Supplement of Earth Syst. Sci. Data, 13, 5847–5866, 2021
https://doi.org/10.5194/essd-13-5847-2021-supplement

Open Access Earth System Science Data

Supplement of

Modelling seabed sediment physical properties and organic matter content in the Firth of Clyde

Matthew C. Pace et al.

Correspondence to: Matthew C. Pace (matthew.pace@glasgow.ac.uk)

Contents

1 Summary

This is a markdown document to report and document the methods used to model the seabed grain size and geotechnical properties and sedimentary organic carbon and nitrogen content.

N.B. - Values may slightly differ from those presented in paper as random number generator seed is not fixed.

2 Load packages and define file directories

```
## [1] "C:/Users/matt8/Documents/R/local_lib_clydemapping"
```

```
## [2] "C:/Users/matt8/Documents/R/win-library/4.0"
## [3] "C:/Program Files/R/R-4.0.3/library"
```

```
## load required packages
suppressWarnings({library(data.table)
library(tidyverse)
library(ranger)
library(pROC)
library(ggvoronoi)
library(maptools)
library(rgdal)
library(akima)
library(geosphere)
library(gridExtra)
library(readxl)
library(latticeExtra)
library(deldir)
library(mgcv)
library(geoR)
library(SDMTools)
library(caret)})

## Print R and package versions
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19042)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.1252
## [2] LC_CTYPE=English_United Kingdom.1252
## [3] LC_MONETARY=English_United Kingdom.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] caret_6.0-86        SDMTools_1.1-221.1  geoR_1.8-1
##  [4] mgcv_1.8-33         nlme_3.1-149        deldir_0.1-16
##  [7] latticeExtra_0.6-29 lattice_0.20-41     readxl_1.3.1
## [10] gridExtra_2.3       geosphere_1.5-10    akima_0.6-2.1
## [13] rgdal_1.5-23        maptools_1.0-2      sp_1.4-5
## [16] ggvoronoi_0.8.4     pROC_1.17.0.1       ranger_0.11.2
## [19] forcats_0.5.1       stringr_1.4.0       dplyr_1.0.7
## [22] purrr_0.3.4         readr_2.0.1         tidyr_1.1.3
## [25] tibble_3.1.4        ggplot2_3.3.5       tidyverse_1.3.1
## [28] data.table_1.14.0
##
## loaded via a namespace (and not attached):
##  [1] RandomFieldsUtils_0.5.3 fs_1.5.0                lubridate_1.7.10
```

```
##  [4] RColorBrewer_1.1-2      httr_1.4.2              tools_4.0.3
##  [7] backports_1.2.1         utf8_1.2.2              R6_2.5.1
## [10] rpart_4.1-15            rgeos_0.5-5             DBI_1.1.1
## [13] colorspace_2.0-2        nnet_7.3-14             raster_3.4-5
## [16] withr_2.4.2             tidyselect_1.1.1        splancs_2.01-42
## [19] compiler_4.0.3          RandomFields_3.3.8      cli_3.0.1
## [22] rvest_1.0.1             xml2_1.3.2              bookdown_0.24
## [25] scales_1.1.1            digest_0.6.27           foreign_0.8-80
## [28] R.utils_2.10.1          rmarkdown_2.10          jpeg_0.1-9
## [31] pkgconfig_2.0.3         htmltools_0.5.1.1       dbplyr_2.1.1
## [34] rlang_0.4.10            rstudioapi_0.13         generics_0.1.0
## [37] jsonlite_1.7.2          ModelMetrics_1.2.2.2    R.oo_1.24.0
## [40] magrittr_2.0.1          Matrix_1.2-18           Rcpp_1.0.7
## [43] munsell_0.5.0           fansi_0.5.0             R.methodsS3_1.8.1
## [46] lifecycle_1.0.0         stringi_1.5.3           yaml_2.2.1
## [49] MASS_7.3-53             recipes_0.1.15          plyr_1.8.6
## [52] grid_4.0.3              crayon_1.4.1            haven_2.4.3
## [55] splines_4.0.3           hms_1.1.0               knitr_1.31
## [58] pillar_1.6.2            tcltk_4.0.3             stats4_4.0.3
## [61] reshape2_1.4.4          codetools_0.2-16        reprex_2.0.1
## [64] glue_1.4.2              evaluate_0.14           modelr_0.1.8
## [67] foreach_1.5.1           png_0.1-7               vctrs_0.3.8
## [70] tzdb_0.1.2              cellranger_1.1.0        gtable_0.3.0
## [73] assertthat_0.2.1        gower_0.2.2             xfun_0.25
## [76] prodlim_2019.11.13      broom_0.7.9             survival_3.2-7
## [79] class_7.3-17            timeDate_3043.102       iterators_1.0.13
## [82] lava_1.6.9              ellipsis_0.3.2          ipred_0.9-10
```

```
filepath <- "data/"

# Import polygon of Clyde coastline
Shape_coast <- rgdal::readOGR(paste(filepath, "high_water_polygon_Clyde.shp",sep = "/"))
```

```
## Warning in OGRSpatialRef(dsn, layer, morphFromESRI = morphFromESRI, dumpSRS =
## dumpSRS, : Discarded datum D_unknown in Proj4 definition: +proj=tmerc +lat_0=49
## +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000 +ellps=airy +units=m +no_defs
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\matt8\Dropbox (Glasgow Uni)\PhD\Manuscripts\Modelling Seabed Clyde\supplementary_ma
## with 369 features
## It has 2 fields
```

```
Shape_coast <- spTransform(Shape_coast, CRS("+proj=longlat +datum=WGS84"))
Shape_coast_f <- fortify(Shape_coast)
```

3 Gridding Environmental Covariates

The first step of the process is the processing of candidate explanatory environmental variables derived from bathymetry data and tidal current data from a high-resolution hydrodynamic model of the Clyde - see the main data paper for detailed information.

This is a computationally expensive process and the code is shown here for illustrative purposes and is not executed within this document.

```
# ------------------------------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT. This takes ~ 5.5 hours to run.
```

```
# ------------------------------------------------------------------------------------------#

# ----------------------------------#
# Seabed depth
# ----------------------------------#

## Load unstructured bathymetry grid
FVCOM_bathymetry <- fread("FVCOM_data/FVCOM Bathymetry.csv")

## Latitude and Longitude
lat <- fread("Clyde_Lat.csv")
lon <- fread("Clyde_Lon.csv")

Clyde_LonLat <- data.frame(Lat = unlist(lat[1,]), Lon = unlist(lon[1,]))

# ----------------------------------------#
# Topographic roughness
# ----------------------------------------#
#
# We need to first interpolate seabed depth to a regular grid for the calculation
# of roughness.

## Use high-resolution regular grid as a reference for spatial positions to interpolate to
EMODnet <- raster("emodnet-mean_replicate.asc")
EMODnet_df <- as.data.frame(EMODnet,xy = TRUE)
names(EMODnet_df) <- c("x","y","Depth")

## Check resolution of reference grid
EMODnet_df$x[1] - EMODnet_df$x[2]   ## 0.002083333
EMODnet_df$y[1] - EMODnet_df$y[737] ## 0.002083333

## Interpolate using the akima package
FVCOM_hires <- interpp(x = FVCOM_bathymetry$Lon,
                       y = FVCOM_bathymetry$Lat,
                       z = FVCOM_bathymetry$Depth,
                       xo = EMODnet_df$x,
                       yo = EMODnet_df$y) %>%
  as.data.frame() %>%
  rename(Lon = x,
         Lat = y,
         Depth = z)

## remove terrestrial areas
FVCOM_hires_spatial<- SpatialPointsDataFrame(coords = FVCOM_hires[,1:2],
                                             data = FVCOM_hires,
                                             proj4string = CRS("+proj=longlat +ellps=WGS84"))

FVCOM_hires_spatial$Terrain <- over(x = FVCOM_hires_spatial,
                                    y = Shape_coast["area"],
                                    returnList = FALSE)

FVCOM_hires_spatial$Terrain <- FVCOM_hires_spatial$Terrain$area
FVCOM_hires_spatial <- FVCOM_hires_spatial %>% as.data.frame() %>%
```

```
  mutate(Depth = if_else(is.na(Terrain), Depth, NA_real_)) %>%
  select(Lon, Lat, Depth)

## Convert to raster and calculate slope
FVCOM_hires_raster <- rasterFromXYZ(FVCOM_hires_spatial, crs = CRS("+proj=longlat +ellps=WGS84"))

# Calculated as the difference between bathymetry at a specific point and mean
# bathymetry within a spatial window (25 cells). This is calculated for each
# cell in the grid. Then standard deviation of residual topography calculated
# via sigma = 1/25*sqrt(sum(xi - xm)^2)

## Loop to calculate mean
winwidth <- 5
ncell <- winwidth^2
res <- 0.00208
n <- nrow(FVCOM_hires)
pb <- txtProgressBar(style = 3)
for(i in 1:n) {

  DepthVector <- FVCOM_hires$Depth[FVCOM_hires$Lat <= FVCOM_hires$Lat[i] + (res * winwidth/2) &
                                     FVCOM_hires$Lat >= FVCOM_hires$Lat[i] - (res * winwidth/2) &
                                     FVCOM_hires$Lon <= FVCOM_hires$Lon[i] + (res * winwidth/2) &
                                     FVCOM_hires$Lon >= FVCOM_hires$Lon[i] - (res * winwidth/2)]

  FVCOM_hires$meanDepth[i] <- mean(DepthVector, na.rm = TRUE)
  FVCOM_hires$ncells[i] <- sum(!is.na(DepthVector))

  setTxtProgressBar(pb, i/n)
}

## Calculate standard deviation
FVCOM_hires <- FVCOM_hires %>%
  group_by(Lon, Lat) %>%
  mutate(Roughness = (1/ncells)*sqrt(sum((Depth - meanDepth)^2)))

## Interpolate topographic roughness to unstructured grid
FVCOM_bathymetry$Roughness <- interpp(x = FVCOM_hires$Lon,
                                      y = FVCOM_hires$Lat,
                                      z = FVCOM_hires$Roughness,
                                      xo = FVCOM_bathymetry$x,
                                      yo = FVCOM_bathymetry$y)$z


# ----------------------------------------#
# Slope and standard deviation of the slope
# ----------------------------------------#

FVCOM_hires_Slope <- slope(FVCOM_hires_raster, latlon = TRUE)
FVCOM_hires_Slope <- as.data.frame(FVCOM_hires_Slope, xy = TRUE)
names(FVCOM_hires_Slope) <- c("Lon", "Lat", "Slope")

# Calculated standard deviation of the slope in a similar way as the residual
# topography using a 25 cell moving window.
```

```
## Calculate standard deviation of the slope
winwidth <- 5
ncell <- winwidth^2
res <- 0.00208
n <- nrow(FVCOM_hires_Slope)
pb <- txtProgressBar(style = 3)
for(i in 1:n) {

  SlopeVector <- FVCOM_hires_Slope$Slope[
    FVCOM_hires_Slope$Lat <= FVCOM_hires_Slope$Lat[i] + (res * winwidth/2) &
    FVCOM_hires_Slope$Lat >= FVCOM_hires_Slope$Lat[i] - (res * winwidth/2) &
    FVCOM_hires_Slope$Lon <= FVCOM_hires_Slope$Lon[i] + (res * winwidth/2) &
    FVCOM_hires_Slope$Lon >= FVCOM_hires_Slope$Lon[i] - (res * winwidth/2)]

  FVCOM_hires_Slope$meanSlope[i] <- mean(SlopeVector, na.rm = TRUE)
  FVCOM_hires_Slope$ncells[i] <- sum(!is.na(SlopeVector))

  setTxtProgressBar(pb, i/n)
}

## Calculate standard deviation
FVCOM_hires_Slope <- FVCOM_hires_Slope %>%
  group_by(Lon, Lat) %>%
  mutate(SDSlope = (1/ncells)*sqrt(sum((Slope - meanSlope)^2)))

## Interpolate slope to unstructured grid
FVCOM_bathymetry$Slope <- interpp(x = FVCOM_hires_Slope$Lon,
                                  y = FVCOM_hires_Slope$Lat,
                                  z = FVCOM_hires_Slope$Slope,
                                  xo = FVCOM_bathymetry$x,
                                  yo = FVCOM_bathymetry$y)$z

## Interpolate standard deviation of slope to structured grid
FVCOM_bathymetry$SDSlope <- interpp(x = FVCOM_hires$Lon,
                                    y = FVCOM_hires$Lat,
                                    z = FVCOM_hires$SDSlope,
                                    xo = FVCOM_bathymetry$x,
                                    yo = FVCOM_bathymetry$y)$z

# ----------------------------------------#
# Minimum, mean and maximum current speed
# ----------------------------------------#

## Set directory into which I will read each line of the data table
setwd("FVCOM_data/DepthAveragedCurrentVelocity_fst")

## N rows in dataframe
n <- 39449

## read in data and calculate dimensionless current velocity (2005 data)
pb <- txtProgressBar(style = 3)
for (i in 1:n){
```

```
  x_velocity <- fread(input = "FVCOM_data/Clyde_Vertically_Averaged_x_velocity_2005.csv",
                      sep = ",", skip = i-1, nrow = 1) %>%
    t() %>%
    as_tibble()

  y_velocity <- fread(input = "FVCOM_data/Clyde_Vertically_Averaged_y_velocity_2005.csv",
                      sep = ",", skip = i-1, nrow = 1) %>%
    t() %>%
    as_tibble()

  ## Calculate velocity
  Velrow <- sqrt(x_velocity^2 + y_velocity^2)

  ## Write RDS file
  Velrow %>%
    write_fst(paste0(i,"_2005.fst"), compress = 0)

  setTxtProgressBar(pb,i/n)
}

## read in data and calculate dimensionless current velocity (2005 data)
pb <- txtProgressBar(style = 3)
for (i in 20675:n){

  x_velocity <- fread(input = "FVCOM_data/Clyde_Vertically_Averaged_x_velocity_2006.csv",
                      sep = ",", skip = i-1, nrow = 1) %>%
    t() %>%
    as_tibble()

  y_velocity <- fread(input = "FVCOM_data/Clyde_Vertically_Averaged_y_velocity_2006.csv",
                      sep = ",", skip = i-1, nrow = 1) %>%
    t() %>%
    as_tibble()

  ## Calculate velocity
  Velrow <- sqrt(x_velocity^2 + y_velocity^2)

  ## Write RDS file
  Velrow %>%
    write_fst(paste0(i,"_2006.fst"), compress = 0)

  setTxtProgressBar(pb,i/n)
}

## Define function that will summarise my data
read_and_summarize <- function(ff){
  read_fst(ff) %>%
    summarise_all(funs(mean, min, max))
}

## Read in each line and apply the summary (2005)
Velocity2005 <- dir() %>%
  as_tibble() %>%
```

```
  filter(endsWith(value, "_2005.fst")) %>%
  mutate(data = purrr::map(value, read_and_summarize)) %>%
  unnest()

Clyde_LonLat <- Clyde_LonLat %>%
  mutate(value = 1:n())

Velocity2005 <- Velocity2005 %>%
  mutate(value = as.numeric(gsub("([0-9]+).*$", "\\1", value))) %>%
  left_join(Clyde_LonLat, by = c("value")) %>%
  rename(MinVelocity = min,
         MaxVelocity = max,
         MeanVelocity = mean)

## Read in each line and apply the summary (2006)
Velocity2006 <- dir() %>%
  as_tibble() %>%
  filter(endsWith(value, "_2006.fst")) %>%
  mutate(data = purrr::map(value, read_and_summarize)) %>%
  unnest()

Clyde_LonLat <- Clyde_LonLat %>%
  mutate(value = 1:n())

Velocity2006 <- Velocity2006 %>%
  mutate(value = as.numeric(gsub("([0-9]+).*$", "\\1", value))) %>%
  left_join(Clyde_LonLat, by = c("value")) %>%
  rename(MinVelocity = min,
         MaxVelocity = max,
         MeanVelocity = mean)

## Find the annual mean at each of the grid points
MeanMinMaxVelocity <- Velocity2005 %>%
  left_join(Velocity2006, by = c("Lon", "Lat")) %>%
  mutate(MeanVelocity = (MeanVelocity.x + MeanVelocity.y)/2,
         MinVelocity = (MinVelocity.x + MinVelocity.y)/2,
         MaxVelocity = (MaxVelocity.x + MaxVelocity.y)/2) %>%
  select(Lat, Lon, MeanVelocity, MinVelocity, MaxVelocity)

# ----------------------------------------#
# Minimum, mean and maximum bed shear stress
# ----------------------------------------#

## ShearStress data
ShearStress2005 <- fread("FVCOM_data/Clyde_Bed_Stress_tauc_2005.csv")
ShearStress2006 <- fread("FVCOM_data/Clyde_Bed_Stress_tauc_2006.csv")

## Concatenate all variable names
var <- paste(names(ShearStress2005), collapse = ", ")

## Create an atomic string with the command to be evaluated
var_Min <- paste0("min(",var,")")
var_Max <- paste0("max(",var,")")
```

```r
## Calculate minimum and maximum shear stress
MinMaxShear2005 <- ShearStress2005 %>%
  rowwise() %>%
  mutate_(.dots = setNames(var_Min, "MinShear")) %>%
  mutate_(.dots = setNames(var_Max, "MaxShear")) %>%
  select(MinShear, MaxShear)

# Calculate mean shear stress
MeanShear2005 <- ShearStress2005 %>%
  mutate(MeanShear = rowMeans(.)) %>%
  select(MeanShear)

## Add Longitude and Latitude information for each point
MinMaxShear2005 <- cbind(Clyde_LonLat, MinMaxShear2005)
MeanShear2005 <- cbind(Clyde_LonLat, MeanShear2005)

## Concatenate all variable names
var <- paste(names(ShearStress2006), collapse = ", ")

## Create an atomic string with the command to be evaluated
var_Min <- paste0("min(",var,")")
var_Max <- paste0("max(",var,")")

## Calculate minimum and maximum shear stress
MinMaxShear2006 <- ShearStress2006 %>%
  rowwise() %>%
  mutate_(.dots = setNames(var_Min, "MinShear")) %>%
  mutate_(.dots = setNames(var_Max, "MaxShear")) %>%
  select(MinShear, MaxShear)

## Calculate mean shear stress
MeanShear2006 <- ShearStress2006 %>%
  mutate(MeanShear = rowMeans(.)) %>%
  select(MeanShear)

## Add Longitude and Latitude information for each point
MinMaxShear2006 <- cbind(Clyde_LonLat, MinMaxShear2006)
MeanShear2006 <- cbind(Clyde_LonLat, MeanShear2006)

## Find the annual mean at each of the grid points
MeanMinMaxShear <- MinMaxShear2005 %>%
  left_join(MeanShear2005, by = c("Lon", "Lat")) %>%
  left_join(MinMaxShear2006, by = c("Lon", "Lat")) %>%
  left_join(MeanShear2006, by = c("Lon", "Lat")) %>%
  mutate(MeanShear = (MeanShear.x + MeanShear.y)/2,
         MinShear = (MinShear.x + MinShear.y)/2,
         MaxShear = (MaxShear.x + MaxShear.y)/2) %>%
  select(Lat, Lon, MeanShear, MinShear, MaxShear)

# ----------------------------------#
# Distance to the nearest coastline
# ----------------------------------#
```

```r
## Collate calculated environmental variables
Clyde_IrregularGrid <- FVCOM_bathymetry %>%
  left_join(MeanMinMaxVelocity, by = c("Lat", "Lon")) %>%
  left_join(MeanMinMaxShear,    by = c("Lat", "Lon")) %>%
  rename(Latitude = Lat,
         Longitude = Lon)

## Define a function to calculate distance to nearest coastline
coast_distance <- function(Longitude, Latitude){
  dist2Line(c(Longitude,Latitude), Shape_coast, distfun = distCosine)[1,1] %>% as.numeric()
}

## Calculate distance to coastline for each point on grid
pb <- txtProgressBar(style = 3)
for (i in 1:nrow(Clyde_IrregularGrid)) {

  Clyde_IrregularGrid$DistCoast[i] <- coast_distance(Longitude = Clyde_IrregularGrid$Longitude[i],
                                                     Latitude =  Clyde_IrregularGrid$Latitude[i])
  setTxtProgressBar(pb, i/nrow(Clyde_IrregularGrid))
}
```

4 Rock Distribution

In this section, we model the presence/absence distribution of rock seabed in the inner Firth of Clyde as a function of environmental covariates. We use outer Firth rock distribution maps from BGS to train Random Forests models and facilitate predictions.

4.1 Define functions for cross-validation

Function to sample spatially explicit training and test seabed rock data. We assign 2/3 data to training and the remainder as test data.

- **df_nested**: A dataframe nested according to latitude/longitude bins

Return: A list of summary statistics.

```r
sample_rockdata <- function(df_nested) {

  sample_ok <- FALSE

  while(sample_ok == FALSE) {
      ## Unnest data and randomly assign 2/3 data as training
      sample_data <- df_nested %>%
        ungroup() %>%
        mutate(Random = runif(nrow(df_nested))) %>%
        mutate(Train = Random < 0.66)

      ## separate training and test datasets
      train_data <- sample_data %>%
        filter(Train) %>%
        select(data) %>%
        unnest(cols = c(data))

      test_data <- sample_data %>%
```

```
        filter(!Train) %>%
        select(data) %>%
        unnest(cols = c(data))

      ## Check that data exist in both training and test sets
      if (nrow(test_data) != 0 & nrow(train_data) != 0 &
          sum(train_data$Rock == "Y") > 0 & sum(test_data$Rock == "Y") > 0 ) {
        sample_ok <- TRUE
      }
    }

  return(list(train_data = train_data,
              test_data  = test_data))
}
```

Function to apply a spatial cross-validation procedure using a pre-specified model structure to supplied data. We assign 2/3 data to training and the remainder as test data.

