(form, data) #data must be dataframe
    #train_sparse <- sparse.model.matrix(training_heiko$log_e.coli~(.)^2, training_heiko[,3:ncol(traini
    #form <- Y ~ (x + y + z)^2
    #testing_heiko<-data[-fold1,]

   # test_sparse <- sparse.model.matrix(testing_heiko$log_e.coli~., testing_heiko[,3:ncol(testing_heiko
    set.seed(4)

    {
    fit_lasso_base <- glmnet(train_sparse, data$log_e.coli , na.rm =T, standardize = F, alpha = 1,relax

    fit_lasso_base_cross <- cv.glmnet(train_sparse, data$log_e.coli,type.measure="mse", alpha=1, family


    fit_lasso_base_stand <- glmnet(train_sparse, data$log_e.coli , na.rm =T, standardize = T, alpha = 1
    fit_lasso_base_cross_stand <- cv.glmnet(train_sparse, data$log_e.coli,type.measure="mse", alpha=1, 

    #par(mfrow=c(2,2))
    #plot(fit_lasso_base, xvar="lambda", label = T, main = "lasso_base")
    #plot(fit_lasso_base_cross,main="LASSO")

    #plot(fit_lasso_base_stand, xvar="lambda", label = T, main = "lasso_base_stand")
    #plot(fit_lasso_base_cross_stand,main="LASSO")

    #plot(fit_elnet_base, xvar="lambda", label = T, main = "elnet_base")
    #plot(fit_elnet_base_cross,main="elnet")

    #plot(fit_elnet_base_stand, xvar="lambda", label = T, main = "elnet_base_stand")
    #plot(fit_elnet_base_cross_stand,main="elnet")



    get_feature_selection_coeficient_names_as_formular_1se <- function(algorithm_list){
      #fit_lasso_base_cross
      #algorithm_list<-fit_lasso_base_cross
      coef_1se<- get_coef_1se_cv(algorithm_list)
      if(dim(coef_1se)[1]==1){
        print("only intercept. nothing to model")
      }else{
            coef_name_lambda_1se<-coef_1se$name[-1]
      #a<-str("")
      coefficients<-paste(coef_name_lambda_1se, collapse = " + " )

      formel<-paste("log_e.coli ~ ", coefficients)
      formel
      formula_from_selector<-formula(formel)
      }
      return(formula_from_selector)
    }
    get_feature_selection_coeficient_names_as_formular_lambda_min <- function(algorithm_list){
      #algorithm_list<-fit_lasso_base_cross
      coef_lambda_min<- get_coef_min_cv(algorithm_list)
      coef_name_lambda_min<-coef_lambda_min$name[-1]
      #a<-str("")
      coefficients<-paste(coef_name_lambda_min, collapse = " + " )
      formel<-paste("log_e.coli ~ ", coefficients)
      formula_from_selector<-formula(formel)
      return(formula_from_selector)
    }

    coef_1se_fit_lasso_base_cross<-get_coef_1se_cv (fit_lasso_base_cross)
    coef_1se_fit_lasso_base_cross_stand<-get_coef_1se_cv (fit_lasso_base_cross_stand)
    coef_lambda_min_fit_lasso_base_cross<-get_coef_min_cv (fit_lasso_base_cross)
    coef_lambda_min_fit_lasso_base_cross_stand<-get_coef_min_cv (fit_lasso_base_cross_stand)

 # add_new_formulas_to_list_if_exists <- function(coef_list){

#    if(exists("coef_1se_fit_lasso_base_cross")== TRUE){
 #     idx <- length(list_lasso)
  #    idx <- idx+1
  #    list_lasso[[idx]] <-coef_1se_fit_lasso_base_cross
  #  }
  #}

    list_lasso <- list()

    coef_1se_fit_lasso_base_cross               <-get_feature_selection_coeficient_names_as_formular_1se

    if(exists("coef_1se_fit_lasso_base_cross")== TRUE){
      idx <- length(list_lasso)
      idx <- idx+1
      list_lasso[[idx]] <-coef_1se_fit_lasso_base_cross
    }
    coef_1se_fit_lasso_base_cross_stand         <-get_feature_selection_coeficient_names_as_formular_1se
    if(exists("coef_1se_fit_lasso_base_cross_stand")== TRUE){
      idx <- length(list_lasso)
      idx <- idx+1
      list_lasso[[idx]] <-coef_1se_fit_lasso_base_cross_stand
    }

    coef_lambda_min_fit_lasso_base_cross        <-get_feature_selection_coeficient_names_as_formular_lam
    if(exists("coef_lambda_min_fit_lasso_base_cross")== TRUE){
      idx <- length(list_lasso)
      idx <- idx+1
      list_lasso[[idx]] <-coef_lambda_min_fit_lasso_base_cross
    }

    coef_lambda_min_fit_lasso_base_cross_stand  <-get_feature_selection_coeficient_names_as_formular_lam
    if(exists("coef_lambda_min_fit_lasso_base_cross_stand")== TRUE){
      idx <- length(list_lasso)
      idx <- idx+1
      list_lasso[[idx]] <-coef_lambda_min_fit_lasso_base_cross_stand
    }
```

```
    #check if all 4 coefficients exist and remove intercepts
    idx <-0
    for(element in list_lasso){
      idx<-idx+1
      if(typeof(element)!="language"){
        list_lasso <- list_lasso[-idx]
        print("f")
      }
    }
    list_lasso
    print(paste(length(list_lasso)," new models added"))
    model_lsit<-list()
    list_lasso
    #builded linear model
    heiko_lm_1<-lm(list_lasso[[1]], data = data)
    heiko_lm_2<-lm(list_lasso[[2]],data=data)
    heiko_lm_3<-lm(list_lasso[[3]],data=data)
    heiko_lm_4<-lm(list_lasso[[4]],data=data)

    list_heiko_lm <- list()
    list_heiko_lm[[1]]<- heiko_lm_1
    list_heiko_lm[[2]]<- heiko_lm_2
    list_heiko_lm[[3]]<- heiko_lm_3
    list_heiko_lm[[4]]<- heiko_lm_4
    #for(form in list_lasso){

     # heiko_lm <- lm(form, data = data)
    #  heiko_lm<-list(heiko_lm)
     # append(heiko_lm,model_lsit)
  #  }
  #heiko_lm<- lm(formula_heiko_1, data = data)

#nicht mehr benötigt
  #### Anwenden der Hauptfunktion ###################

  stepwise <- function (river, pattern, data, null, full ){
 # Definition maximum number of steps

    nsteps <- 5 #ifelse(round(nrow(data)/10) < 10, round(nrow(data)/10), 5 )

    selection <- list()

    fmla <- list()



    # Creating list of candidate models with 1 ...n predictors
    #split up this piece in stpe and new algorithms/formulars

    for(i in 1: nsteps){
```

```
    selection[[i]] <- step(null, data = data,

                           direction = "forward",

                           list(lower=null, upper=full), steps = i, trace=FALSE)


    fmla[[i]] <- as.list(selection[[i]]$call)$formula



   }

   #heiko_add_formular to fmla list function function
   #selection[[6]] <- heiko_lm
   #fmla[[6]] <- as.list(selection[[6]]$call)$formula
   step_returns <- list(fmla, selection)
   return(step_returns)

  }





  # order of pattern, q_old and q_new is important!

  #fb <- stepwise(river = river, pattern = "(i_mean|q_mean_mean|r_mean_mean|ka_mean_mean)", data,null, 
  step_returns <- stepwise(river = river, pattern = "(i_mean|q_mean_mean|r_mean_mean|ka_mean_mean)", da
  fmla <- step_returns[[1]]
  selection <- step_returns[[2]]






#adding new linear models, featureselection with lasso/elnet
#selection<-append(selection, list(heiko_lm_1,heiko_lm_2,heiko_lm_3,heiko_lm_4))
  selection<-append(selection, list_heiko_lm)
  fb<- selection
#fb[6] <- list(heiko_lm)

  #selection[6] <- list(heiko_lm)
  #selection
  #fb

  fmla_heiko_1 <-eval(heiko_lm_1$call$formula)
  fmla_heiko_2 <-eval(heiko_lm_2$call$formula)
  fmla_heiko_3 <-eval(heiko_lm_3$call$formula)
  fmla_heiko_4 <-eval(heiko_lm_4$call$formula)

  fmla_heiko <- list()
```

```r
fmla_heiko[[1]]<- fmla_heiko_1
fmla_heiko[[2]]<- fmla_heiko_2
fmla_heiko[[3]]<- fmla_heiko_3
fmla_heiko[[4]]<- fmla_heiko_4
# as.list(selection[[6]]$call)$formula

fmla<-append(fmla, fmla_heiko)
#fmla
if(class(fmla[[length(fmla)]]) !="formula"){
  print("new element is no formula!!")
}