- **df_nested**: A dataframe nested according to latitude/longitude bins
- **reps**: Number of models to be iteratively fitted

Return: A list of summary statistics.

```
spatial_R2_rock <- function(df_nested, reps = 1000, ...) {

  ## Prepare object to store iterative results
  all_r2 <- list()

  ## The following loop is carried out for each replicate
  pb <- txtProgressBar(style = 3)
  for(i in 1:reps){

    # ----------------------------------------#
    # Section 1: sample training and test data
    # ----------------------------------------#

    sample_data <- sample_rockdata(df_nested)
    train_data  <- sample_data$train_data
    test_data   <- sample_data$test_data

    # ----------------------------------------#
    # Section 2.1: fit Random Forests model
    # ----------------------------------------#

    ## 2500 trees were found sufficient to stabilise model outputs
    rf_model <- ranger(Rock ~ ., importance = "impurity", data = train_data, num.trees = 2500)

    # ----------------------------------------#
    # Section 2.2: Generate predictions & calculate performance statistics
    # ----------------------------------------#

    ## predict using test data and calculate performance metrics
    all_r2[[i]] <- test_data %>%
      mutate(Predict     = predict(rf_model,.) %>%
               predictions(),
```

```
            Probability = predict(rf_model,.,predict.all = TRUE) %>%
              predictions() %>% apply(.,MARGIN =  1, mean)) %>%
      mutate(Probability = Probability - 1) %>%
      select(Rock, Predict, Probability) %>%
      summarize(Accuracy    = sum(Rock == Predict)/length(Predict),
                AUC         = auc(suppressMessages(roc(as.numeric(as.factor(Rock)),
                                                       as.numeric(as.factor(Predict)))))[1],
                MSE         = mean(Probability - (as.numeric(Rock)-1)),
                Sensitivity = sum(Predict == "Y" & Predict == Rock)/sum(Predict == "Y"),
                Specificity = sum(Predict == "N" & Predict == Rock)/sum(Predict == "N")) %>%
      ungroup()

    setTxtProgressBar(pb, i/reps)
  }

  # ------------------------------------------------------------#
  # Section 3: Collate and summarise model performance statistics
  # ------------------------------------------------------------#

  ## find mean error across iterations and bind to results vector
  results_vector <- all_r2 %>%
    bind_rows() %>%
    mutate_all(function(x){
      cumsum(x)/row_number()})

  ## Find the overall summary performance for the cross-validation
  result_summary <- results_vector %>%
    summarize(Accuracy = tail(Accuracy, 1),
              AUC      = tail(AUC, 1),
              MSE      = tail(MSE, 1),
              Sensitivity = tail(Sensitivity, 1),
              Specificity = tail(Specificity, 1))

  ## Return the raw, processed and overall performance statistics
  return(list(Rsq_raw = all_r2 %>% bind_rows(),
              Rsq_vec = results_vector,
              Rsq_sum = result_summary))
}
```

4.2 Import rock data and covariates

The first step is the importation of base data layers and covariate data. Where necessary, covariate data have been interpolated to an irregular grid using the akima package. See main data paper for details on how this was performed.

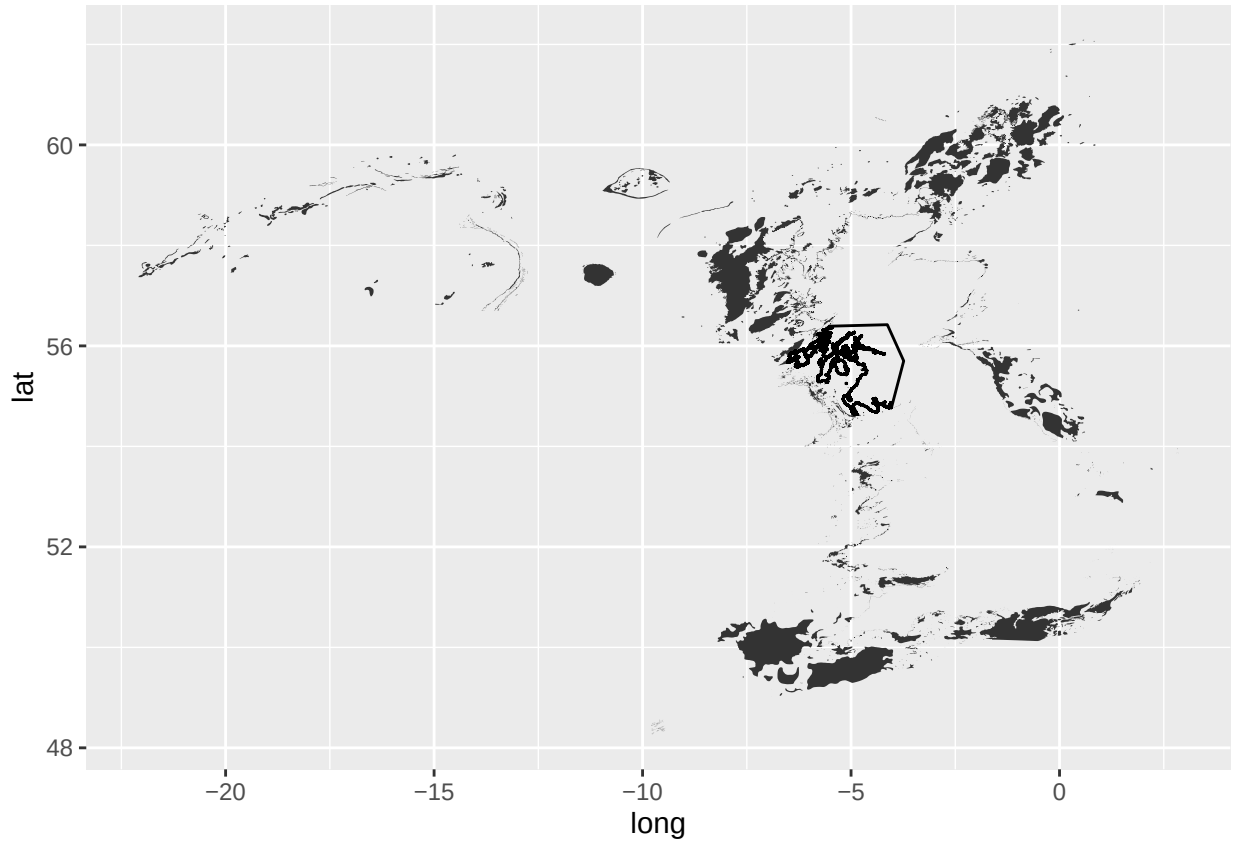
```
## Import Polygon of BGS rock cover
Clyde_rock_BGS <-  rgdal::readOGR(paste0(filepath, "BGS_250k_HardSubstrate_WGS84_v1.shp"))
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\matt8\Dropbox (Glasgow Uni)\PhD\Manuscripts\Modelling Seabed Clyde\supplementary_ma
## with 3474 features
## It has 10 fields
```

```
Clyde_rock_BGS <- spTransform(Clyde_rock_BGS, CRS("+proj=longlat +datum=WGS84"))
```

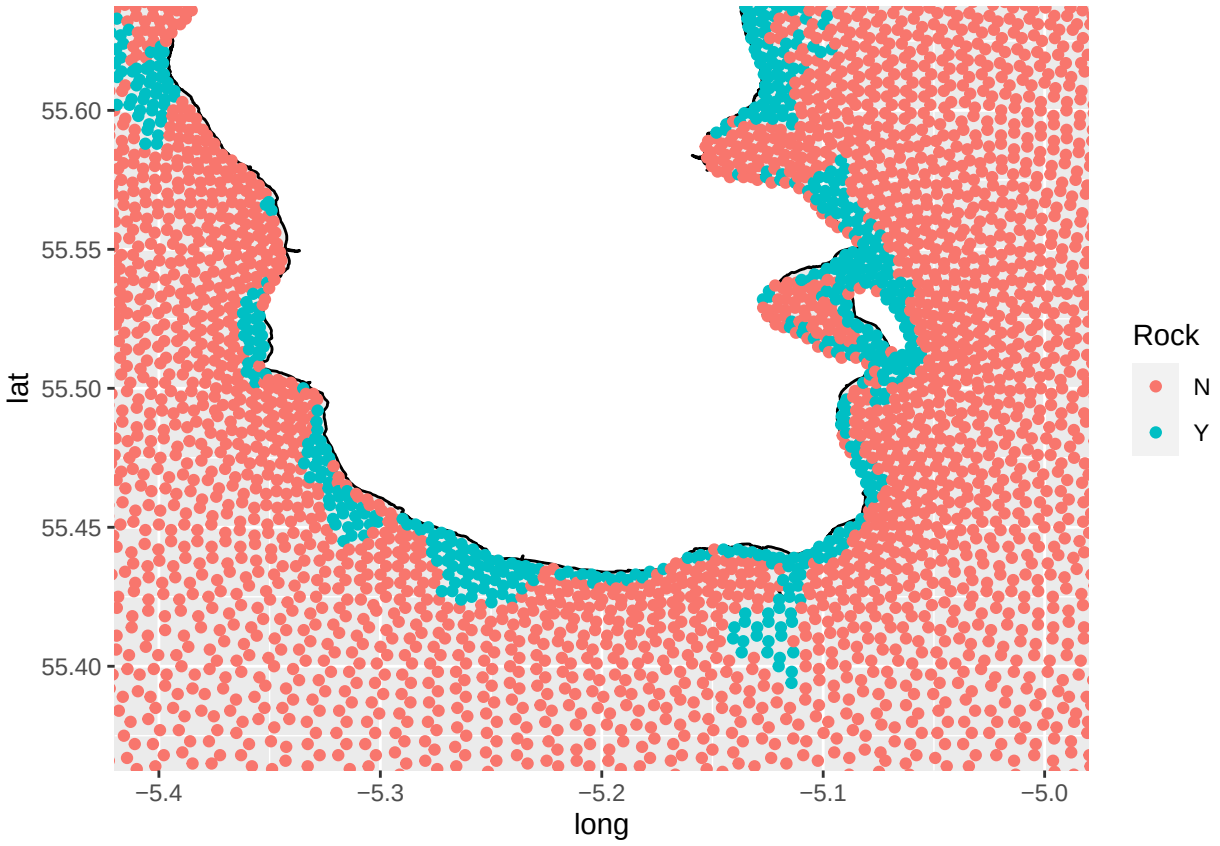
```
## Import UKHO points for substrate type
Clyde_rock_UKHO <- fread(paste0(filepath, "Clyde_RockCover_UKHO.csv"))

## Import Clyde grid (with covariates)
Clyde_grid <- fread(paste0(filepath, "Clyde_IrregularGrid.csv"))

#' First step is to identify points on the grid where rock was recorded.

## Check distribution of data
ggplot() +
  geom_polygon(aes(x = long, y = lat, group = group), data = fortify(Clyde_rock_BGS)) +
  geom_path(aes(x = long, y = lat, group = group), data = Shape_coast_f)
```

```
## Regions defined for each Polygons
```




```
## Convert to spatialpointsdataframe
coordinates(Clyde_grid) <- Clyde_grid[,c("Longitude","Latitude")]
proj4string(Clyde_grid) <- CRS("+proj=longlat +datum=WGS84")

Clyde_grid$Object_ID <- over(x = Clyde_grid, y = Clyde_rock_BGS)$OBJECTID

Clyde_grid <- Clyde_grid@data %>%
  mutate(Rock = "N",
         Rock = ifelse(!is.na(Object_ID), "Y", Rock))

## Divide into training and test datasets
```

```
Clyde_grid$Train <- Clyde_grid$Latitude <=
  max(Clyde_grid$Latitude[!is.na(Clyde_grid$Object_ID) & Clyde_grid$Longitude > -5.4])

## Plot distribution of points
ggplot() +
  geom_polygon(aes(x = long, y = lat, group = group), colour = "black", fill = "white",
               data = Shape_coast_f) +
  geom_point(aes(x = Longitude, y = Latitude, colour = Rock), data = Clyde_grid) +
  coord_cartesian(xlim = c(-5.4, -5), ylim = c(55.375, 55.625))
```



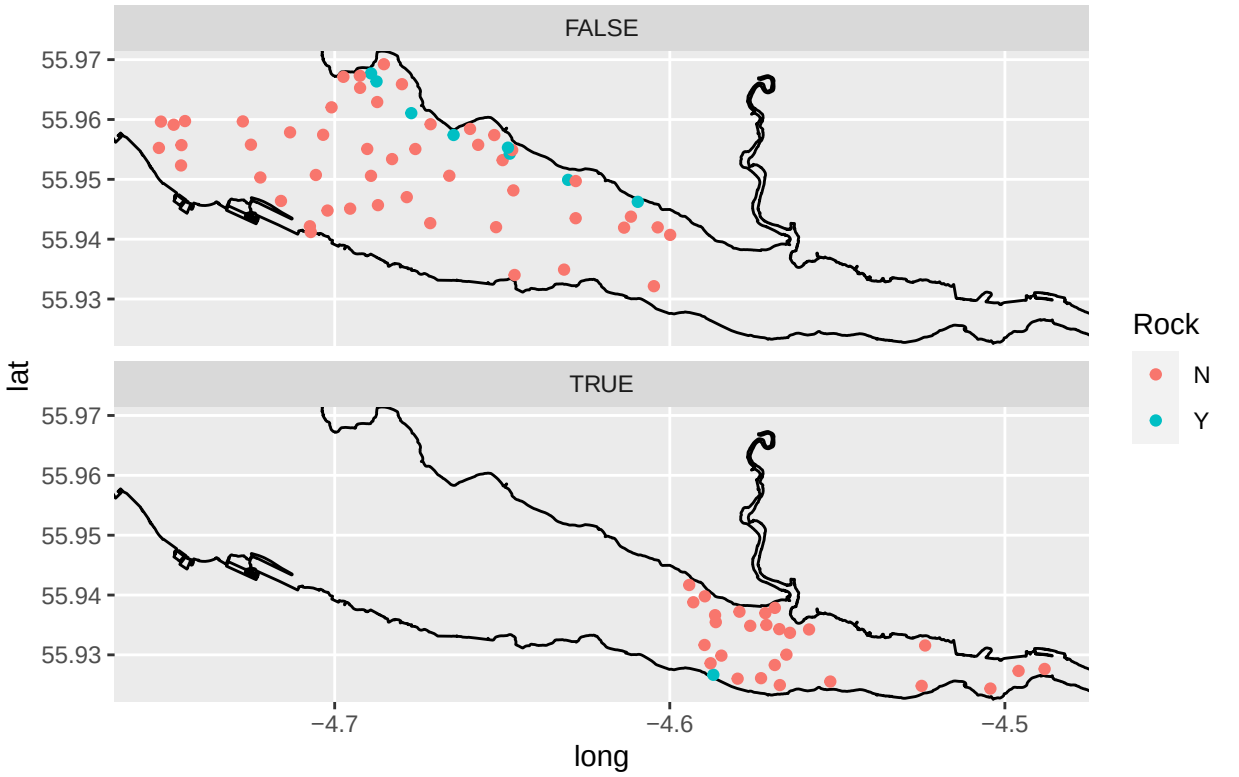

```
## How many data points added from Admiralty Charts?
Clyde_rock_UKHO %>%
  select(-Substrate, -Longitude, -Latitude) %>%
  drop_na() %>%
  nrow() ## 57
```

```
## [1] 57
```

4.3 Model fitting

```
## Clean up UKHO data
Clyde_rock_UKHO <- Clyde_rock_UKHO %>%
  mutate(Rock = ifelse(Substrate %in% c("stone", "rock", "boulder"),
                       "Y", "N"))

## plot distribution of available rock data
```

```
Clyde_rock_UKHO %>%
  mutate(OutSideBoundary = is.na(MeanVelocity)) %>%
  ggplot()+
  geom_path(aes(x = long, y = lat, group = group),data = Shape_coast_f) +
  geom_point(aes(x = Longitude, y = Latitude, colour = Rock)) +
  coord_map(xlim = c(min(Clyde_rock_UKHO$Longitude),max(Clyde_rock_UKHO$Longitude)),
            ylim = c(min(Clyde_rock_UKHO$Latitude),max(Clyde_rock_UKHO$Latitude))) +
  facet_wrap( ~ OutSideBoundary, ncol = 1)
```

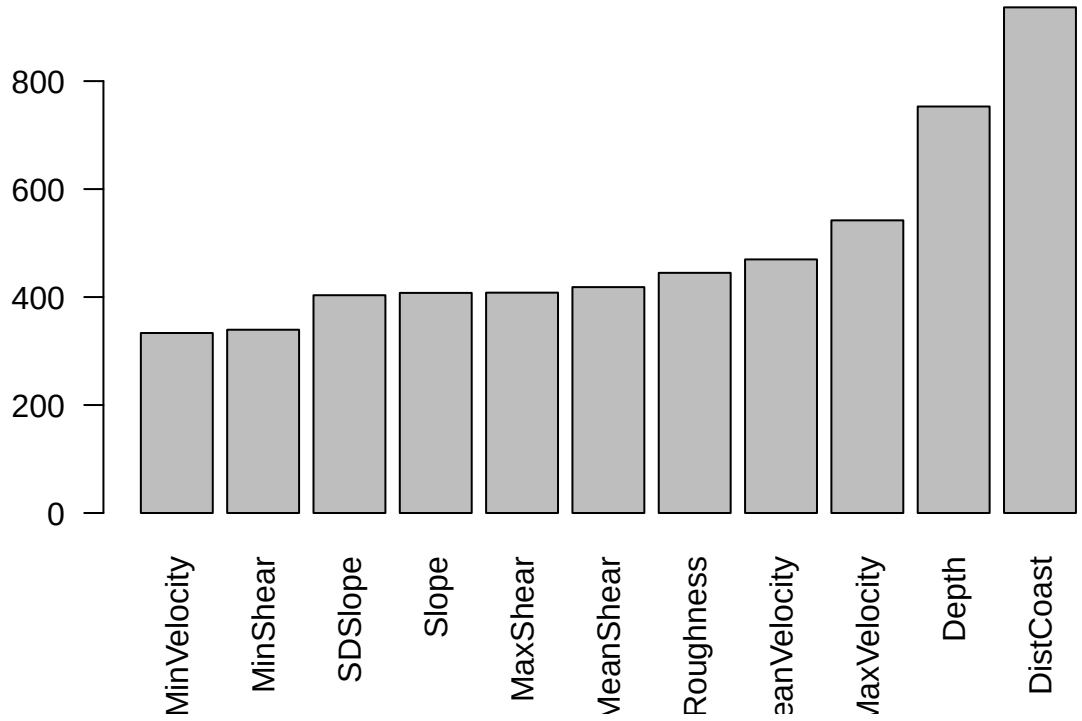



```
## Fit full model
Rock_rf <- Clyde_grid %>%
  filter(Train) %>%
  select(-Latitude, -Longitude, -Train, -Object_ID) %>%
  bind_rows(Clyde_rock_UKHO %>% select(-Substrate, -Longitude, -Latitude) %>% drop_na()) %>%
  mutate(Rock = as.factor(Rock)) %>%
  ranger(Rock ~ ., data = ., importance = "impurity", num.trees = 2500)

## Check out-of-bag prediction Error
Rock_rf # 10.15%
```

```
## Ranger result
##
## Call:
##  ranger(Rock ~ ., data = ., importance = "impurity", num.trees = 2500)
##
## Type:                             Classification
```

```
## Number of trees:                  2500
## Sample size:                      23237
## Number of independent variables:  11
## Mtry:                             3
## Target node size:                 1
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error:             10.23 %
```

```
## Plot variable importance
barplot(Rock_rf$variable.importance[order(Rock_rf$variable.importance)], las = 2)
```




```
## Predict Rock presence
Clyde_grid <- Clyde_grid %>%
  mutate(Rock = ifelse(Train == FALSE, predict(Rock_rf, data = .) %>% predictions(), Rock))
```

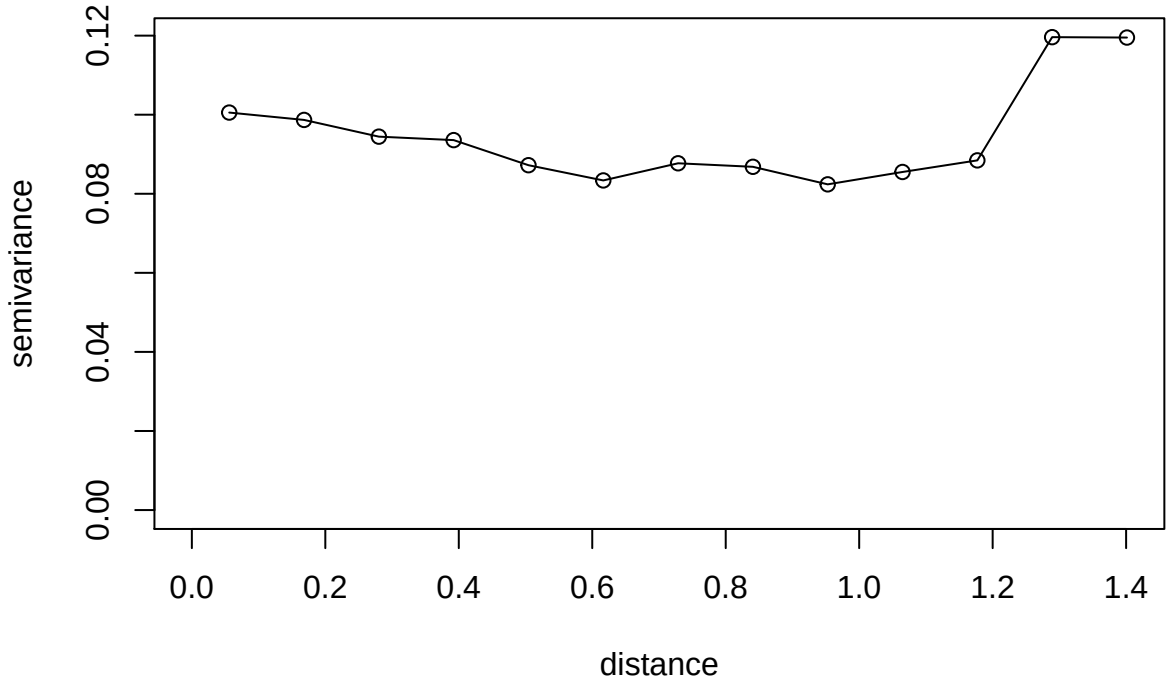
Limited evidence of spatial autocorrelation in rock residuals. 0.125 and 0.25 degree binning should suffice.

```
## compile all data - keep lat/lon to compute spatial autocorrelation
Clyde_Rock_all <- Clyde_grid %>%
  filter(Train) %>%
  select(-Train, -Object_ID) %>%
  bind_rows(Clyde_rock_UKHO %>% select(-Substrate) %>% drop_na()) %>%
  mutate(Rock = as.factor(Rock))

## Check spatial autocorrelation in rock model residuals - Compute semivariogram using geoR package
SampleRock_variogram <- geoR::variog(geodata = list(coords = matrix(c(Clyde_Rock_all$Latitude,
                                                                      Clyde_Rock_all$Longitude),
```

```
                                                                    ncol = 2),
                                                  data = Rock_rf$predictions == Clyde_Rock_all$Rock))
```

```
## variog: computing omnidirectional variogram
```

```
## Little autocorrelation evident
plot(SampleRock_variogram, type = "o")
```