#add my models here


#q_old = "q_cochem",

#q_new = "q_cochem_abs_1")






names(fb) <- sprintf(paste0(river,"model_%02d"), seq_along(1:length(fb)))



################ Validation ########################



# calculate statistical tests for residuals: Normality and s2 = const

# shapiro-wilk test and breusch-pagan test

get_stat_tests <- function(model) {
  c(N = shapiro.test(model$residuals)$p.value, lmtest::bptest(model)$p.value,
  R2 = summary(model)[["adj.r.squared"]], n_obs = length(model$residuals))

}



# Eliminieren von modelled die doppelt vorkommen, da forward selection früher

#fertig als n steps

#heiko add fb beforehand to this
#fb
unique_index <- length(unique(fb))
fb <- fb[1:unique_index]
```

```
  # testing for classical statistical model assumtions, normality of residuals and

  # heteroskelasdicity
  river_stat_tests <- sapply(fb, get_stat_tests)%>%
    t() %>%
    dplyr::as_tibble(rownames = "model")  %>%
    dplyr::bind_rows(.id = "river") %>%
    dplyr::mutate(stat_correct = N > .05 & BP > .05)



  # creating list of independent training rows
  #-test/train split

  #weirde zeile, setze alle stat tests auf 0
  river_stat_tests$in95 <- river_stat_tests$below95 <-river_stat_tests$below90 <- river_stat_tests$in50


  train_rows <- caret::createFolds(1:nrow(fb[[paste0(river, "model_01")]]$model),

                                   k = 5, list = T, returnTrain = T)



  if(class(fmla[[length(fmla)]]) !="formula"){
    print("new element is no formula!!")
  }

  test_beta <- function(true, false, percentile){
    if( pbeta(q = percentile, shape1 = true + 1, shape2 = false + 1) > 0.05){
      TRUE}
    else{FALSE}

  }


  names(fmla) <- sprintf(paste0(river,"model_%02d"), seq_along(1:length(fb)))




  counter<-0

  #fb<-fb[-6]
  #names(fb)
}

  for(i in names(fb)){
  counter<- counter+1
       #i="havelmodel_01"
    for(j in 1:5){
      counter <- counter+1
       #   j=1




        training <- as.data.frame(fb[[i]]$model)[c(train_rows[[j]]),]
        #training <- as.data.frame(fb[[6]]$model)[c(train_rows[[1]]),]
        test <- as.data.frame(fb[[i]]$model)[-c(train_rows[[j]]),]
        #test <- as.data.frame(fb[[6]]$model)[-c(train_rows[[1]]),]
        #formel<-formula(formula_heiko_1)


        #fmla[6]<- list(formel)



        fit <- rstanarm::stan_glm(fmla[[i]], data = training, refresh=0) #fitting #suppress print out w
        #fit <- rstanarm::stan_glm(fmla[[1]], data = training) #fitting


        df <- apply(rstanarm::posterior_predict(fit, newdata = test), 2, quantile, #predicting

                    probs = c(0.025, 0.25, 0.75, 0.9, 0.95, 0.975)) %>% t() %>% as.data.frame() %>%

          dplyr::mutate(log_e.coli = test$log_e.coli, #evaluating ther model has to be classified corre
                        #--> here 5 different splits, if all validations correct than everywhere ==5

                        below95 = log_e.coli < `95%`,

                        below90 = log_e.coli < `90%`,

                        within95 = log_e.coli < `97.5%`& log_e.coli > `2.5%`,

                        within50 = log_e.coli < `75%`& log_e.coli > `25%`,

          )

        #validation step if allpercentile categories are set to 1

        river_stat_tests$in95[river_stat_tests$model == i] <-

          river_stat_tests$in95[river_stat_tests$model == i] +

          test_beta(true = sum(df$within95), false = sum(!df$within95), percentile = .95 )

        river_stat_tests$below95[river_stat_tests$model == i] <-

          river_stat_tests$below95[river_stat_tests$model == i] +

          test_beta(true = sum(df$below95), false = sum(!df$below95), percentile = .95 )

        river_stat_tests$below90[river_stat_tests$model == i] <-
```

```
          river_stat_tests$below90[river_stat_tests$model == i] +

          test_beta(true = sum(df$below90), false = sum(!df$below90), percentile = .90 )

        river_stat_tests$in50[river_stat_tests$model == i] <-

          river_stat_tests$in50[river_stat_tests$model == i] +

          test_beta(true = sum(df$within50), false = sum(!df$within50), .5)



      }

  }

#fmla

}

}
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: Matrix

## Loaded glmnet 4.0

##
## Attaching package: 'purrr'

## The following object is masked from 'package:magrittr':
##
##     set_names

## -- Attaching packages --------------------------------------------------------------------------------

## v ggplot2 3.3.1     v readr   1.3.1
## v tibble  3.0.1     v stringr 1.4.0
## v tidyr   1.1.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------------------------------------------------------
## x tidyr::expand()    masks Matrix::expand()
## x tidyr::extract()   masks magrittr::extract()
## x dplyr::filter()    masks stats::filter()
## x dplyr::lag()       masks stats::lag()
## x tidyr::pack()      masks Matrix::pack()
## x purrr::set_names() masks magrittr::set_names()
## x tidyr::unpack()    masks Matrix::unpack()
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
##
## Attaching package: 'kwb.flusshygiene'
## The following objects are masked from 'package:fhpredict':
##
##     build_model, predict_quality
## [1] "4  new models added"
```

```
sorted_modellist <- river_stat_tests %>%

  filter( below95 == 5 & below90 == 5& in95) %>%

  dplyr::arrange(desc(in50), desc(R2))

#river_stat_tests
#sorted_modellist

best_valid_model_stats <- sorted_modellist[1,]

best_valid_model <- fb[[best_valid_model_stats$model]]

coef(best_valid_model)
```

```
##                      (Intercept)                    q_mean_mean_45
##                     1.494515e+00                      5.902623e-04
## ka_mean_mean_12:ka_mean_mean_123  ka_mean_mean_12:ka_mean_mean_34
##                    -5.067832e-05                     -8.297288e-05
##    q_mean_mean_12:r_mean_mean_12    q_mean_mean_45:r_mean_mean_12
##                     1.491028e-02                     -6.673480e-03
##  q_mean_mean_45:r_mean_mean_1234    q_mean_mean_45:r_mean_mean_45
##                     5.781464e-03                      2.543349e-03
```

```
#refit best model
stanfit <- rstanarm::stan_glm(fmla[[best_valid_model_stats$model]],

                              data = best_valid_model$model)
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.21 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.303884 seconds (Warm-up)
## Chain 1:                0.293915 seconds (Sampling)
## Chain 1:                0.597799 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.293798 seconds (Warm-up)
## Chain 2:                0.332186 seconds (Sampling)
## Chain 2:                0.625984 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.295063 seconds (Warm-up)
## Chain 3:                0.340178 seconds (Sampling)
## Chain 3:                0.635241 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.1e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.300569 seconds (Warm-up)
## Chain 4:                0.318844 seconds (Sampling)
## Chain 4:                0.619413 seconds (Total)
## Chain 4:
```

```r
brmsfit <- brms::brm(fmla[[best_valid_model_stats$model]],

                     data = best_valid_model$model, iter = 10000)
```

```
## Compiling the C++ model
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -mmacosx-version-min=10.13 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                 ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/4.0/Resources/library/RcppEigen/include/Eigen/Core:96:10: f
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
```

```
## Start sampling
```

```
##
## SAMPLING FOR MODEL 'a978e1c2b077df5ea455a5c1a23de831' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.6e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.26 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 1: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 1: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 1: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 1: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 1: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 1: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 1: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 1: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 1: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 1: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 1: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 11.2136 seconds (Warm-up)
## Chain 1:                8.01748 seconds (Sampling)
## Chain 1:                19.2311 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'a978e1c2b077df5ea455a5c1a23de831' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 2: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 2: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 2: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 2: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 2: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 2: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 2: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 2: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 2: Iteration: 8000 / 10000 [ 80%]  (Sampling)
```