4.4 Model spatial cross-validation

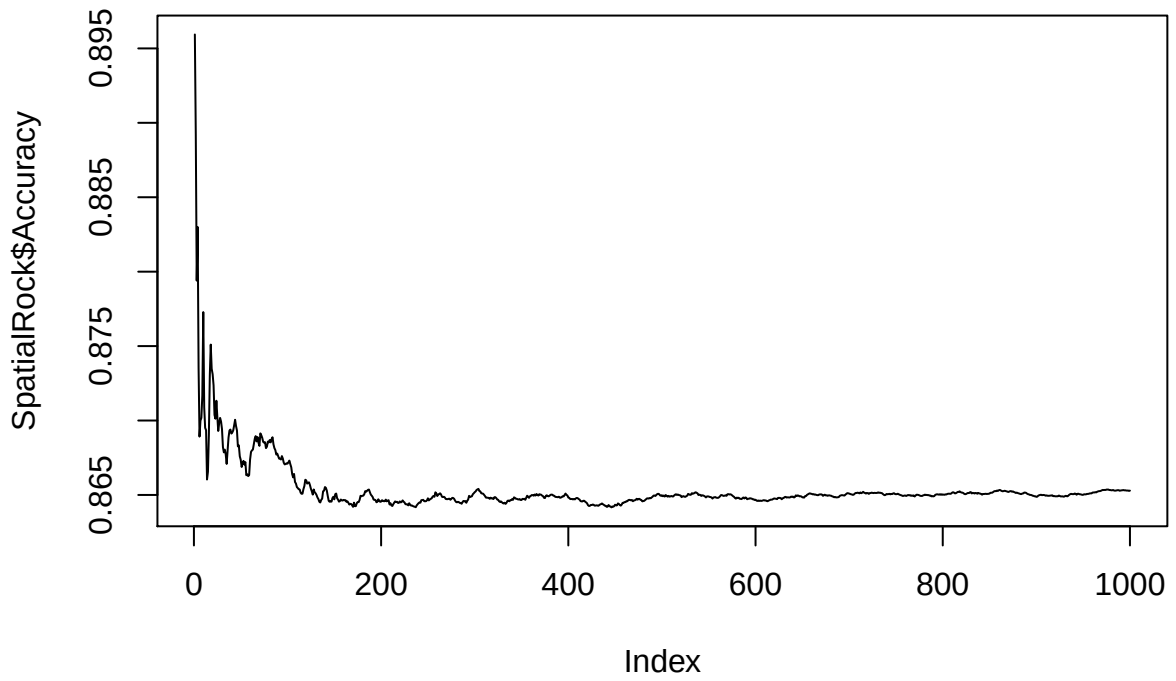
Spatial binning at 0.125 degrees latitude and 0.25 degrees longitude. 1000 iterations.

```
# ------------------------------------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT. This takes ~ 5.5 hours to run.
# ------------------------------------------------------------------------------------------#

## run cross-validation using 1000 iterations
SpatialRock <- Clyde_grid %>%
  filter(Train) %>%
  select(-Train, -Object_ID) %>%
  bind_rows(Clyde_rock_UKHO %>% select(-Substrate) %>% drop_na()) %>%
  mutate(bin_lat = findInterval(Latitude, vec = seq(55, 57, 0.125), left.open = TRUE),
         bin_lon = findInterval(Longitude, vec = seq(-6, -4, 0.250), left.open = TRUE)) %>%
  mutate(bin_lat = as.character(bin_lat),
         bin_lon = as.character(bin_lon)) %>%
  group_by(bin_lat, bin_lon) %>%
  select(-Longitude, -Latitude) %>%
```

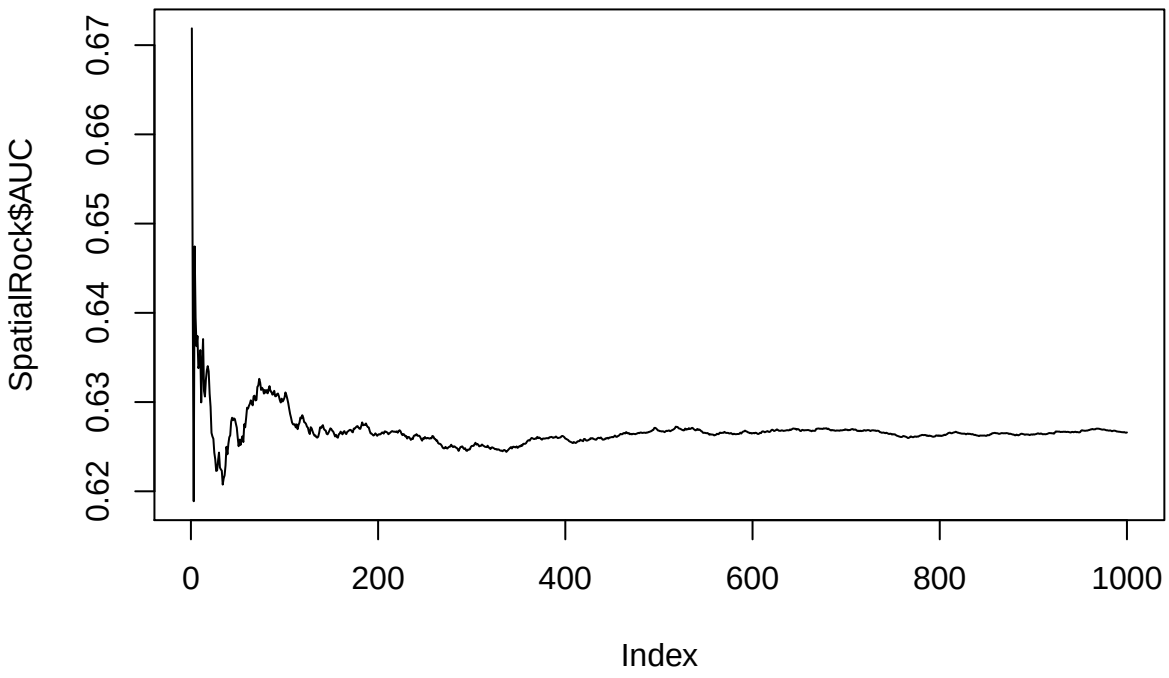
```
  mutate(Rock = as.factor(Rock)) %>%
  nest() %>%
  spatial_R2_rock(df_nested = ., reps = 1000)

fwrite(SpatialRock$Rsq_raw, "RockDistribution_SpatialCrossValidation_raw.csv")
```

All model performance metrics are stabilised at 1000 iterations. Cross-validation shows that presence-absence models for seabed rock have relatively high overall accuracy and low mean square error.

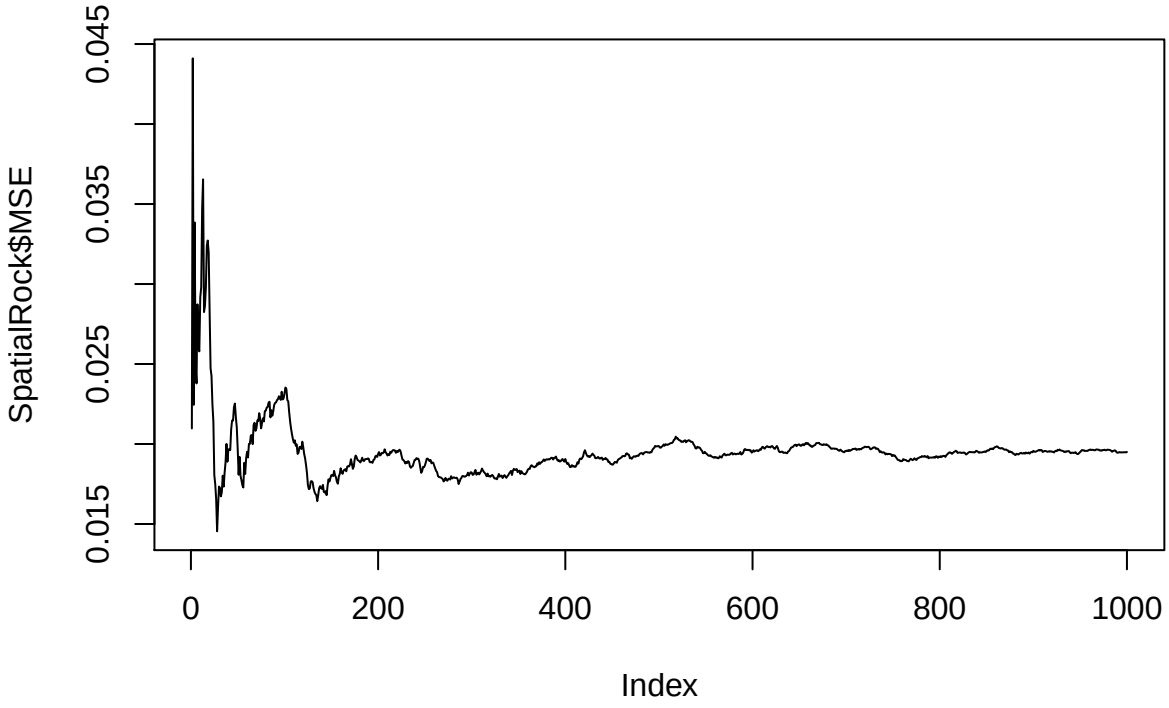
```
# ------------------------------------------------------------------------------------------#
# LOAD PREVIOUS RUN.
# ------------------------------------------------------------------------------------------#

SpatialRock <- data.table::fread("RockDistribution_SpatialCrossValidation_raw.csv")

SpatialRock <- SpatialRock %>%
  mutate_all(function(x){
    cumsum(x)/row_number()})

## Plot performance metrics
plot(SpatialRock$Accuracy, type = "l")
```



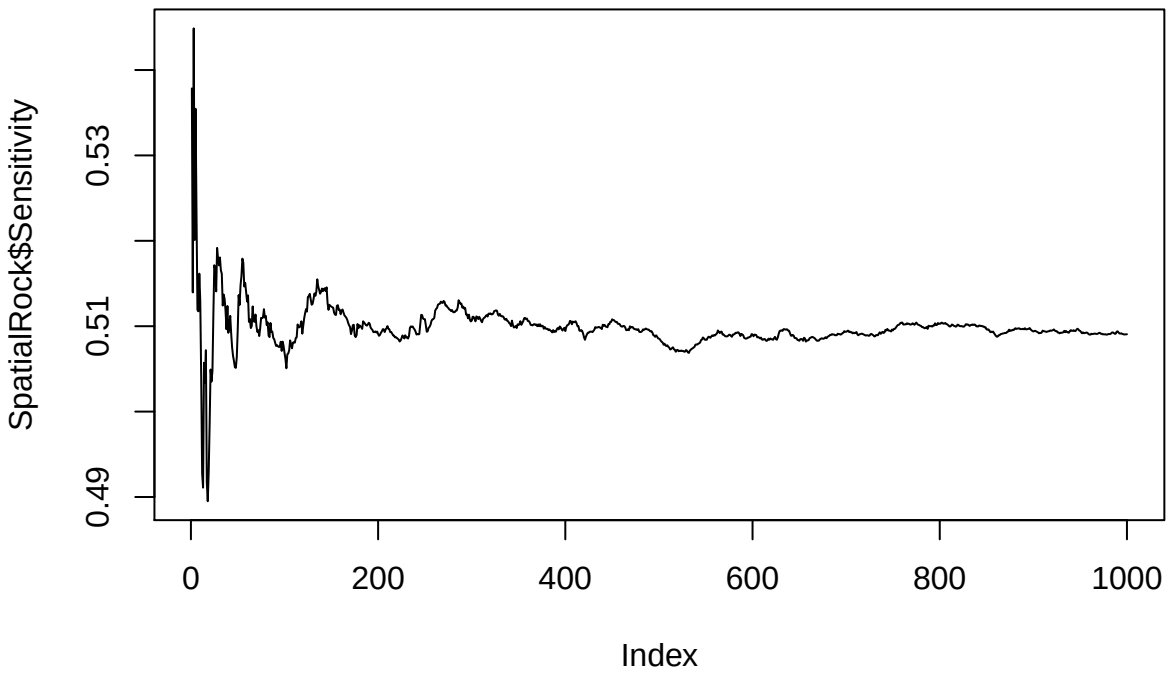

```
plot(SpatialRock$AUC, type = "l")
```



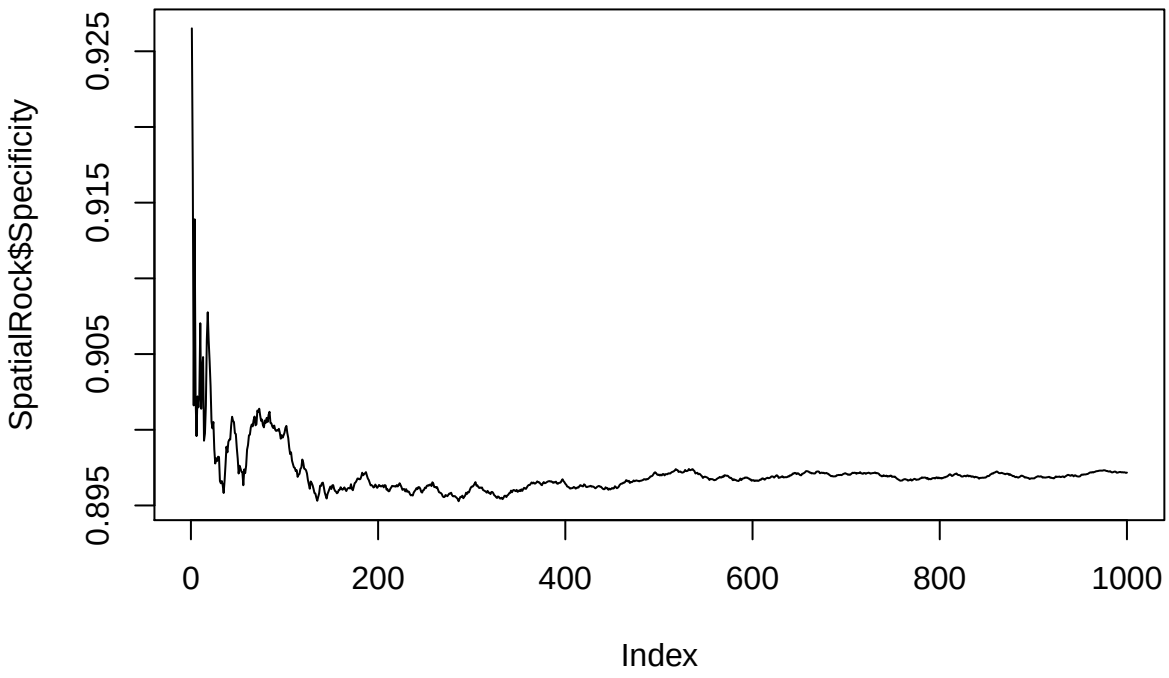

```
plot(SpatialRock$MSE, type = "l")
```




```
plot(SpatialRock$Sensitivity, type = "l")
```




```
plot(SpatialRock$Specificity, type = "l")
```



5 Sediment fractions

In this section, we model the percentage mud, sand and gravel content in Clyde seabed sediments as a function of environmental covariates. Using fitted models, we then predict the spatial distribution sediment fractional composition in the Firth of Clyde. We use historical published mineralogy datasets for the Firth of Clyde to train Random Forests models and facilitate predictions.

5.1 Define functions for cross-validation

Function to sample spatially explicit training and test sediment fraction data. Because two Random Forests models are required to model the sediment compositional data (see main data paper), two separate nested dataframes must be supplied. We assign 2/3 data to training and the remainder as test data.

- **df_nested_SM**: A dataframe containing ratios of sand and mud nested according to latitude/longitude bins
- **df_nested_GM**: A dataframe containing ratios of gravel and mud nested according to latitude/longitude bins
- **reps**: Number of models to be iteratively fitted

Return: A list of summary statistics

```
sample_sedimentfractions <- function(df_nested_SM, df_nested_GM) {

  sample_ok <- FALSE

    while(sample_ok == FALSE) {
```

```
      Random <- sample(nrow(df_nested_GM))/nrow(df_nested_GM)

      ## Sample Test and training data
      sample_data_GM <- df_nested_GM %>%
        ungroup() %>%
        mutate(Train = Random < 0.66)

      sample_data_SM <- df_nested_SM %>%
        ungroup() %>%
        mutate(Train = Random < 0.66)

      ## Extract training data
      train_data_GM <- sample_data_GM %>%
        filter(Train) %>%
        select(data) %>%
        unnest(cols = c(data)) %>%
        select(-Gravel, -Mud, -lonlat)

      train_data_SM <- sample_data_SM %>%
        filter(Train) %>%
        select(data) %>%
        unnest(cols = c(data)) %>%
        select(-Sand, -Mud, -lonlat)

      ## Extract test data
      test_data_GM <- sample_data_GM %>%
        filter(!Train) %>%
        select(data) %>%
        unnest(cols = c(data))

      test_data_SM <- sample_data_SM %>%
        filter(!Train) %>%
        select(data) %>%
        unnest(cols = c(data))

      ## Check that data exist in both training and test sets
      if (nrow(test_data_GM) != 0 & nrow(train_data_GM) != 0 &
          nrow(test_data_SM) != 0 & nrow(train_data_SM) != 0) {
        sample_ok <- TRUE
      }
    }

  return(list(train_data_GM = train_data_GM,
              train_data_SM = train_data_SM,
              test_data_GM  = test_data_GM,
              test_data_SM  = test_data_SM))
}
```

Function to apply a spatial cross-validation procedure using a pre-specified model structure to supplied data.The function assumes that the sediment fraction data are in percentages rather than proportions.

Because two Random Forests models are required to model the sediment compositional data (see main data paper), two separate nested dataframes must be supplied. We assign 2/3 data to training and the remainder as test data.

- **df_nested_SM**: A dataframe containing ratios of sand and mud nested according to latitude/longitude bins
- **df_nested_GM**: A dataframe containing ratios of gravel and mud nested according to latitude/longitude bins
- **reps**: Number of models to be iteratively fitted

Return: A list of summary statistics

```
spatial_logR_R2 <- function(df_nested_SM, df_nested_GM, reps = 200, ...) {

  all_r2 <- list()

  ## The following loop is carried out for each replicate
  pb <- txtProgressBar(style = 3)
  for(i in 1:reps){

    # ----------------------------------------#
    # Section 1: sample training and test data
    # ----------------------------------------#

    sample_data <- sample_sedimentfractions(df_nested_SM, df_nested_GM)
    train_data_GM <- sample_data$train_data_GM
    train_data_SM <- sample_data$train_data_SM
    test_data_GM  <- sample_data$test_data_GM
    test_data_SM  <- sample_data$test_data_SM

    # ----------------------------------------#
    # Section 2.1: fit Random Forests model
    # ----------------------------------------#

    RF_GM <- ranger(logGM ~ . , importance = "impurity", data = train_data_GM, ...)
    RF_SM <- ranger(logSM ~ . , importance = "impurity", data = train_data_SM, ...)

    # ----------------------------------------#
    # Section 2.2: Generate predictions & calculate performance statistics
    # ----------------------------------------#

    # Quick note here: the reason that na.rm is necessary for the Gravel calculations
    # is that the MSS measurements miss Gravel completely and hence NA is in the
    # source data

    ## Generate predictions and calculate summary statistics
    all_r2[[i]] <- test_data_SM %>%
      left_join(test_data_GM %>% select(lonlat, Mud, Gravel),
                by = c("lonlat", "Mud")) %>%
      mutate(Predict_GM = predict(RF_GM, .) %>% predictions(),
             Predict_SM = predict(RF_SM, .) %>% predictions()) %>%
      mutate(Predict_Sand = (exp(Predict_SM)/(1+exp(Predict_SM)+exp(Predict_GM)))*100,
             Predict_Gravel = (exp(Predict_GM)/(1+exp(Predict_SM)+exp(Predict_GM)))*100,
             Predict_Mud = (1/(1+exp(Predict_SM)+exp(Predict_GM)))*100) %>%
      select(Mud, Gravel, Sand, Predict_Mud, Predict_Gravel, Predict_Sand) %>%
      summarise(EucDist = mean(sqrt((Mud - Predict_Mud)^2 +
                                      (Sand - Predict_Sand)^2 +
                                      (Gravel - Predict_Gravel)^2),
                               na.rm = TRUE),
```

```
                MSE = mean((Mud - Predict_Mud)^2) +
                  mean((Sand - Predict_Sand)^2) +
                  mean((Gravel - Predict_Gravel)^2,
                       na.rm = TRUE),
                aRMSE = (sqrt(mean((Mud - Predict_Mud)^2)) +
                           sqrt(mean((Sand - Predict_Sand)^2)) +
                           sqrt(mean((Gravel - Predict_Gravel)^2,
                                     na.rm = TRUE)))/3,
                R2_Mud = caret::R2(pred = Predict_Mud, obs = Mud,
                                   formula = "traditional"),
                R2_Sand = caret::R2(pred = Predict_Sand, obs = Sand,
                                    formula = "traditional"),
                R2_Gravel = caret::R2(pred = Predict_Gravel,
                                      obs = Gravel,
                                      formula = "traditional",
                                      na.rm = TRUE)) %>%
      ungroup()

    setTxtProgressBar(pb, i/reps)
  }

  # ------------------------------------------------------------------#
  # Section 3: Collate and summarise model performance statistics
  # ------------------------------------------------------------------#

  ## find mean error across iterations and bind to results vector
  results_vector <- all_r2 %>%
    bind_rows() %>%
    mutate_all(function(x){
      cumsum(x)/row_number()})

  result_summary <- results_vector %>%
    summarise_all(function(x){tail(x,1)})

  return(list(Rsq_raw = all_r2 %>% bind_rows(),
              Rsq_vec = results_vector,
              Rsq_sum = result_summary))
}
```

5.2 Load training data

Legacy mud, sand and gravel data with associated environmental covariates is loaded. BGS data requires correction from sieve to laser diffraction estimates of mud content. See data paper for details.

```
# Import the three available data sets that I will predict from
BGS_Clyde <- fread(paste0(filepath,"BGS_Clyde_Grab_covariates_FINAL.csv"))
MSS_Clyde <- fread(paste0(filepath,"MSS_Clyde_covariates_FINAL.csv"))
SEPA_Clyde<- fread(paste0(filepath,"SEPA_Clyde_covariates_FINAL.csv"))

## Remove NAs
BGS_Clyde <- BGS_Clyde %>% drop_na()
MSS_Clyde <- MSS_Clyde %>%  drop_na()
SEPA_Clyde<- SEPA_Clyde %>%  drop_na()
```

```
## Correct BGS data < 60 % mud to laser estimates of mud
BGS_Clyde <- BGS_Clyde %>%
  mutate(MudCorr =  (3.157*(((Mud/(Mud + Sand))*100)^0.7225)*(Mud + Sand)/100)) %>%
  mutate(Sand = ifelse(Mud > 0 & Mud/(Sand+Mud) < 0.6,
                       Sand - (MudCorr - Mud),
                       Sand),
         Mud  = ifelse(Mud > 0 & Mud/(Sand+Mud) < 0.6,
                       MudCorr,
                       Mud)) %>%
  select(-MudCorr)
```

5.3 Fit additive log-ratio Random Forests model

```
## log(Sand) - log(Mud)
Combined_MudSand <- rbind(BGS_Clyde %>% select(-Gravel),
                          SEPA_Clyde %>% select(-Gravel),
                          MSS_Clyde)

## Impute 0 values with very small values
Combined_MudSand$Sand[Combined_MudSand$Sand == 0] <- 1e-06
Combined_MudSand$Mud[Combined_MudSand$Mud == 0] <- 1e-06

Combined_MudSand <- Combined_MudSand %>%
  mutate(logSM = log(Sand) - log(Mud)) %>%
  select(-c(Sand, Mud))

## Generate Random Forest
RF_SM <- ranger(logSM ~ . , importance = "impurity", data = Combined_MudSand, num.trees = 2500)
RF_SM
```

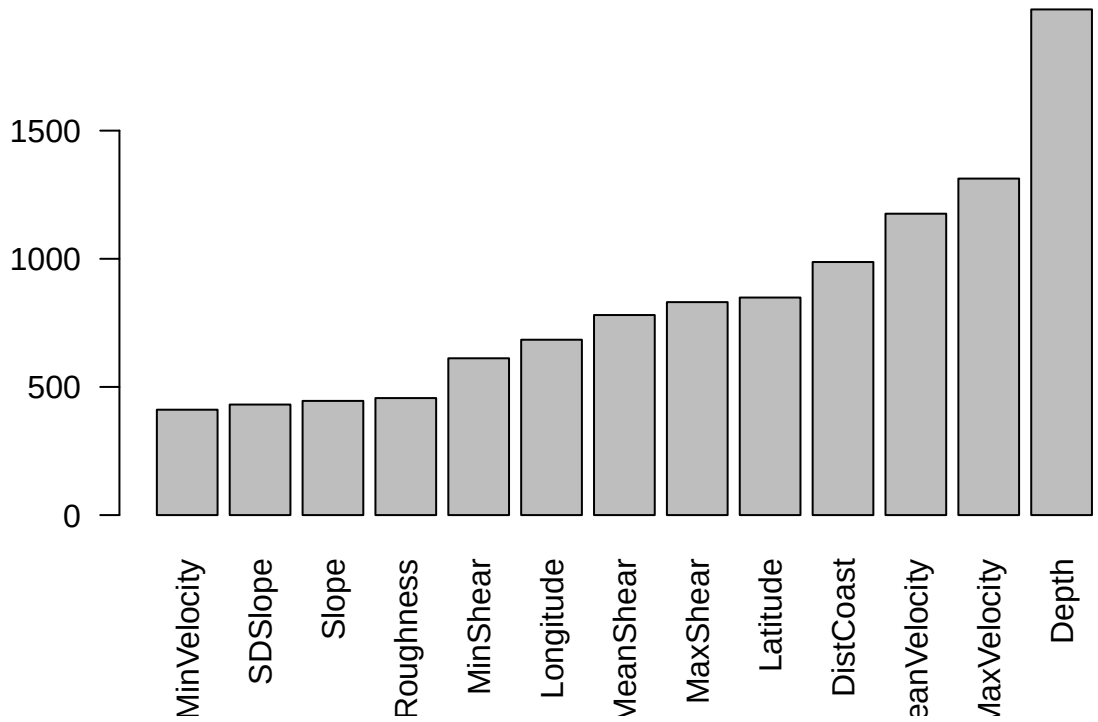
```
## Ranger result
##
## Call:
##  ranger(logSM ~ ., importance = "impurity", data = Combined_MudSand,      num.trees = 2500)
##
## Type:                             Regression
## Number of trees:                  2500
## Sample size:                      1388
## Number of independent variables:  13
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       3.823694
## R squared (OOB):                  0.5423005
```

```
## log(Gravel) - log(Mud)
Combined_MudGravel <- rbind(BGS_Clyde %>% select(-Sand),
                            SEPA_Clyde %>% select(-Sand))

## Impute 0 values with very small values
Combined_MudGravel$Gravel[Combined_MudGravel$Gravel == 0] <- 1e-06
Combined_MudGravel$Mud[Combined_MudGravel$Mud == 0] <- 1e-06
```

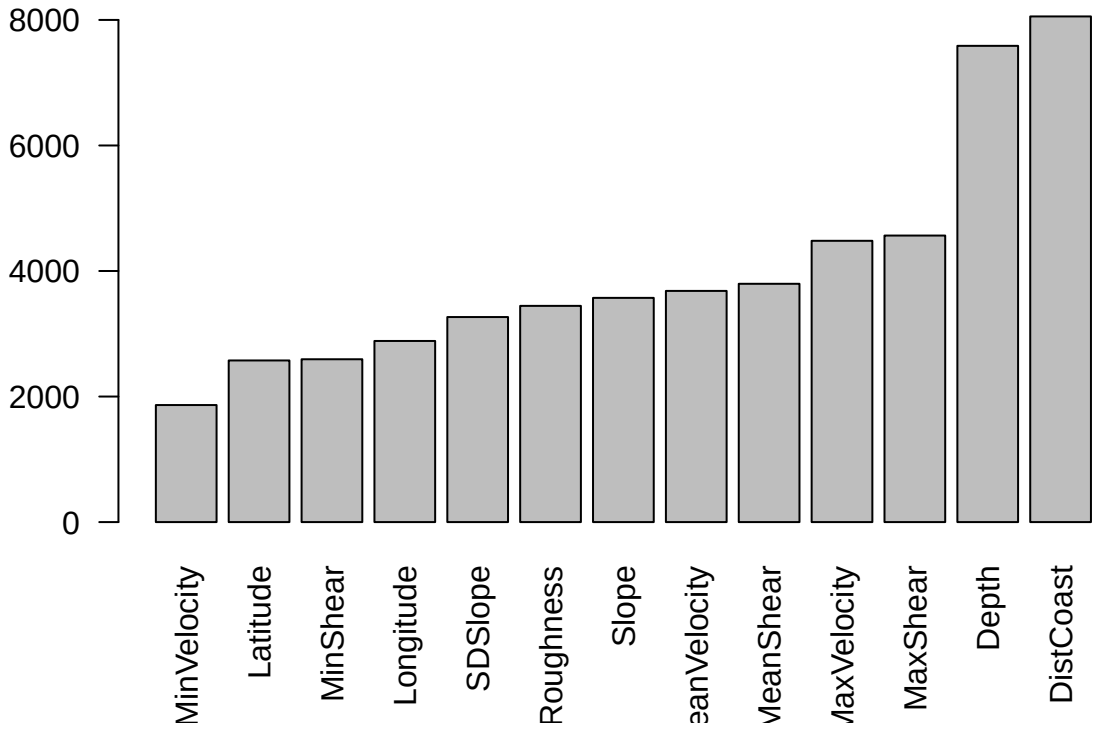
```r
Combined_MudGravel <- Combined_MudGravel %>%
  mutate(logGM = log(Gravel) - log(Mud)) %>%
  select(-c(Gravel, Mud))

## Generate Random Forest
RF_GM <- ranger(logGM ~ . , importance = "impurity", data = Combined_MudGravel, num.trees = 2500)
RF_GM
```

```
## Ranger result
##
## Call:
##  ranger(logGM ~ ., importance = "impurity", data = Combined_MudGravel,      num.trees = 2500)
##
## Type:                             Regression
## Number of trees:                  2500
## Sample size:                      943
## Number of independent variables:  13
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       27.16292
## R squared (OOB):                  0.5347998
```

```r
#' The figure below shows the variables that account for the most variation in
#' the ratio of sand and mud, and gravel and mud.

## plot variable importance
barplot(RF_SM$variable.importance[order(RF_SM$variable.importance)], las = 2)
```




```
barplot(RF_GM$variable.importance[order(RF_GM$variable.importance)], las = 2)
```



5.4 Generate predicted map of sediment fractions

The section of code below imports an unstructured grid comprising full-coverage mapped environmental covariates for the Firth of Clyde. This is then used to predict percentage mud, sand and gravel content for seabed sediments using the fitted additive log-ratio Random Forests models.

```
## import full-coverage environmental covariates
Clyde_IrregularGrid <- fread(paste0(filepath, "Clyde_IrregularGrid.csv"))

## Predict Sediment fractions for irregular network
logSM <- predict(RF_SM, data = Clyde_IrregularGrid, type = "response")$predictions
logGM <- predict(RF_GM, data = Clyde_IrregularGrid, type = "response")$predictions

## back-transform predictions
Clyde_IrregularGrid$Sand <- exp(logSM)/(1+exp(logSM)+exp(logGM))
Clyde_IrregularGrid$Gravel <- exp(logGM)/(1+exp(logSM)+exp(logGM))
Clyde_IrregularGrid$Mud <- 1/(1+exp(logSM)+exp(logGM))
```

The proposed bin size exceeds range of the spatial autocorrelation. There also appears to be some spatial trends in mud-sand across the entire range of the dataset - supports use of latitude and longitude as predictors in this model.

```
## Compute semivariogram using geoR package
Sample_GM_variogram <- geoR::variog(geodata = list(
  coords = matrix(c(Combined_MudGravel$Latitude,
                    Combined_MudGravel$Longitude),
                  ncol = 2),
```

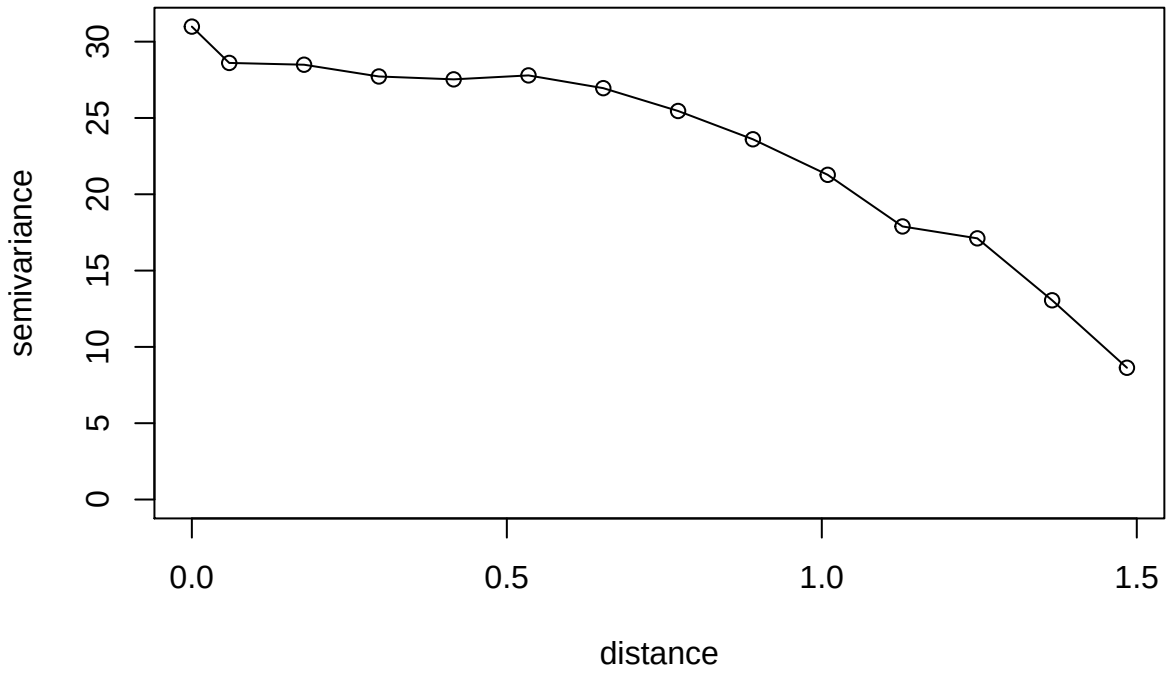
```
                  data = (Combined_MudGravel$logGM - RF_GM$predictions)))
```

```
## variog: computing omnidirectional variogram
## variog: co-locatted data found, adding one bin at the origin
```

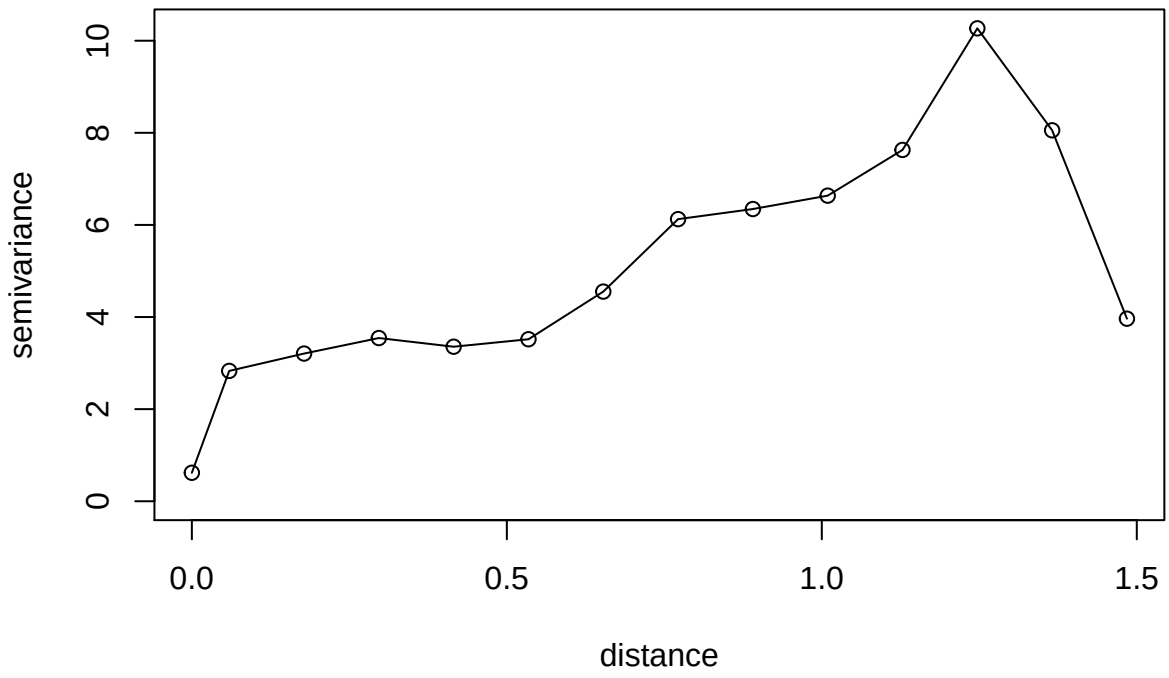
```
Sample_SM_variogram <- geoR::variog(geodata = list(
  coords = matrix(c(Combined_MudSand$Latitude,
                    Combined_MudSand$Longitude),
                  ncol = 2),
                  data = (Combined_MudSand$logSM - RF_SM$predictions)))
```

```
## variog: computing omnidirectional variogram
## variog: co-locatted data found, adding one bin at the origin
```

```
## spatial autocorrelation in Gravel-Mud model
plot(Sample_GM_variogram, type = "o")
```




```
## spatial autocorrelation in Sand-Mud model
plot(Sample_SM_variogram, type = "o")
```



5.5 Spatial cross-validation

The first step is to prepare the data objects for spatial cross-validation

```
## log(Sand) - log(Mud)
Combined_MudSand <- rbind(BGS_Clyde %>% select(-Gravel),
                          SEPA_Clyde %>% select(-Gravel),
                          MSS_Clyde)

## Impute 0 values with very small values
Combined_MudSand$Sand[Combined_MudSand$Sand == 0] <- 1e-06
Combined_MudSand$Mud[Combined_MudSand$Mud == 0] <- 1e-06

df_SM <- Combined_MudSand %>%
  mutate(logSM = log(Sand) - log(Mud)) %>%
  mutate(bin_lat = cut(Latitude, breaks = seq(54, 57, 0.125)),
         bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25)),
         lonlat = paste0(Longitude, Latitude)) %>%
  group_by(bin_lat, bin_lon) %>%
  nest()

## log(Gravel) - log(Mud)
Combined_MudGravel <- rbind(BGS_Clyde %>% select(-Sand),
                            SEPA_Clyde %>% select(-Sand))

## Impute 0 values with very small values
```

```
Combined_MudGravel$Gravel[Combined_MudGravel$Gravel == 0] <- 1e-06
Combined_MudGravel$Mud[Combined_MudGravel$Mud == 0] <- 1e-06

df_GM <- Combined_MudGravel %>%
  mutate(logGM = log(Gravel) - log(Mud)) %>%
  mutate(bin_lat = cut(Latitude, breaks = seq(54, 57, 0.125)),
         bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25)),
         lonlat = paste0(Longitude, Latitude)) %>%
  group_by(bin_lat, bin_lon) %>%
  nest()
```

Next, check that the spatial binning and subsequent bootstrapping, coupled with bootstrapping within the Random Forests model is sufficient to remove spatial autocorrelation in the residuals of fitted models.

```
## Check spatial autocorrelation in model residuals
Sample_GM_variogram <- list()
Sample_SM_variogram <- list()
for (ii in 1:5) {

  SampleSediment <- sample_sedimentfractions(df_nested_SM = df_SM, df_nested_GM = df_GM)

SampleSediment_GM <- ranger(logGM ~ . , importance = "impurity",
                            data = SampleSediment$train_data_GM, num.trees = 2500)
SampleSediment_SM <- ranger(logSM ~ . , importance = "impurity",
                            data = SampleSediment$train_data_SM, num.trees = 2500)




}
```

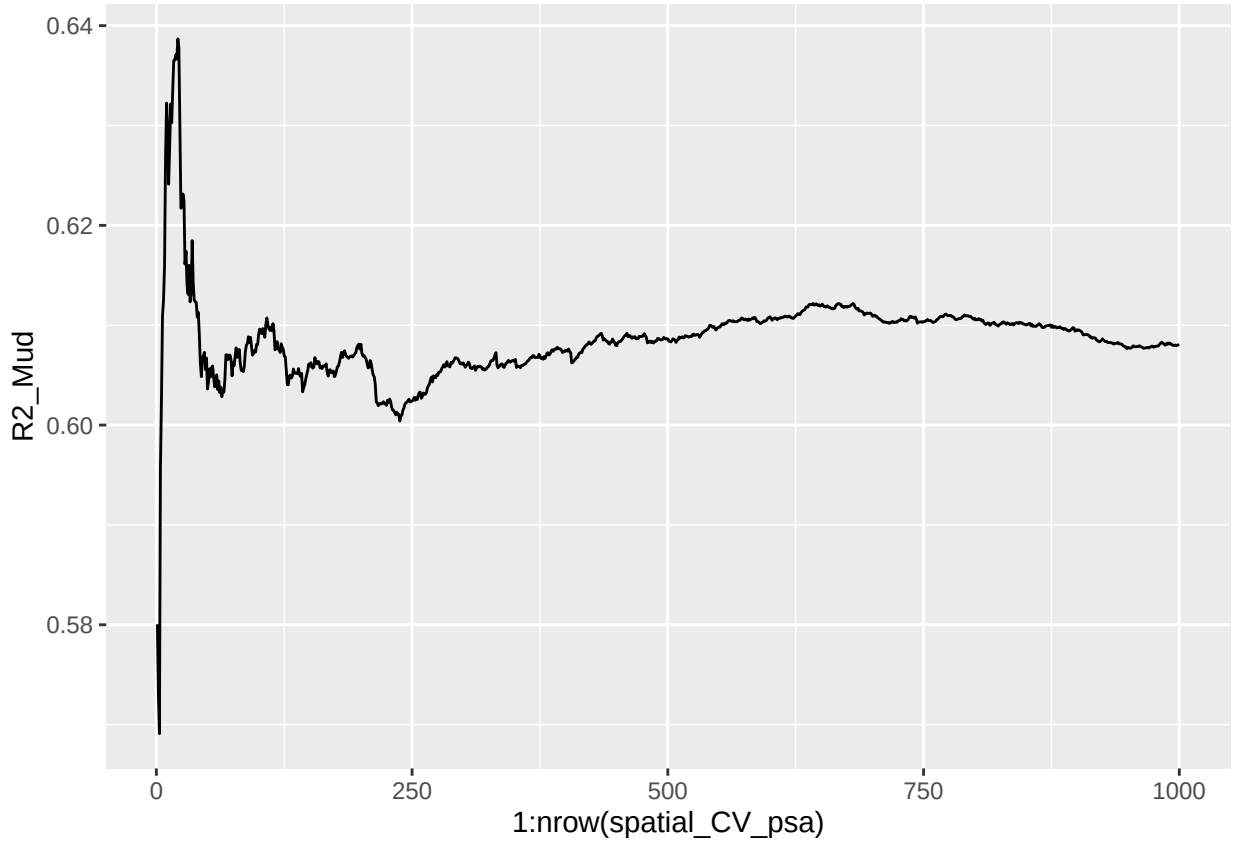
The following section is not run within this document due to the computation time but the code is provided as an illustration of the settings used in the cross validation process.

Results from an externally run cross-validation are instead loaded and plotted.