```
## Chain 2: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 2: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 10.7723 seconds (Warm-up)
## Chain 2:                7.23688 seconds (Sampling)
## Chain 2:                18.0092 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'a978e1c2b077df5ea455a5c1a23de831' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 7e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 3: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 3: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 3: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 3: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 3: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 3: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 3: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 3: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 3: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 3: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 3: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 11.1286 seconds (Warm-up)
## Chain 3:                8.38621 seconds (Sampling)
## Chain 3:                19.5148 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'a978e1c2b077df5ea455a5c1a23de831' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 9e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 10000 [  0%]  (Warmup)
## Chain 4: Iteration: 1000 / 10000 [ 10%]  (Warmup)
## Chain 4: Iteration: 2000 / 10000 [ 20%]  (Warmup)
## Chain 4: Iteration: 3000 / 10000 [ 30%]  (Warmup)
## Chain 4: Iteration: 4000 / 10000 [ 40%]  (Warmup)
## Chain 4: Iteration: 5000 / 10000 [ 50%]  (Warmup)
## Chain 4: Iteration: 5001 / 10000 [ 50%]  (Sampling)
## Chain 4: Iteration: 6000 / 10000 [ 60%]  (Sampling)
## Chain 4: Iteration: 7000 / 10000 [ 70%]  (Sampling)
## Chain 4: Iteration: 8000 / 10000 [ 80%]  (Sampling)
## Chain 4: Iteration: 9000 / 10000 [ 90%]  (Sampling)
## Chain 4: Iteration: 10000 / 10000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 10.5697 seconds (Warm-up)
## Chain 4:                7.61655 seconds (Sampling)
## Chain 4:                18.1862 seconds (Total)
## Chain 4:
```

```
  #return(list(sorted_modellist = sorted_modellist,

   #           best_model = best_valid_model,

    #          stanfit = stanfit,

     #         brmsfit = brmsfit))
```

The used data is from river 'Havel': "/Data_preprocess/Daten_TestPackage_Berlin/Havel/DATA_preprocessed_csv"

It is taken from the data folder provided by Wolfgang and has the following structure