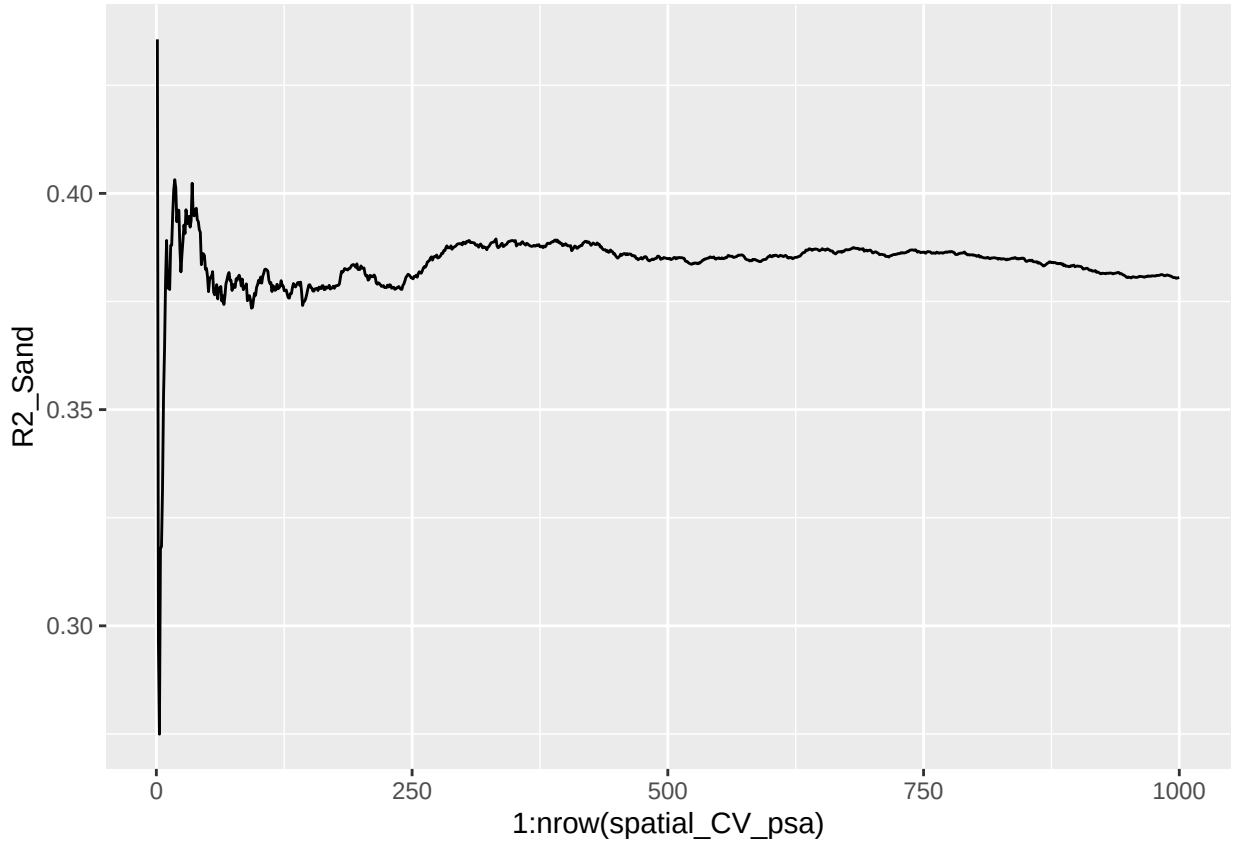
```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

## Spatial Crossvalidation of Random Forest
spatial_CV_psa <- spatial_logR_R2(df_nested_SM = df_SM,
                              df_nested_GM = df_GM,
                              reps = 1000, num.trees = 2500)
```

```
# ------------------------------------------------------------------------------------------#
# LOAD PREVIOUS RUN.
# ------------------------------------------------------------------------------------------#

spatial_CV_psa <- fread("SedimentFractions_SpatialCrossValidation_raw.csv")

## Running for 1000 iterations should help stabilise the model summaries.

spatial_CV_psa %>%
  mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  ggplot() +
```

```
  geom_line(aes(y = R2_Mud, x = 1:nrow(spatial_CV_psa)))
```




```
spatial_CV_psa %>%
    mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  ggplot() +
  geom_line(aes(y = R2_Sand, x = 1:nrow(spatial_CV_psa)))
```




```
spatial_CV_psa %>% mutate_all(function(x){
      cumsum(x)/row_number()}) %>% tail(.,1)
```

```
##     EucDist      MSE    aRMSE    R2_Mud   R2_Sand R2_Gravel
## 1: 30.30542 1496.586 22.16254 0.6080552 0.3805395 0.3206224
```

5.6 Independent Validation

The collection of additional Firth of Clyde sedimentology data in 2017 creates the opportunity to validate the generated seabed maps against independent data.

5.6.1 Import independent observations and interpolate predictions

```
## Import April & October 2017 data
Clyde_PSA <- fread(paste0(filepath, "Clyde_PSA_2017.csv"))

## Bilinear interpolation of predicted sediment data from seabed maps
Clyde_PSA$PredictedMud <- interpp(x = Clyde_IrregularGrid$Longitude,
                                  y = Clyde_IrregularGrid$Latitude,
                                  z = Clyde_IrregularGrid$Mud,
                                  xo = Clyde_PSA$Longitude,
                                  yo = Clyde_PSA$Latitude)$z
```

```
## Warning in interpp.old(x, y, z, xo, yo, ncp = 0, extrap = FALSE, duplicate =
## duplicate, : interpp.old() is deprecated, future versions will only provide
## interpp()
```

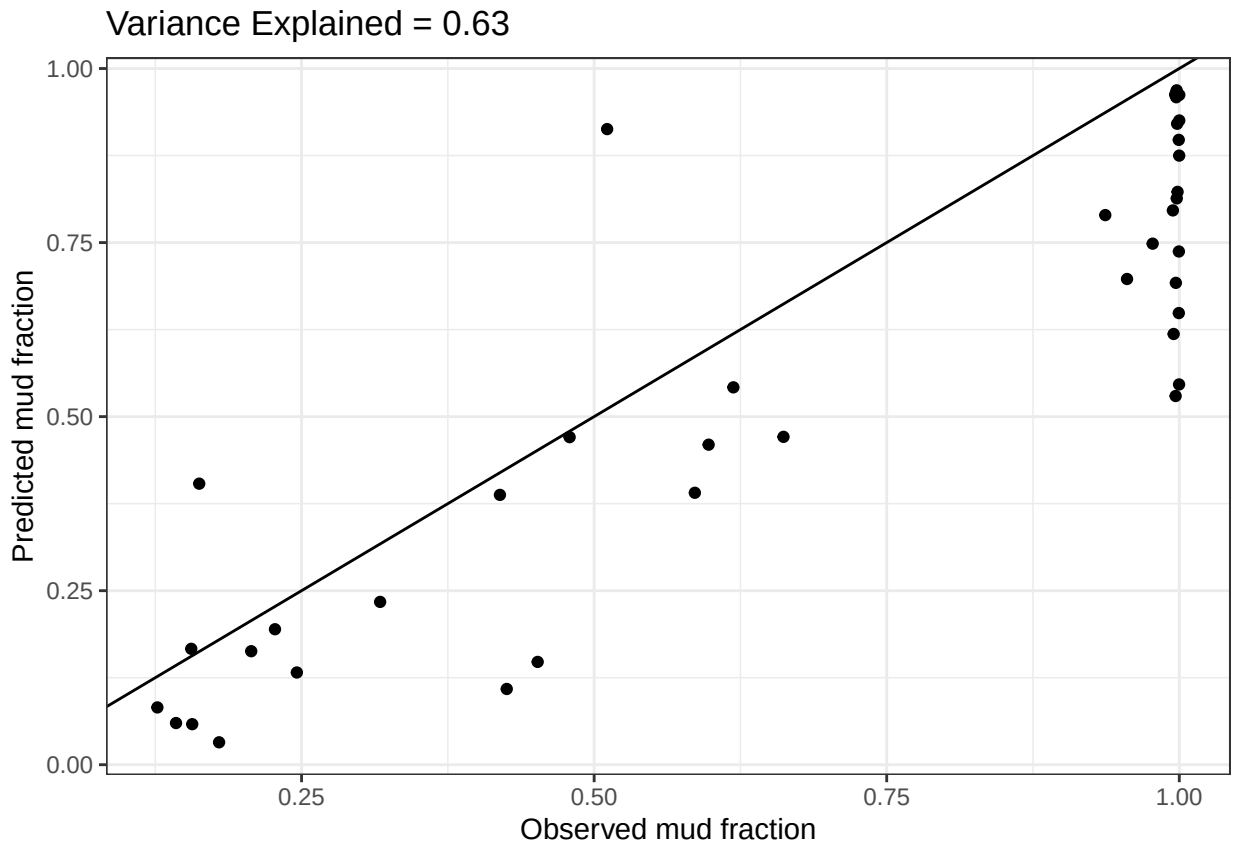
```
Clyde_PSA$PredictedSand <- interpp(x = Clyde_IrregularGrid$Longitude,
                                   y = Clyde_IrregularGrid$Latitude,
                                   z = Clyde_IrregularGrid$Sand,
                                   xo = Clyde_PSA$Longitude,
                                   yo = Clyde_PSA$Latitude)$z
```

```
## Warning in interpp.old(x, y, z, xo, yo, ncp = 0, extrap = FALSE, duplicate =
## duplicate, : interpp.old() is deprecated, future versions will only provide
## interpp()
```

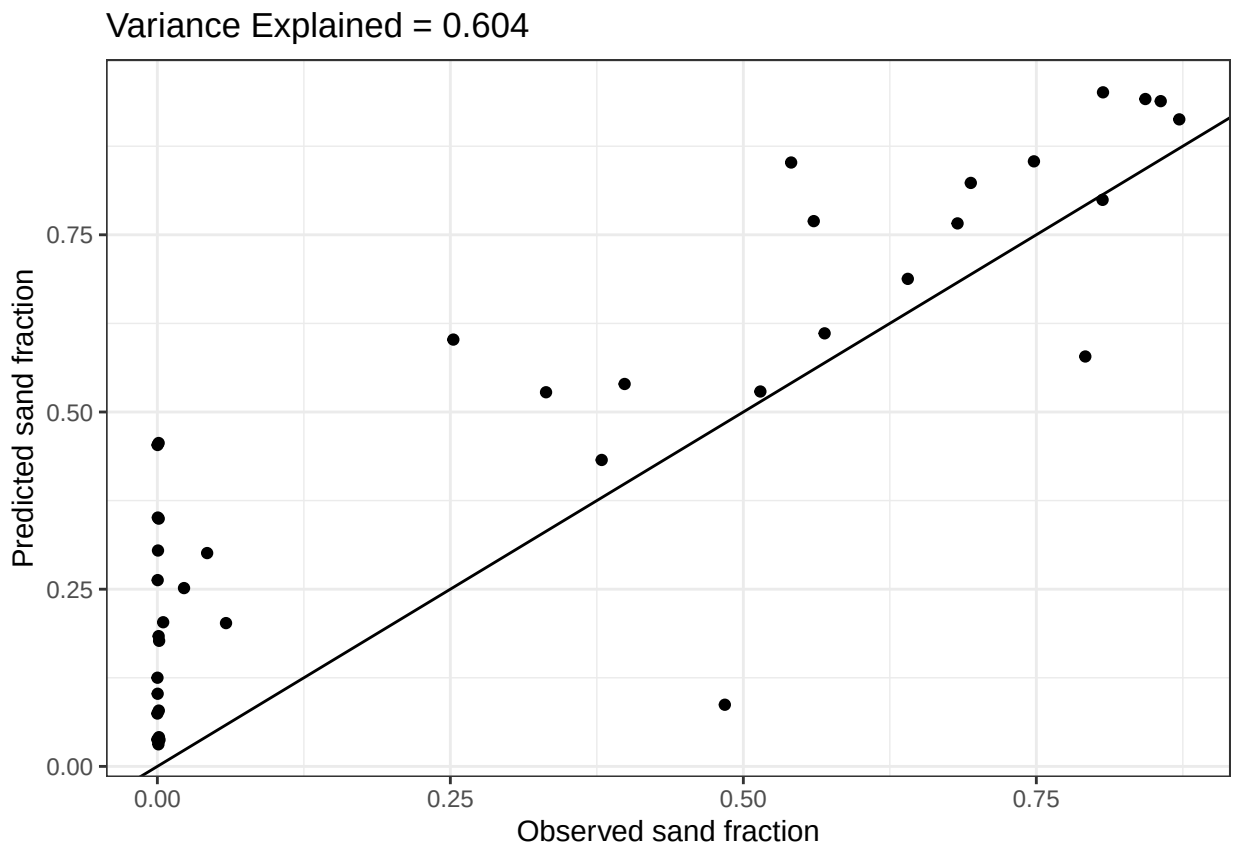
```
Clyde_PSA$PredictedGravel <- interpp(x = Clyde_IrregularGrid$Longitude,
                                     y = Clyde_IrregularGrid$Latitude,
                                     z = Clyde_IrregularGrid$Gravel,
                                     xo = Clyde_PSA$Longitude,
                                     yo = Clyde_PSA$Latitude)$z
```

```
## Warning in interpp.old(x, y, z, xo, yo, ncp = 0, extrap = FALSE, duplicate =
## duplicate, : interpp.old() is deprecated, future versions will only provide
## interpp()
```

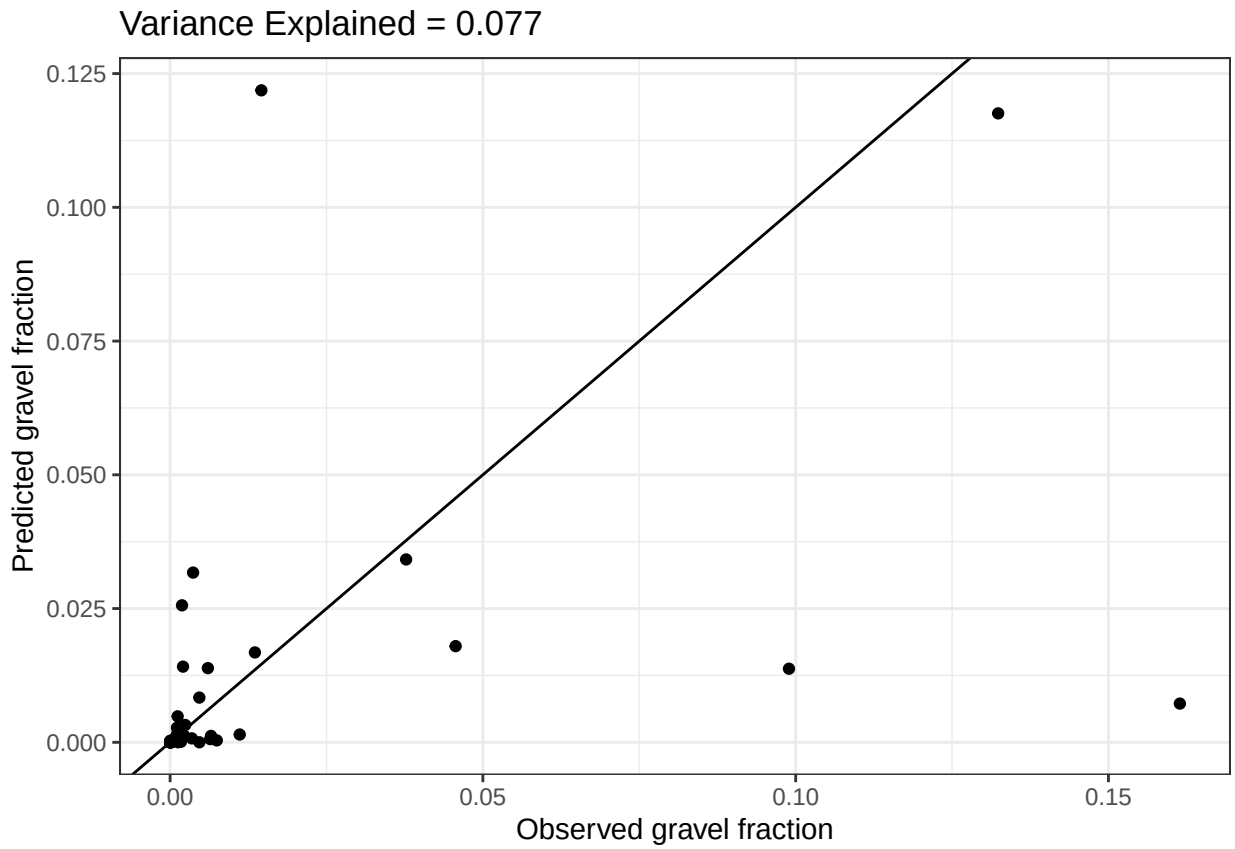
```
## Summarise by station
Clyde_PSA <- Clyde_PSA %>%
  mutate(Sample_ID = substr(Sample_ID, 1,3)) %>%
  select(Sample_ID, Gravel, Sand, Mud, PredictedGravel, PredictedSand, PredictedMud,
         Longitude, Latitude) %>%
  group_by(Sample_ID) %>%
  summarise_all(mean)

# In this next section, we calculate the variance in the observed data that is
# explained by the predicted map.

## Function to calculate variance explained
VE <- function(observed, predicted) {
  return(1 - (sum((observed - predicted)^2)/sum((observed - mean(observed))^2)))
}


## Variance explained by model (Total) n = 60
VE_mud <- VE(observed = Clyde_PSA$Mud, predicted = Clyde_PSA$PredictedMud)
VE_sand <- VE(observed = Clyde_PSA$Sand, predicted = Clyde_PSA$PredictedSand)
VE_gravel <- VE(observed = Clyde_PSA$Gravel, predicted = Clyde_PSA$PredictedGravel)

## Predicted Mud
Clyde_PSA %>%
  ggplot() +
  geom_point(aes(x = Mud, y = PredictedMud)) +
  geom_abline(slope =  1, intercept = 0) +
  scale_x_continuous("Observed mud fraction") +
  scale_y_continuous("Predicted mud fraction") +
  ggtitle(paste0("Variance Explained = ", round(VE_mud, digits = 3))) +
  theme_bw()
```




```
## Predicted Sand
Clyde_PSA %>%
  ggplot() +
  geom_point(aes(x = Sand, y = PredictedSand)) +
  geom_abline(slope =  1, intercept = 0) +
  scale_x_continuous("Observed sand fraction") +
  scale_y_continuous("Predicted sand fraction") +
  ggtitle(paste0("Variance Explained = ", round(VE_sand, digits = 3))) +
  theme_bw()
```

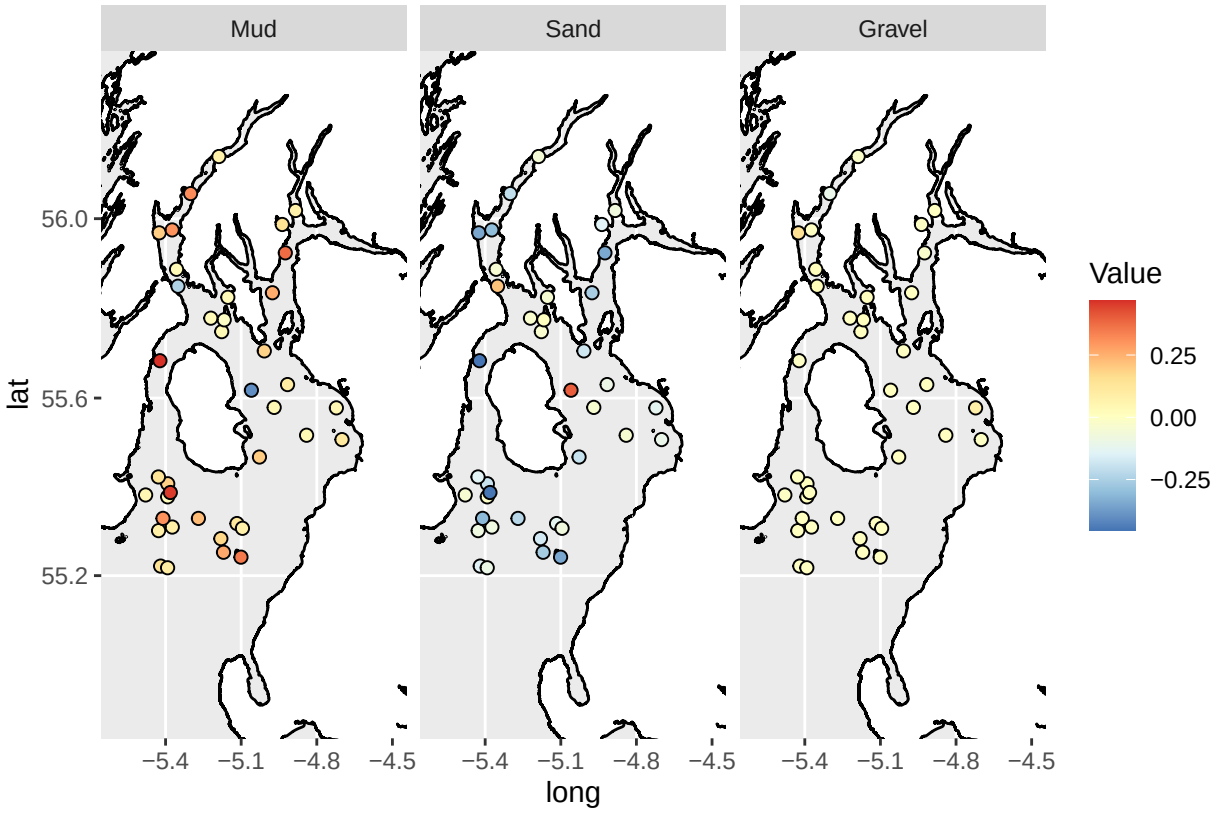



```
## Predicted Gravel
Clyde_PSA %>%
  ggplot() +
  geom_point(aes(x = Gravel, y = PredictedGravel)) +
  geom_abline(slope =  1, intercept = 0) +
  scale_x_continuous("Observed gravel fraction") +
  scale_y_continuous("Predicted gravel fraction") +
  ggtitle(paste0("Variance Explained = ", round(VE_gravel, digits = 3))) +
  theme_bw()
```



We also map the distribution of these residuals.

```
Clyde_PSA %>%
  mutate(residMud    = Mud - PredictedMud,
         residSand   = Sand - PredictedSand,
         residGravel = Gravel - PredictedGravel) %>%
  select(Longitude, Latitude, residMud, residSand, residGravel) %>%
  rename(Mud = residMud,
         Sand = residSand,
         Gravel = residGravel) %>%
  gather(Variable, Value, -Longitude, -Latitude) %>%
  mutate(Variable = factor(Variable,levels = c("Mud", "Sand", "Gravel"))) %>%
  ggplot() +
  geom_polygon(aes(x = long, y = lat, group = group), colour = "black", fill = "white",
               data = Shape_coast_f) +
  geom_point(aes(x = Longitude, y = Latitude, fill = Value), shape =21, size = 2) +
  scale_fill_distiller(palette = "RdYlBu") +
  facet_wrap(~Variable) +
  coord_map(xlim = c(-5.6,-4.5),ylim = c(54.9,56.3))
```



6 Median grain size

In this section, the whole sediment median grain size is predicted from data on fractions of mud, sand and gravel.