```
glimpse(data)
```

```
## Rows: 36
## Columns: 31
## $ log_e.coli         <dbl> 2.100371, 2.428135, 2.252853, 2.838849, 1.662758...
## $ ka_mean_mean_12    <dbl> 38.0145, 69.7390, 0.0000, 109.7215, 32.4950, 36....
## $ ka_mean_mean_123   <dbl> 25.34300000, 108.61533333, 0.00000000, 85.481000...
## $ ka_mean_mean_1234  <dbl> 19.00725, 86.94825, 0.00000, 97.09825, 19.73725,...
## $ ka_mean_mean_12345 <dbl> 15.2058, 79.3838, 13.3450, 84.9316, 17.4996, 18....
## $ ka_mean_mean_2345  <dbl> 17.36950, 70.51750, 16.68125, 74.91325, 5.62700,...
## $ ka_mean_mean_345   <dbl> 0.000000, 85.813667, 22.241667, 68.405000, 7.502...
## $ ka_mean_mean_45    <dbl> 0.0000, 35.5365, 33.3625, 84.1075, 11.2540, 9.56...
## $ ka_mean_mean_234   <dbl> 23.1593333, 77.6480000, 0.0000000, 87.7960000, 4...
## $ ka_mean_mean_23    <dbl> 34.7390, 105.4985, 0.0000, 65.7190, 0.0000, 28.8...
## $ ka_mean_mean_34    <dbl> 0.0000, 104.1575, 0.0000, 84.4750, 6.9795, 5.353...
## $ q_mean_mean_12     <dbl> 61.65, 75.15, 68.30, 80.30, 66.95, 25.85, 37.25,...
## $ q_mean_mean_123    <dbl> 61.43333, 75.16667, 70.56667, 78.16667, 65.93333...
## $ q_mean_mean_1234   <dbl> 62.175, 68.925, 72.650, 78.325, 65.125, 27.325, ...
## $ q_mean_mean_12345  <dbl> 61.48, 64.44, 74.76, 76.68, 64.82, 25.50, 27.00,...
## $ q_mean_mean_2345   <dbl> 61.325, 61.900, 76.525, 74.650, 63.975, 25.625, ...
## $ q_mean_mean_345    <dbl> 61.36667, 57.30000, 79.06667, 74.26667, 63.40000...
## $ q_mean_mean_45     <dbl> 61.55, 48.35, 81.05, 74.45, 63.15, 25.35, 16.85,...
## $ q_mean_mean_234    <dbl> 62.20000, 67.03333, 74.30000, 76.16667, 64.10000...
## $ q_mean_mean_23     <dbl> 61.10, 75.45, 72.00, 74.85, 64.80, 25.90, 29.70,...
## $ q_mean_mean_34     <dbl> 62.70, 62.70, 77.00, 76.35, 63.30, 28.80, 23.05,...
## $ r_mean_mean_12     <dbl> 1.15234521, 1.34171239, 0.04126964, 1.93422257, ...
## $ r_mean_mean_123    <dbl> 0.82315166, 1.95004080, 0.06128739, 1.41172257, ...
## $ r_mean_mean_1234   <dbl> 0.6435257, 2.1261437, 0.3237925, 1.6186162, 0.59...
## $ r_mean_mean_12345  <dbl> 0.5262331, 1.7113038, 0.5347389, 1.4190294, 0.59...
## $ r_mean_mean_2345   <dbl> 0.2533034, 1.6019403, 0.6573695, 1.1881228, 0.41...
## $ r_mean_mean_345    <dbl> 0.1088251, 1.9576981, 0.8637185, 1.0755673, 0.54...
## $ r_mean_mean_45     <dbl> 0.08085530, 1.35319831, 1.24491623, 1.42998969, ...
## $ r_mean_mean_234    <dbl> 0.3187169, 2.1186057, 0.4169844, 1.3772697, 0.37...
## $ r_mean_mean_23     <dbl> 0.42575152, 1.85068227, 0.06982273, 0.94625589, ...
## $ r_mean_mean_34     <dbl> 0.1347061, 2.9105750, 0.6063153, 1.3030099, 0.54...
```

The prebuild list of 31 averaged variables/features are increased through interactions between each other. This is done with by building all interaction formulars with "formula(paste(part1," ~ (.)^2")" So every feature besides e.coli will be multiplied with every other feature which leads to 466 features from which the Lasso-algorithms are selecting the most important.

here is an insight of the last 50 formulas