Predictions are carried out on the basis of a model fitted to Clyde data and MSS/CEFAS North Sea data for mud, sand and gravel. I then predict directly over a grid of each sediment fraction so any uncertainty is consistent between the datasets. For brevity, selection of the optimal model is not shown here.

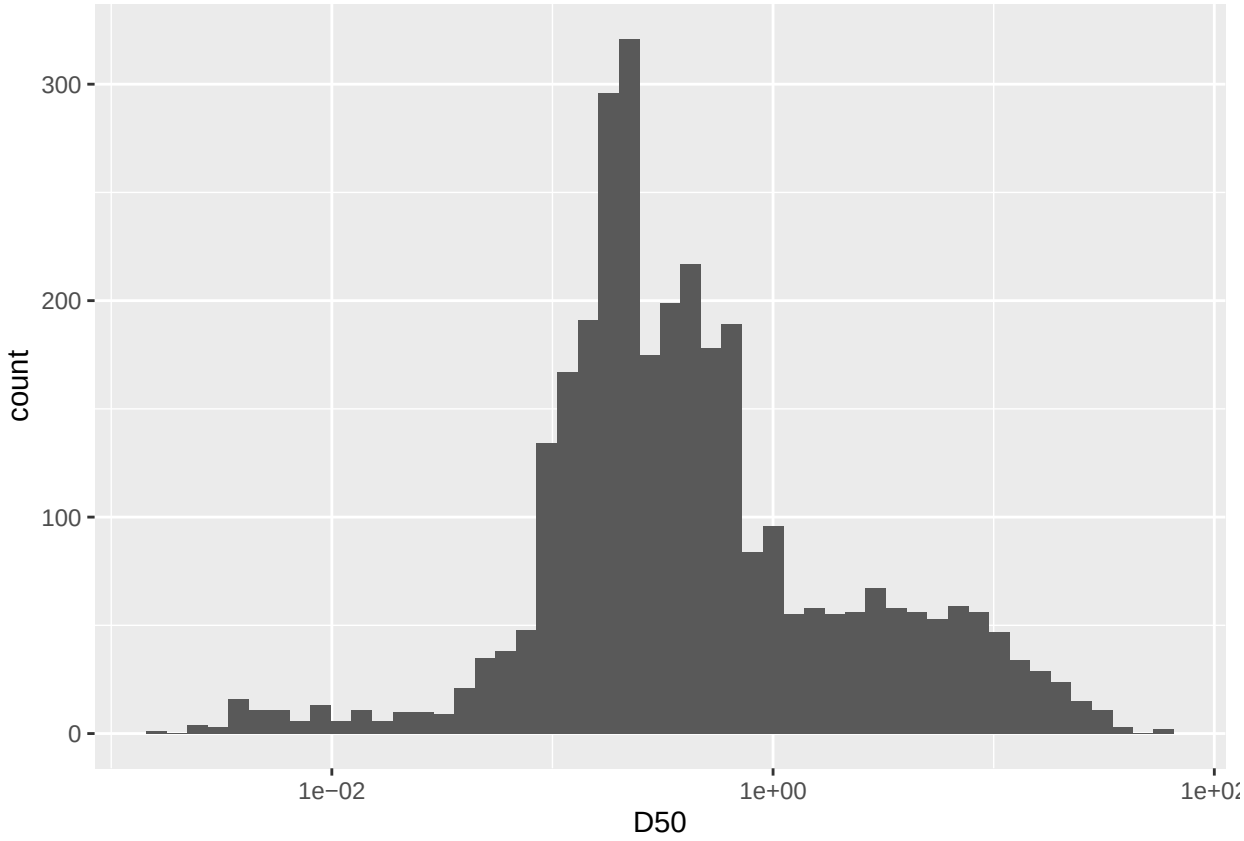
6.1 Import training data

```
D50_MSS_CEFAS <- fread("data/D50_MSS_CEFAS_NorthSea.csv") %>%  filter(complete.cases(.))
D50_MSS_Clyde <- fread("data/D50_MSS_Clyde.csv")
D50_PSA       <- fread(paste0(filepath, "Clyde_D50_2017.csv"))
```

6.2 Generate predictive models

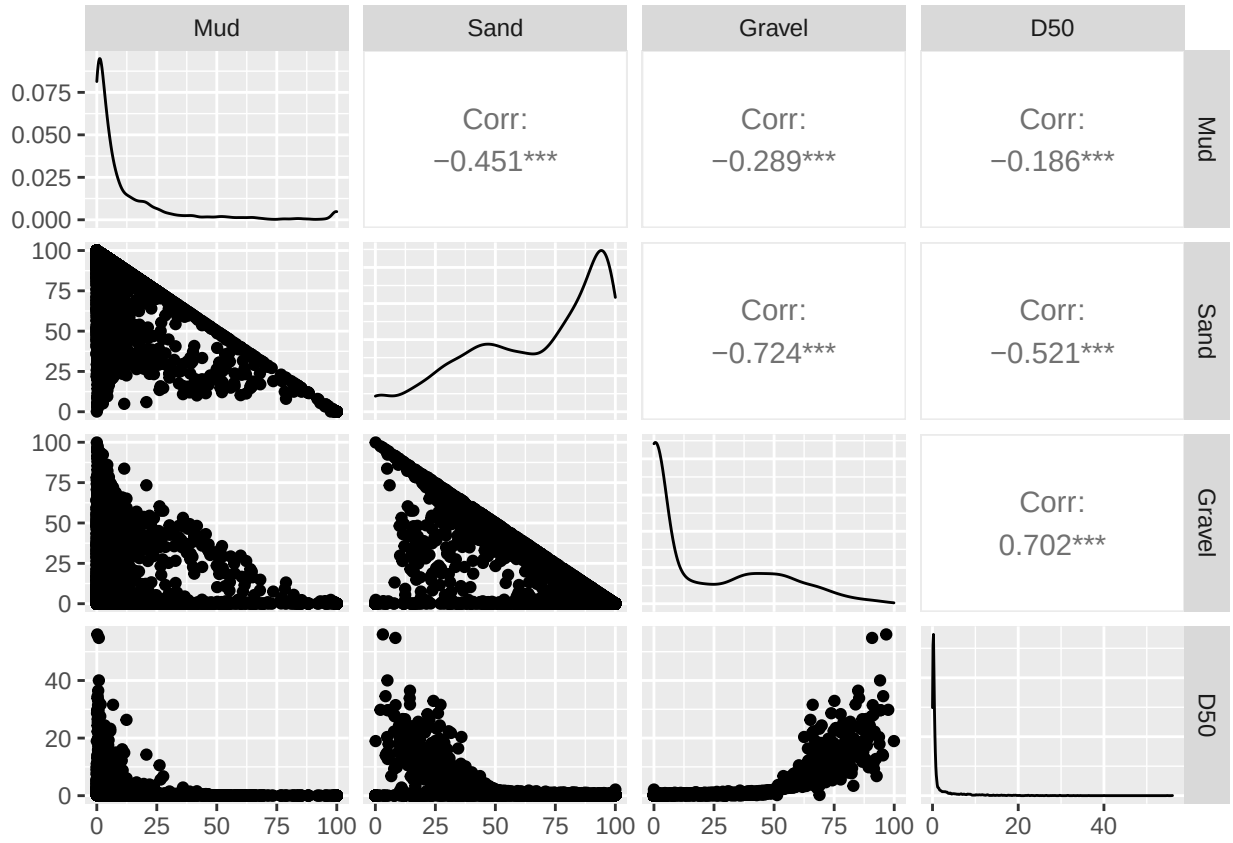
```
D50_NS_PSA <- D50_MSS_CEFAS %>%
  select(Mud, Sand, Gravel, TotalD50) %>%
  rename(D50 = TotalD50) %>%
  bind_rows(D50_PSA %>%
              select(Mud, Sand, Gravel, D50))

## Check distribution of particle sizes
D50_NS_PSA %>% ggplot() +
```

```
  geom_histogram(aes(x = D50), bins = 50) +
  scale_x_continuous(trans = "log10")
```




```
## Check whether collinearity?
D50_NS_PSA %>% GGally::ggpairs()
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

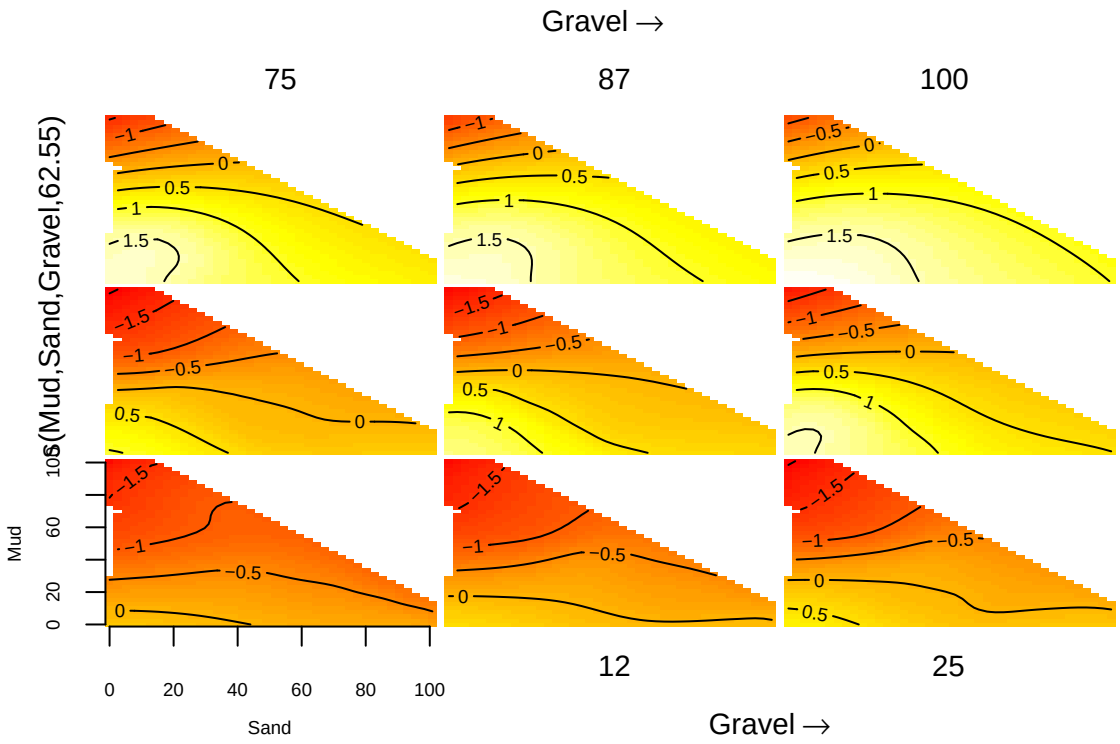



```
## GAM for whole Sediment D50 (mm)
GAMmod <- gam(log10(D50) ~ s(Mud, Sand, Gravel), data = D50_NS_PSA)

summary(GAMmod) ## 96.2% deviance explained. looks good!
```

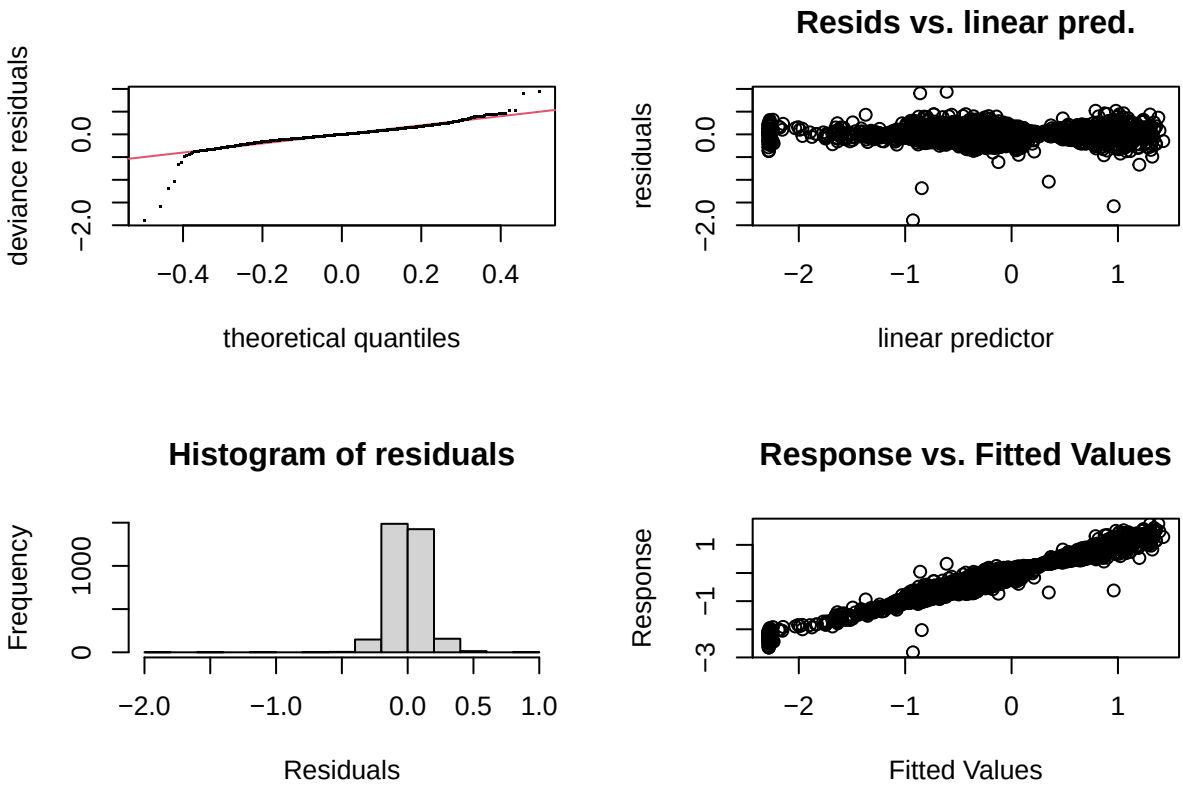
```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## log10(D50) ~ s(Mud, Sand, Gravel)
##
## Parametric coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.359891   0.002419  -148.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                       edf Ref.df    F p-value
## s(Mud,Sand,Gravel) 62.55  78.07 1034  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Rank: 106/110
## R-sq.(adj) =  0.961   Deviance explained = 96.2%
```

```
## GCV = 0.01936  Scale est. = 0.018981  n = 3244
```

```
plot(GAMmod)
```




```
par(mfrow = c(2,2)); gam.check(GAMmod); par(mfrow = c(1,1))
```




```
##
## Method: GCV   Optimizer: magic
## Smoothing parameter selection converged after 8 iterations.
## The RMS GCV score gradient at convergence was 2.242984e-08 .
## The Hessian was positive definite.
## Model rank =  106 / 110
##
## Basis dimension (k) checking results. Low p-value (k-index<1) may
## indicate that k is too low, especially if edf is close to k'.
##
##                         k'   edf k-index p-value
## s(Mud,Sand,Gravel) 109.0  62.5    0.97    0.15
```

```
concurvity(GAMmod)
```

```
##                para s(Mud,Sand,Gravel)
## worst    0.03421647       4.238960e-02
## observed 0.03421647       5.825602e-28
## estimate 0.03421647       2.304080e-29
```

6.3 Predict Clyde sediment median grain size

```
Clyde_IrregularGrid$D50 <- 10^(predict(GAMmod,
                                       newdata = Clyde_IrregularGrid %>%
                                         mutate(Mud    = Mud * 100,
                                                Sand   = Sand * 100,
```

```
                                                Gravel = Gravel * 100),
                                        type = "response"))
```

7 Permeability and porosity

This section encompasses the fitting of permeability and porosity relationships to measures of sediment particle size, and the prediction of large-scale permeability and porosity patterns in the Clyde from these fitted relationships

```
library(RColorBrewer)
library(lme4)
```

```
## Warning: package 'lme4' was built under R version 4.0.4
```

7.1 Define error functions

Here, we define objective functions to minimise the residual square error between function fit and observed data

```
## Define Function to calculate Residual Sums of Squares
minRSS_perm <- function(params, data) {

  ## Extract parameters and variables
  a <- params["a"]
  b <- params["b"]
  c <- params["c"]

  Permeability <- log10(data$Permeability)
  D50 <- data$D50

  ## Statistical Relationship
  PredictedPerm <- log10((10^a)*(D50^b) + c)

  ## Residual Sum of Squares
  return(sum((PredictedPerm - Permeability)^2))

}

## Define Function to calculate Residual Sums of Squares
minRSS_por <- function(params, data) {

  ## Extract parameters and variables
  p1 <- params["p1"]
  p2 <- params["p2"]
  p3 <- params["p3"]
  p4 <- params["p4"]

  Porosity <- log10(data$Porosity)
  D50 <- data$D50

  ## Statistical Relationship
  PredictedPor <- p1 + p2*(1/(1 + exp(-(log10(D50) - p3)/p4)))

  ## Residual Sum of Squares
```

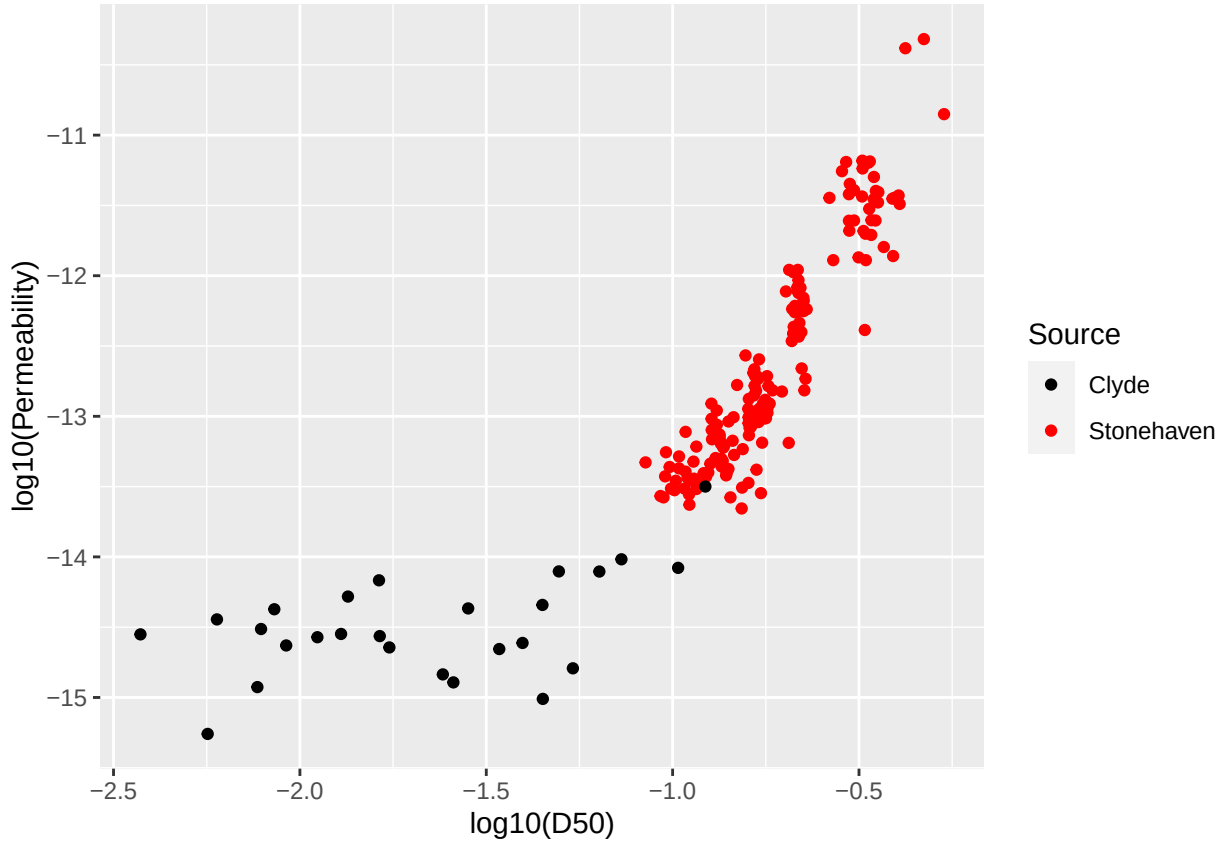
```
  return(sum((PredictedPor - Porosity)^2))

}
```

7.2 LOad data

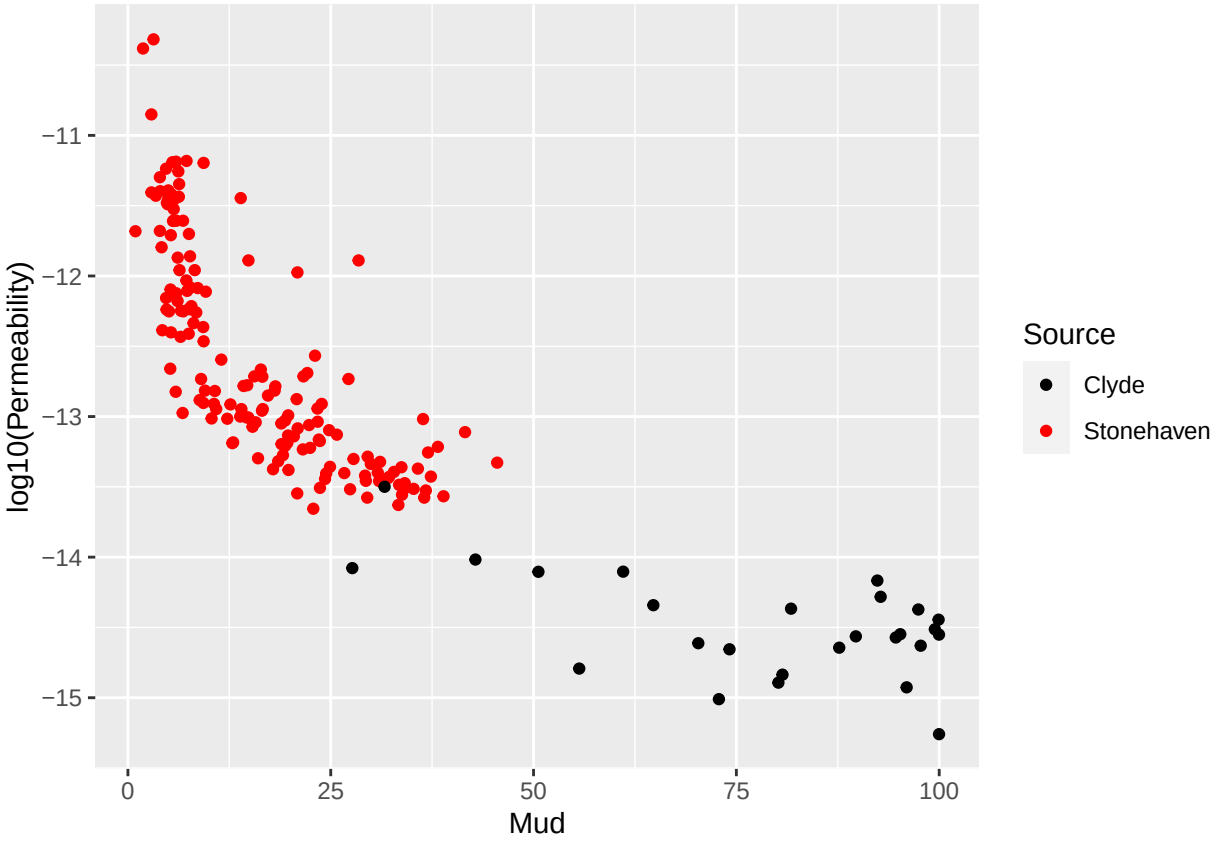
Load data and run some exploratory plots.

Note - some data pre-processing (merging station locations and measurements) is not shown here for clarity.

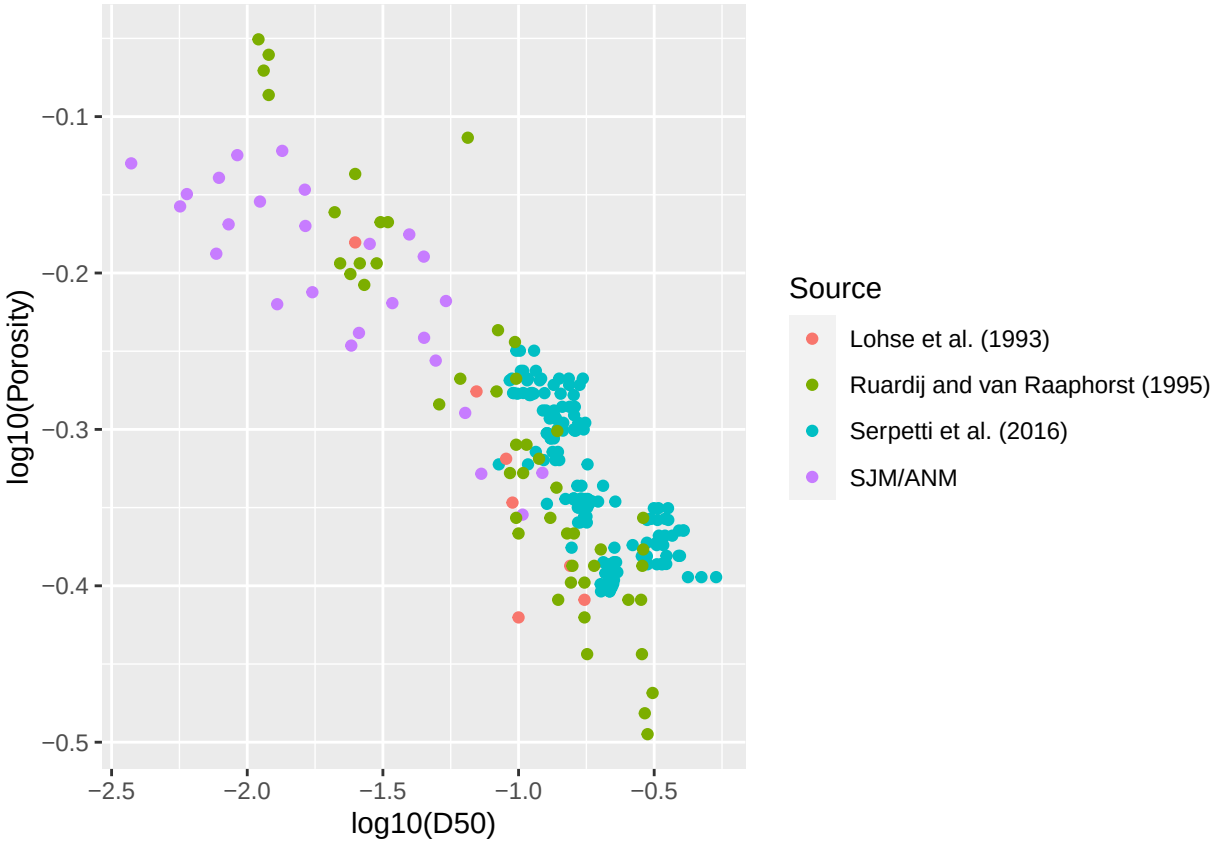
```
Permeability_data <- fread(paste0(filepath,"Permeability_data.csv"))
Porosity_data     <- fread(paste0(filepath,"Porosity_data.csv"))

## Combined Dataset - Log10(Permeability) & Log10(D50)
ggplot() +
  geom_point(aes(x = log10(D50), y = log10(Permeability), colour = Source),
             data = Permeability_data) +
  scale_colour_manual(values = c("black","red"))
```




```
## Combined Dataset - Log10(Permeability) & Mud
ggplot() +
  geom_point(aes(x = Mud, y = log10(Permeability), colour = Source),
             data = Permeability_data) +
  scale_colour_manual(values = c("black","red"))
```

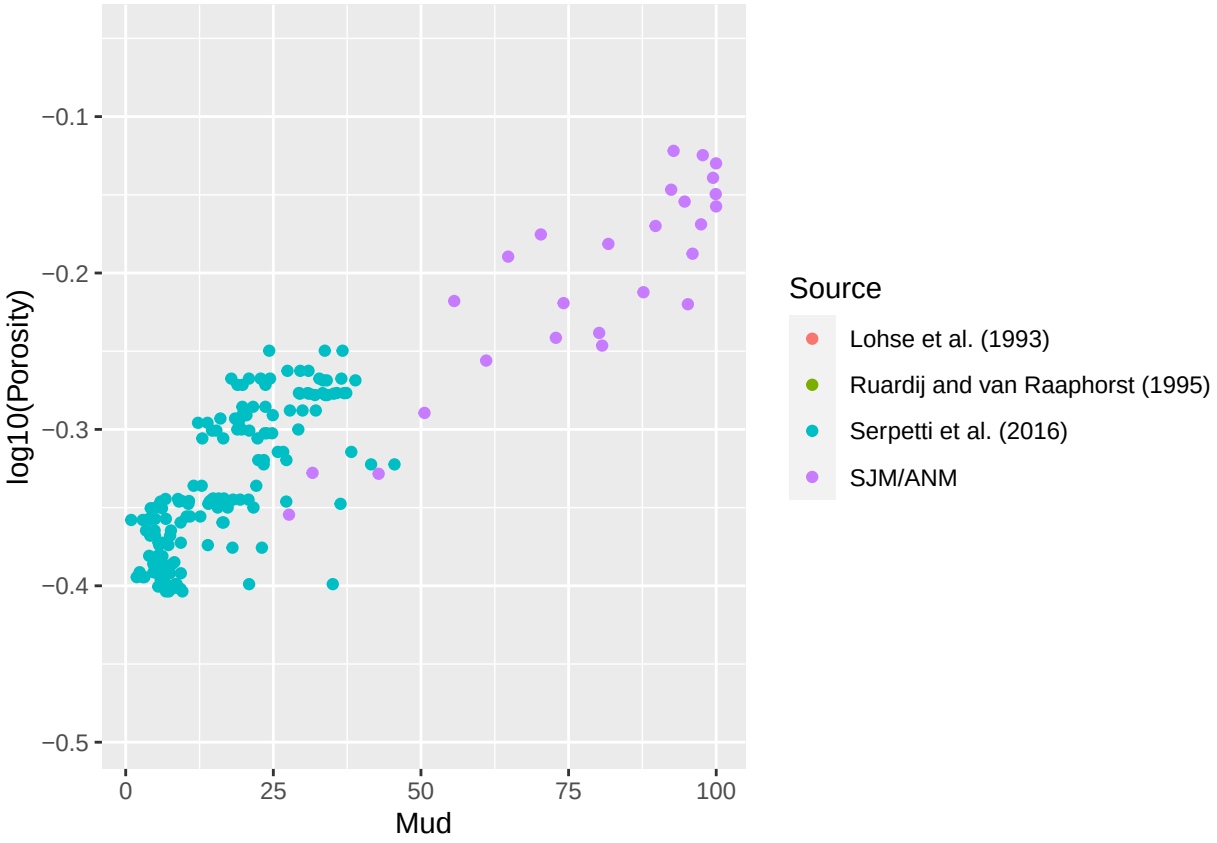



```
## Combined Dataset - Log10(Porosity) & log10(D50)
ggplot() +
  geom_point(aes(x = log10(D50), y = log10(Porosity), colour = Source),
             data = Porosity_data)
```




```
## Combined Dataset - Log10(Porosity) & Mud
ggplot() +
  geom_point(aes(x = Mud, y = log10(Porosity), colour = Source),
             data = Porosity_data)
```

```
## Warning: Removed 56 rows containing missing values (geom_point).
```



7.3 Fit statistical relationships

7.3.1 Permeability relationships

```
## Power Function + y-intercept
params <- c(a = -9.213, b = 4.615, c = 0)
suppressMessages(optimPar <- optim(par = params, fn = minRSS_perm,
                                   data = Permeability_data,
                                   method = "Nelder-Mead"))
```

7.3.1.1 Median grain size

```
## Warning in fn(par, ...): NaNs produced

## Warning in fn(par, ...): NaNs produced

## Warning in fn(par, ...): NaNs produced

## Warning in fn(par, ...): NaNs produced

## Warning in fn(par, ...): NaNs produced

## Warning in fn(par, ...): NaNs produced

## Warning in fn(par, ...): NaNs produced
```

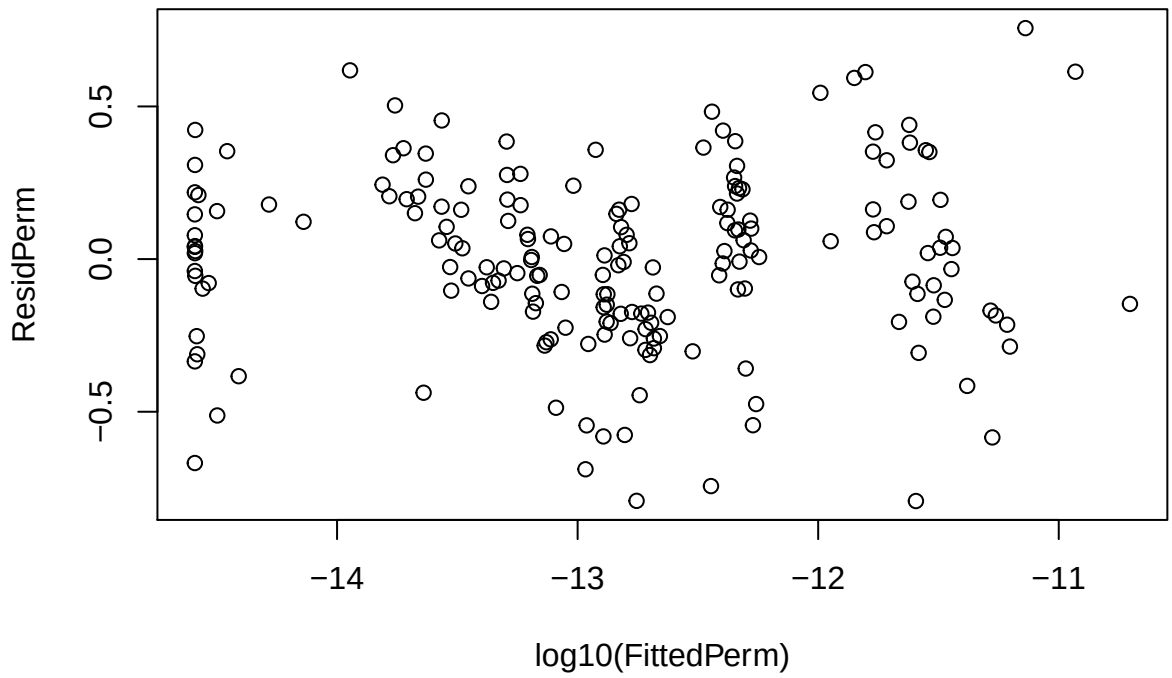
```
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
```

```
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
## Warning in fn(par, ...): NaNs produced
```

```
print(optimPar)
```

```
## $par
##             a             b             c
## -9.565872e+00  4.187677e+00  2.563446e-15
##
## $value
## [1] 15.36483
##
## $counts
## function gradient
##      354       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
## Check distribution of residuals
FittedPerm <- (10^optimPar$par[1])*(Permeability_data$D50^optimPar$par[2]) + optimPar$par[3]
ResidPerm <- log10(Permeability_data$Permeability) - log10(FittedPerm)

plot(x = log10(FittedPerm), y = ResidPerm)
```




```
## R2 for fitted relationships
caret::R2(pred = log10(FittedPerm),
```

```
          obs  = log10(Permeability_data$Permeability),
          formula = "traditional")
```

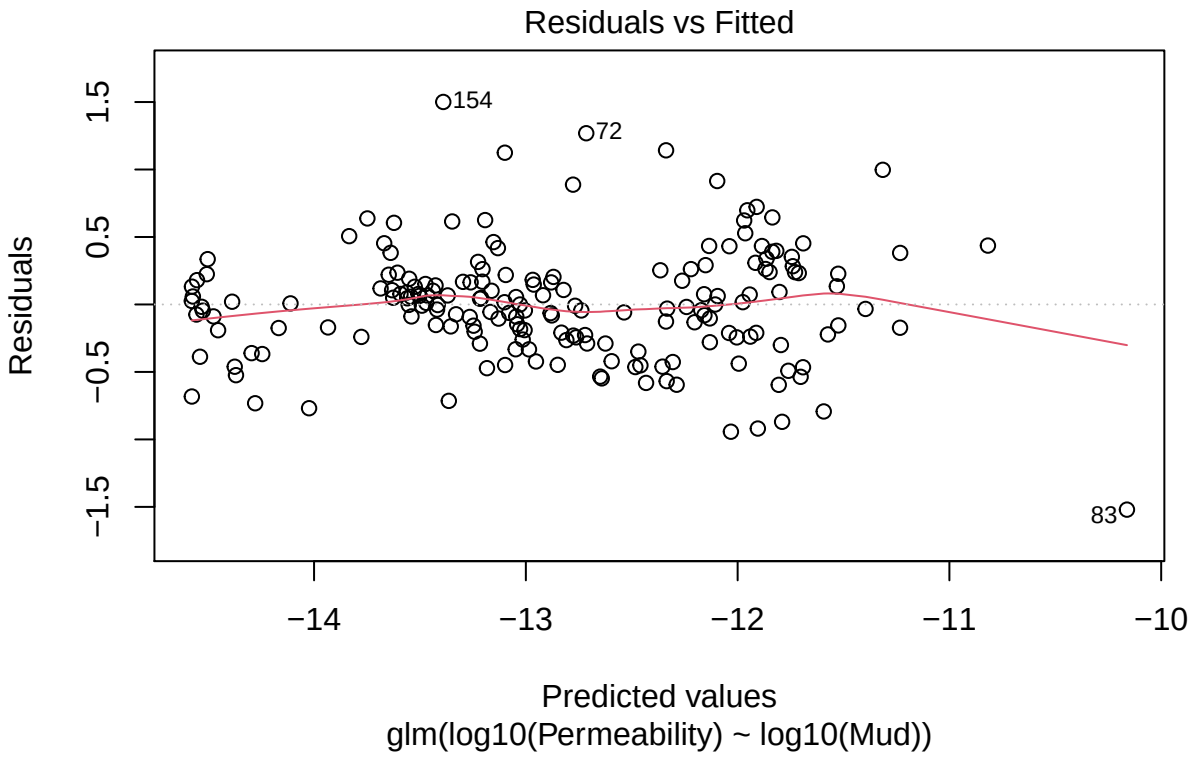
```
## [1] 0.9150123
```

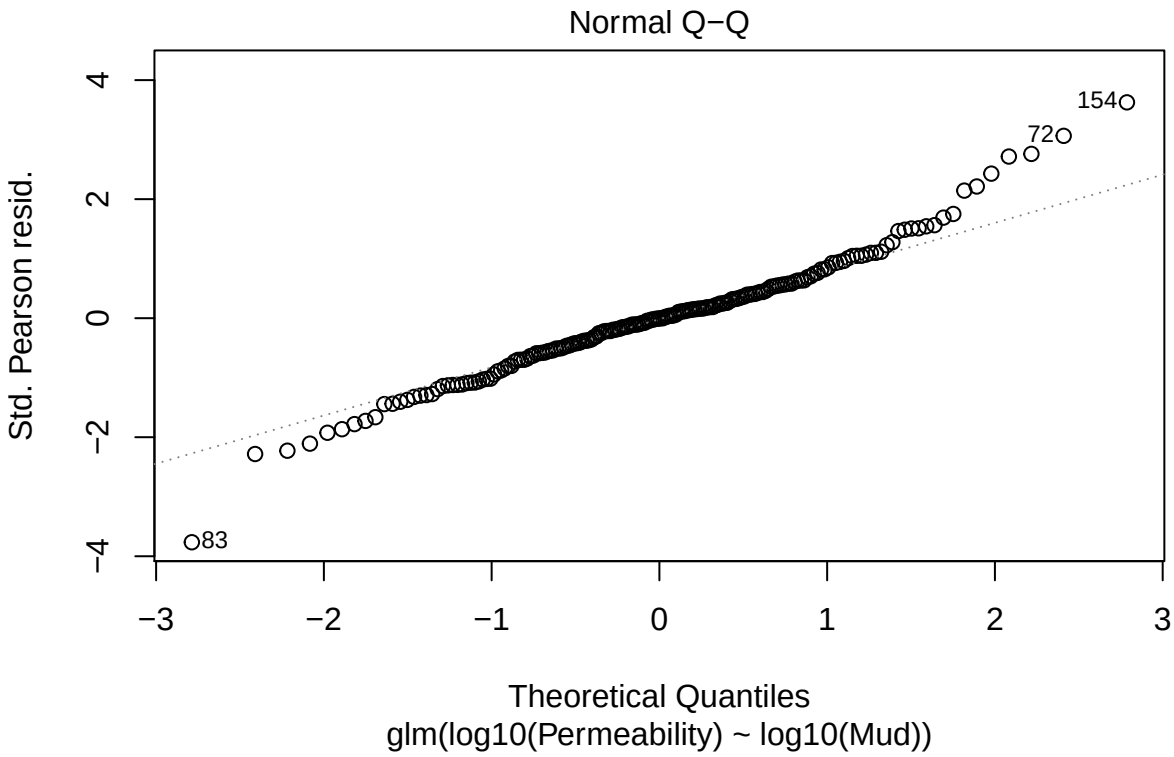
```
## Fit log-log relationship
Mod1 <- Permeability_data %>%
  glm(log10(Permeability) ~ log10(Mud),data = ., family = gaussian)