```
tail(get_formula_variable_names(form,data), 50)
```

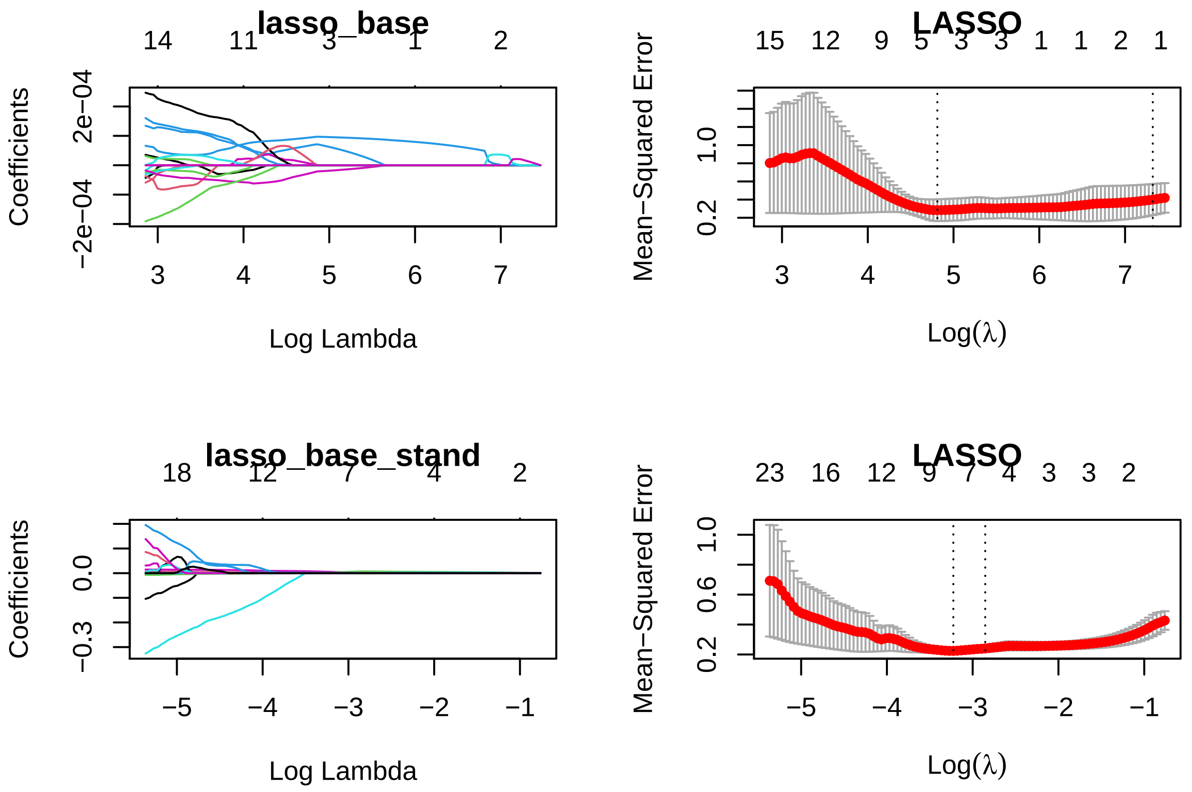
```
##  [1] "q_mean_mean_34:r_mean_mean_345"     "q_mean_mean_34:r_mean_mean_45"
##  [3] "q_mean_mean_34:r_mean_mean_234"     "q_mean_mean_34:r_mean_mean_23"
##  [5] "q_mean_mean_34:r_mean_mean_34"      "r_mean_mean_12:r_mean_mean_123"
##  [7] "r_mean_mean_12:r_mean_mean_1234"    "r_mean_mean_12:r_mean_mean_12345"
##  [9] "r_mean_mean_12:r_mean_mean_2345"    "r_mean_mean_12:r_mean_mean_345"
## [11] "r_mean_mean_12:r_mean_mean_45"      "r_mean_mean_12:r_mean_mean_234"
## [13] "r_mean_mean_12:r_mean_mean_23"      "r_mean_mean_12:r_mean_mean_34"
## [15] "r_mean_mean_123:r_mean_mean_1234"   "r_mean_mean_123:r_mean_mean_12345"
## [17] "r_mean_mean_123:r_mean_mean_2345"   "r_mean_mean_123:r_mean_mean_345"
## [19] "r_mean_mean_123:r_mean_mean_45"     "r_mean_mean_123:r_mean_mean_234"
## [21] "r_mean_mean_123:r_mean_mean_23"     "r_mean_mean_123:r_mean_mean_34"
## [23] "r_mean_mean_1234:r_mean_mean_12345" "r_mean_mean_1234:r_mean_mean_2345"
## [25] "r_mean_mean_1234:r_mean_mean_345"   "r_mean_mean_1234:r_mean_mean_45"
## [27] "r_mean_mean_1234:r_mean_mean_234"   "r_mean_mean_1234:r_mean_mean_23"
## [29] "r_mean_mean_1234:r_mean_mean_34"    "r_mean_mean_12345:r_mean_mean_2345"
## [31] "r_mean_mean_12345:r_mean_mean_345"  "r_mean_mean_12345:r_mean_mean_45"
## [33] "r_mean_mean_12345:r_mean_mean_234"  "r_mean_mean_12345:r_mean_mean_23"
## [35] "r_mean_mean_12345:r_mean_mean_34"   "r_mean_mean_2345:r_mean_mean_345"
## [37] "r_mean_mean_2345:r_mean_mean_45"    "r_mean_mean_2345:r_mean_mean_234"
## [39] "r_mean_mean_2345:r_mean_mean_23"    "r_mean_mean_2345:r_mean_mean_34"
## [41] "r_mean_mean_345:r_mean_mean_45"     "r_mean_mean_345:r_mean_mean_234"
## [43] "r_mean_mean_345:r_mean_mean_23"     "r_mean_mean_345:r_mean_mean_34"
## [45] "r_mean_mean_45:r_mean_mean_234"     "r_mean_mean_45:r_mean_mean_23"
## [47] "r_mean_mean_45:r_mean_mean_34"      "r_mean_mean_234:r_mean_mean_23"
## [49] "r_mean_mean_234:r_mean_mean_34"     "r_mean_mean_23:r_mean_mean_34"
```

The features are getting selected through the lasso algorithm with cross validation without (1.row) or with (2.row) internal standardization of the features

The following 4 graphs show the the number of selected features (upper sclae) dependend on log lambda (lower scale) The left dotted line shows the lambda with the minimal MSE - identified throug cross-validation The right dotted line shows the lambda 1 standard deviation away from the with the minimal MSE - identified throug cross-validation (shall give a better generalization than minimum mse)

```
par(mfrow=c(2,2))
   plot(fit_lasso_base, xvar="lambda", label = T, main = "lasso_base")
   plot(fit_lasso_base_cross,main="LASSO")

   plot(fit_lasso_base_stand, xvar="lambda", label = T, main = "lasso_base_stand")
   plot(fit_lasso_base_cross_stand,main="LASSO")
```



lambda 1se for lasso base cross

```
get_coef_1se_cv(fit_lasso_base_cross)$name
```

```
## [1] "(Intercept)"                    "q_mean_mean_123:q_mean_mean_23"
```

lambda 1se for lasso base cross standardized

```
get_coef_1se_cv(fit_lasso_base_cross_stand)$name
```

```
## [1] "(Intercept)"                        "q_mean_mean_2345"
## [3] "ka_mean_mean_12:ka_mean_mean_123" "ka_mean_mean_12:ka_mean_mean_34"
## [5] "q_mean_mean_12:r_mean_mean_12"    "q_mean_mean_45:r_mean_mean_12"
## [7] "q_mean_mean_45:r_mean_mean_12345" "q_mean_mean_45:r_mean_mean_45"
```

9 different models were build in the very first code snippet: - 5 from stepwise regression, with 1-5 explanatory variables (choosen with Information Criterion) - 4 Lasso based models:

from the two cross validated lasso fits: fit_lasso_base_cross and fit_lasso_base_cross_stand (with internal standardization of features) I used lambda min and lambda-1se to get 4 lasso based models with the following modelnumbers: 06: fit_lasso_base_cross lambda-1se 07: fit_lasso_base_cross_stand lambda-1se 08: fit_lasso_base_cross lambda-min 09: fit_lasso_base_cross_stand lambda-min

The 4 Lasso based models were than used in the posterior function for validation. If in50, below90, below95, and in 95 are all == 5 –> the model is validated through the validation method from wolfgang.

The best Model was chosen by further checking statistical correctness and sorting R2 in descending order.

```
sorted_modellist
```

```
## # A tibble: 9 x 11
##   river model        N     BP    R2 n_obs stat_correct  in50 below90 below95
##   <chr> <chr>    <dbl>  <dbl> <dbl> <dbl> <lgl>        <dbl>   <dbl>   <dbl>
## 1 1     have~ 5.69e-1 0.155  0.697    36 TRUE             5       5       5
## 2 1     have~ 7.56e-1 0.0807 0.681    36 TRUE              5       5       5
## 3 1     have~ 8.32e-2 0.391  0.654    36 TRUE              5       5       5
## 4 1     have~ 2.22e-2 0.365  0.636    36 FALSE             5       5       5
## 5 1     have~ 5.06e-4 0.514  0.591    36 FALSE             5       5       5
## 6 1     have~ 4.95e-2 0.0225 0.560    36 FALSE             5       5       5
## 7 1     have~ 1.60e-2 0.0524 0.423    36 FALSE             5       5       5
## 8 1     have~ 8.46e-2 0.153  0.503    36 TRUE              4       5       5
## 9 1     have~ 3.68e-2 0.0236 0.474    36 FALSE             4       5       5
## # ... with 1 more variable: in95 <dbl>
```

Whats interesting is, that only the standardized models (07 & 09) were able to beat the stepwise builded models with this evaluation method.

Best Model here is havelmodel_09 with the following coefficients:

```r
names(coef(best_valid_model))
```

```
## [1] "(Intercept)"                      "q_mean_mean_45"
## [3] "ka_mean_mean_12:ka_mean_mean_123" "ka_mean_mean_12:ka_mean_mean_34"
## [5] "q_mean_mean_12:r_mean_mean_12"    "q_mean_mean_45:r_mean_mean_12"
## [7] "q_mean_mean_45:r_mean_mean_1234"  "q_mean_mean_45:r_mean_mean_45"
```