summary(Mod1)
```

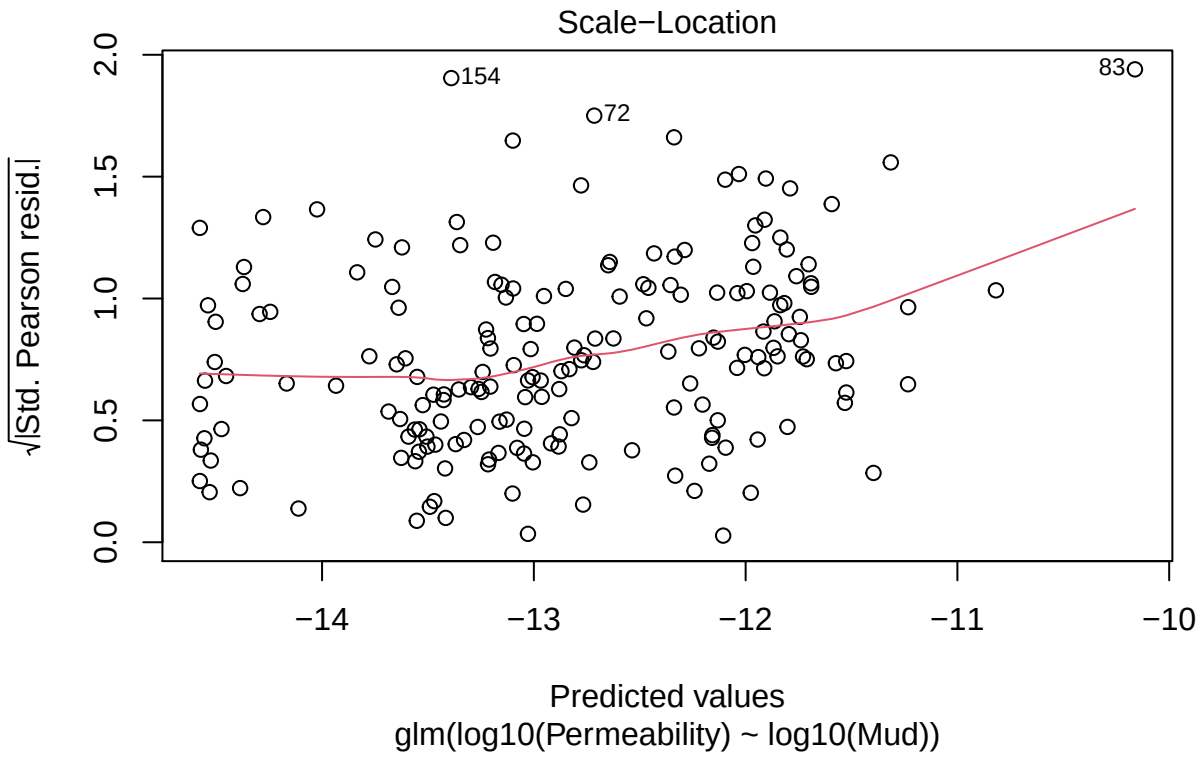
7.3.1.2 % Mud content

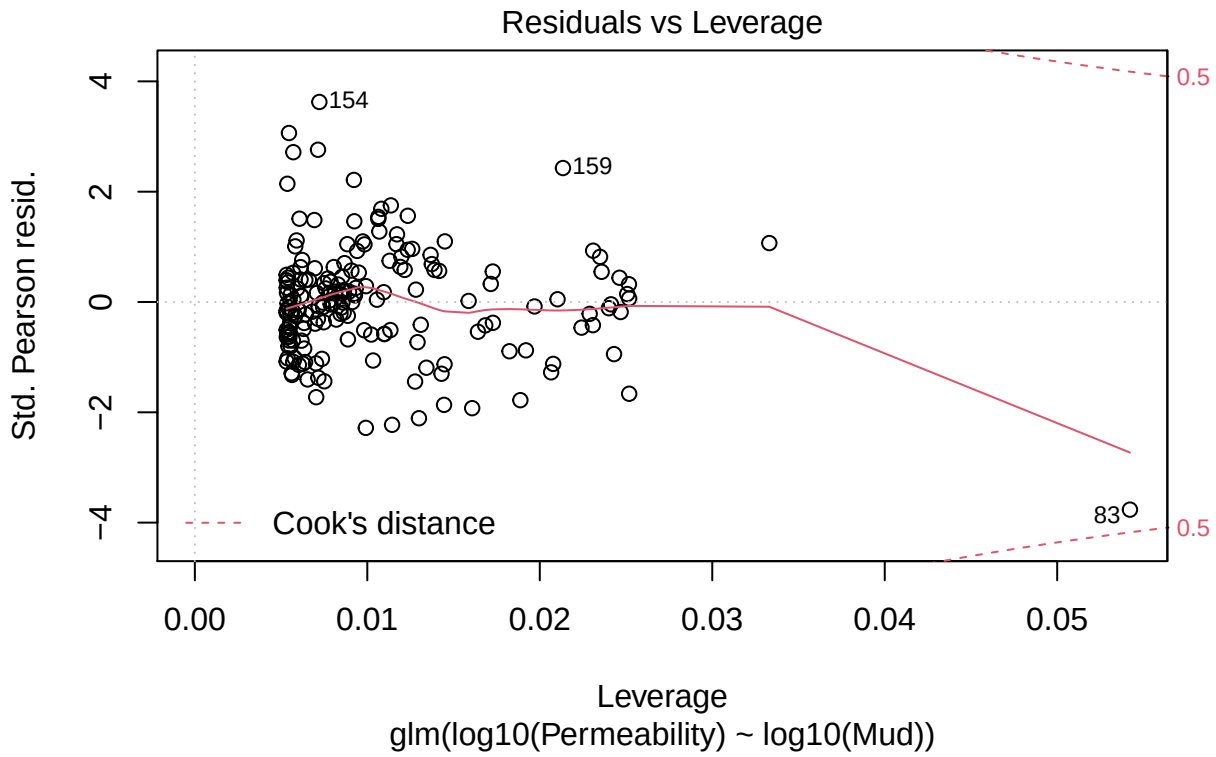
```
##
## Call:
## glm(formula = log10(Permeability) ~ log10(Mud), family = gaussian,
##     data = .)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.52027  -0.23266  -0.00185   0.21902   1.50048
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -10.22917    0.09451 -108.23   <2e-16 ***
## log10(Mud)   -2.17403    0.07403  -29.37   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1724332)
##
##     Null deviance: 180.789  on 187  degrees of freedom
## Residual deviance:  32.073  on 186  degrees of freedom
## AIC: 207.05
##
## Number of Fisher Scoring iterations: 2
```

```
plot(Mod1)
```


Residuals vs Fitted
Residuals
1.5
0.5
−0.5
−1.5
154
72
83
−14
−13
−12
−11
−10
Predicted values
glm(log10(Permeability) ~ log10(Mud))


Normal Q−Q
Std. Pearson resid.
Theoretical Quantiles
glm(log10(Permeability) ~ log10(Mud))
154
72
83
−4
−2
0
2
4
−3
−2
−1
0
1
2
3


Scale−Location
√|Std. Pearson resid.|
0.0
0.5
1.0
1.5
2.0
154
72
83
−14
−13
−12
−11
−10
Predicted values
glm(log10(Permeability) ~ log10(Mud))




```
#' Diagnostics look pretty good. Residuals are random and centred around zero.
#' There is one outlier but not really a problem. The fitted relationship is:
#'
#' $$Permeability = -2.17403 10^{Mud} - 10.22917$$

## R2 for fitted relationships
caret::R2(obs  = log10(Permeability_data$Permeability),
          pred = (log10(Permeability_data$Mud)*-2.17403 - 10.22917),
          formula = "traditional")
```

```
## [1] 0.8225965
```

7.3.2 Porosity relationships

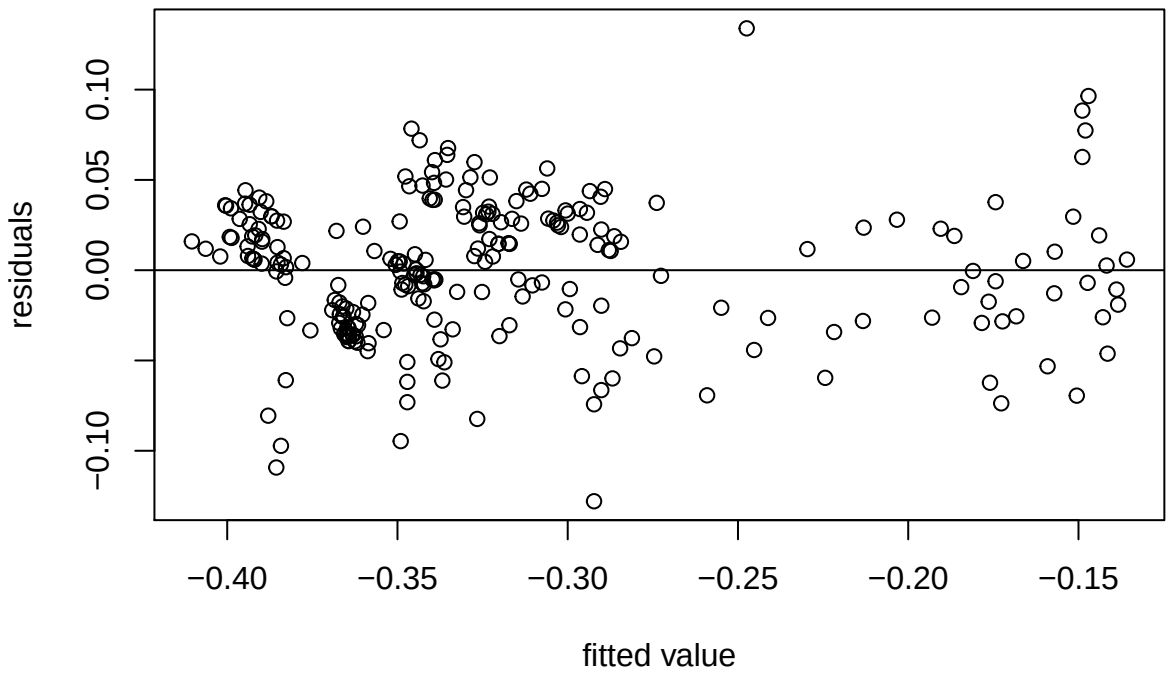
```
## Fit parameters
params <- c(p1 = -0.436,
            p2 = 0.366,
            p3 = -1.227,
            p4 = -0.270)

optim(par = params, fn = minRSS_por, data = Porosity_data, method = "Nelder-Mead")
```

7.3.2.1 Median grain size

```
## $par
##         p1         p2         p3         p4
## -0.4338169  0.3013978 -1.0376464 -0.3096143
```

```
##
## $value
## [1] 0.3567365
##
## $counts
## function gradient
##      229       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
#' When starting values: p1 = -0.436, p2 = 0.366, p3 = -1.227, p4 = -0.270;
#' fitted RSS = 0.3567365.
#'
#'         p1         p2         p3         p4
#' -0.4338169  0.3013978 -1.0376464 -0.3096143

## fitted relationship
p1 <- -0.4338169
p2 <- 0.3013978
p3 <- -1.0376464
p4 <- -0.3096143

## Check plot of residuals against fitted value
FittedPor <- (p1 + p2*(1/(1 + exp(-(log10(Porosity_data$D50) - p3)/p4))))
ResidPor  <- log10(Porosity_data$Porosity) - FittedPor
plot(y = ResidPor,
     x = FittedPor,
     xlab = "fitted value", ylab = "residuals")
abline(h = 0)
```




```
## R2 for fitted relationships
caret::R2(obs  = log10(Porosity_data$Porosity),
          pred = FittedPor,
          formula = "traditional")
```

```
## [1] 0.776003
```

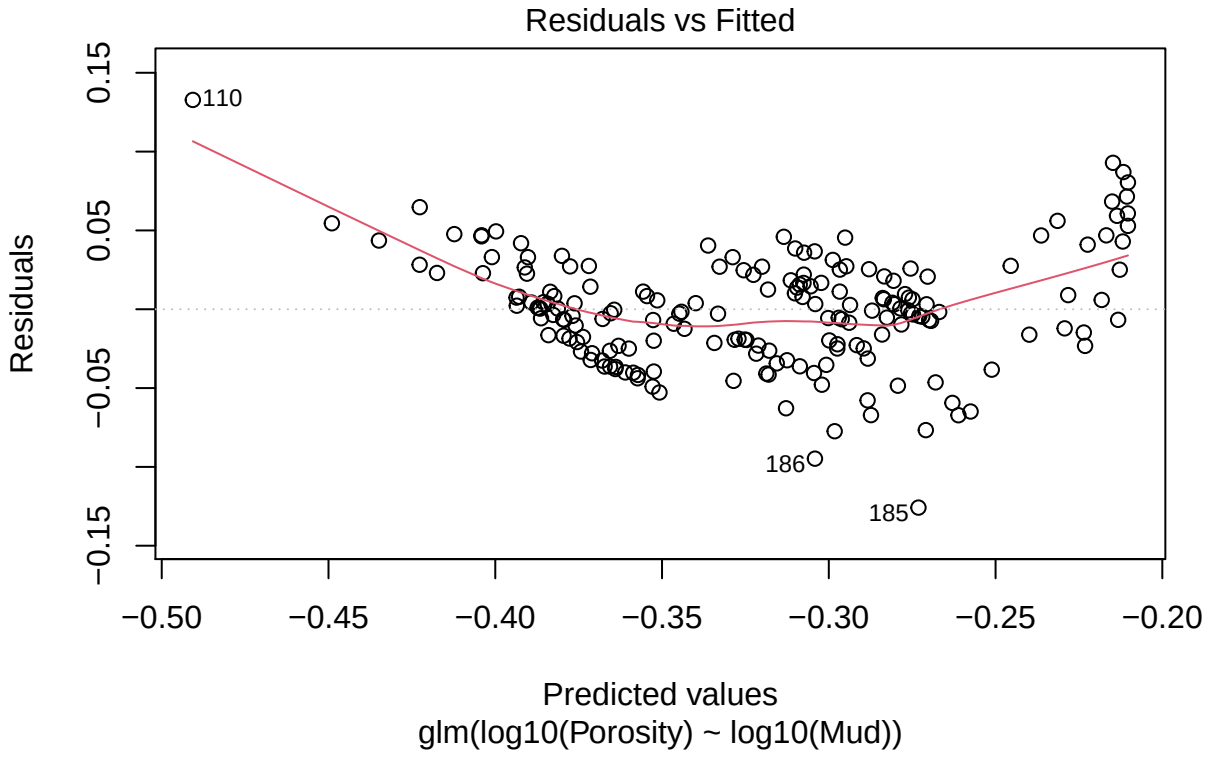
```
## Fit log-log relationship
Mod2 <- Porosity_data %>%
  glm(log10(Porosity) ~ log10(Mud),data = ., family = gaussian)

summary(Mod2)
```

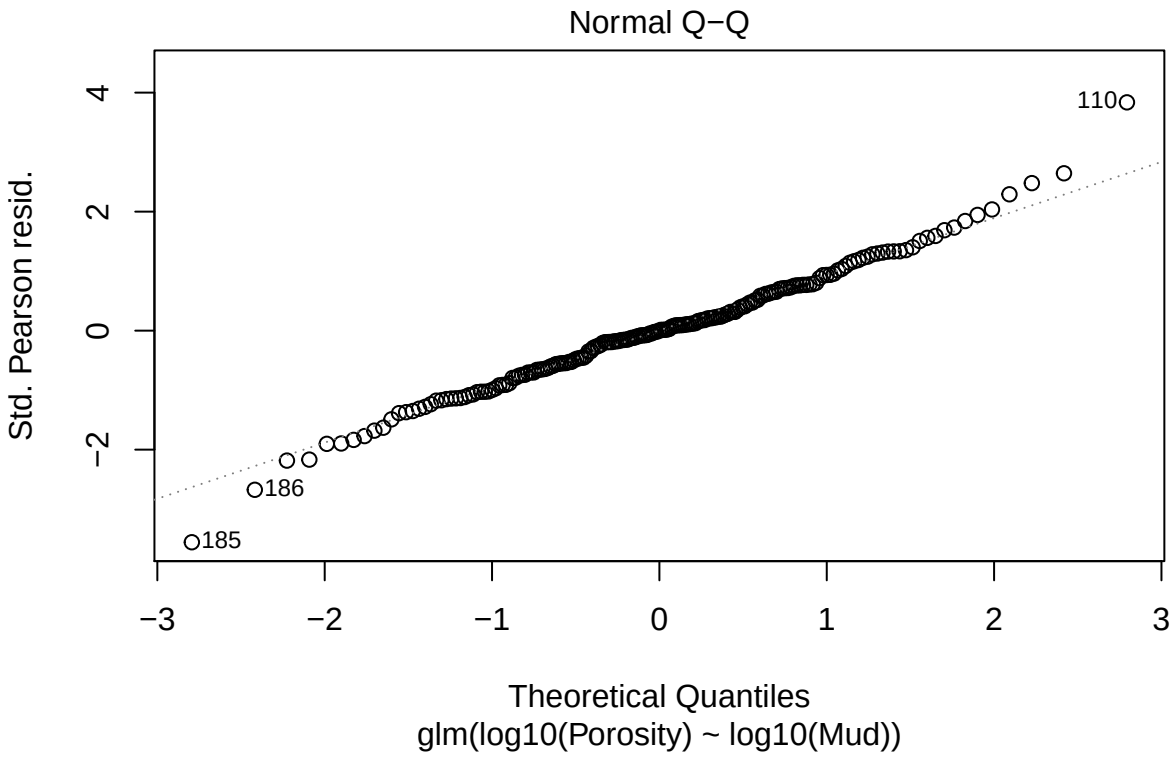
7.3.2.2 % Mud content

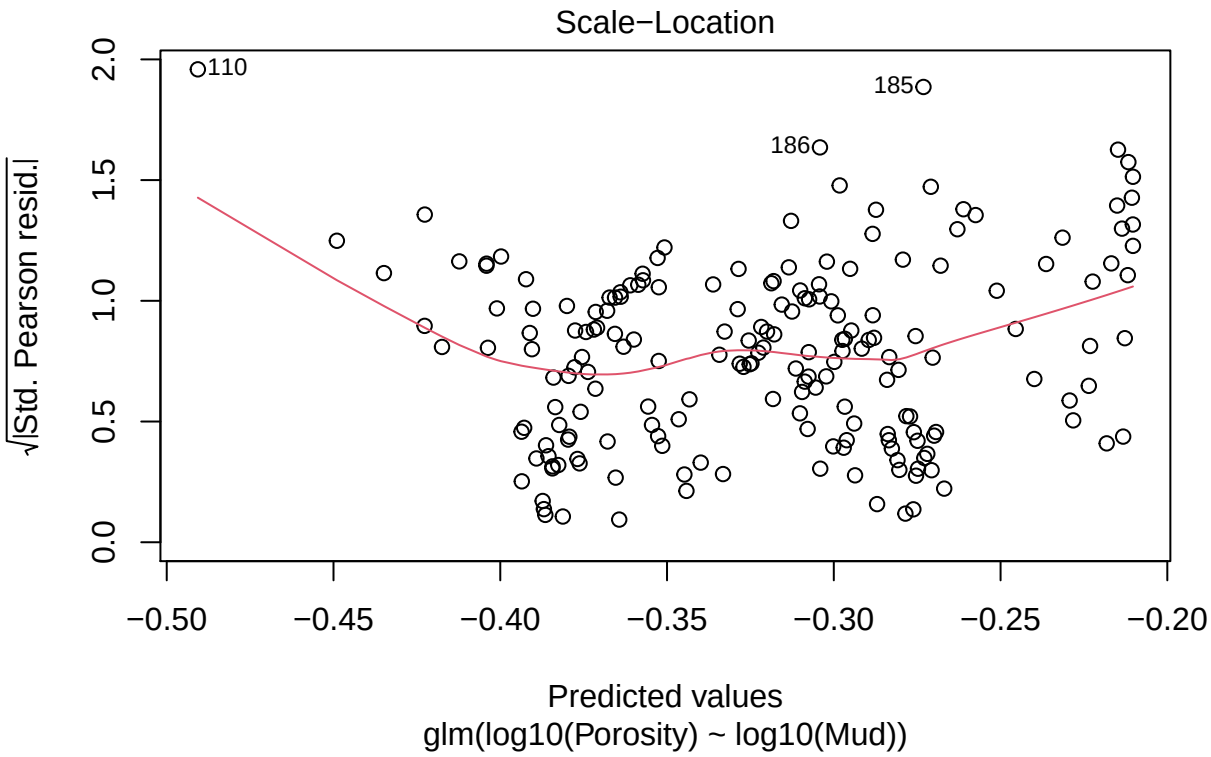
```
##
## Call:
## glm(formula = log10(Porosity) ~ log10(Mud), family = gaussian,
##     data = .)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.125815  -0.022360   0.000045   0.022641   0.132775
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

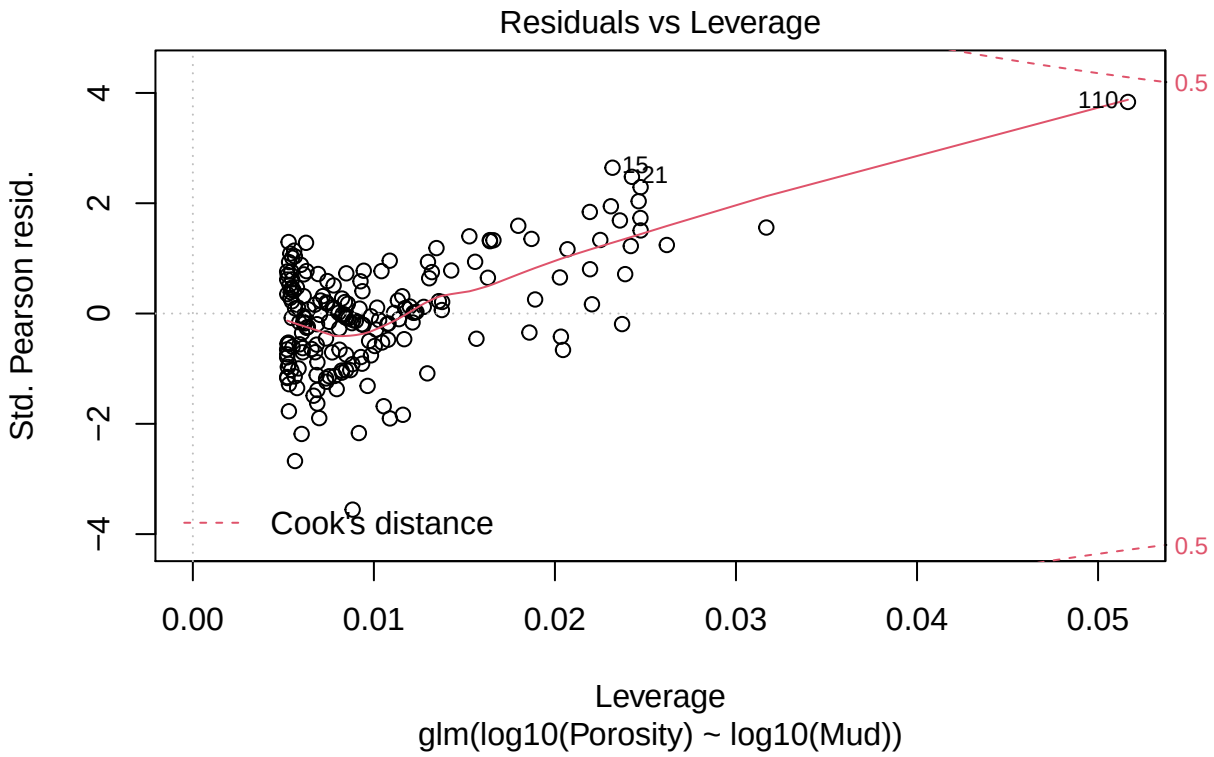
```
## (Intercept) -0.486388   0.007894  -61.62   <2e-16 ***
## log10(Mud)   0.138045   0.006216   22.21   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.0012629)
##
##     Null deviance: 0.86285  on 191  degrees of freedom
## Residual deviance: 0.23995  on 190  degrees of freedom
##   (56 observations deleted due to missingness)
## AIC: -732.61
##
## Number of Fisher Scoring iterations: 2
```

```
plot(Mod2)
```




Normal Q−Q
Std. Pearson resid.
4
2
0
−2
110
186
185
−3
−2
−1
0
1
2
3
Theoretical Quantiles
glm(log10(Porosity) ~ log10(Mud))


Scale−Location
√|Std. Pearson resid.|
0.0
0.5
1.0
1.5
2.0
110
185
186
−0.50
−0.45
−0.40
−0.35
−0.30
−0.25
−0.20
Predicted values
glm(log10(Porosity) ~ log10(Mud))




```
#' Diagnostics look pretty good. Residuals are random and centred around zero.
#' The fitted relationship is:
#'
#' $$Porosity = 0.138045 10^{Mud} -0.486388$$

## R2 for fitted relationships
Porosity_data %>%
  select(Porosity, Mud) %>%
  drop_na() %>%
  summarise(R2 = caret::R2(obs  = log10(Porosity),
                           pred = (log10(Mud)*0.138045 -0.486388),
                           formula = "traditional"))
```

```
##          R2
## 1 0.7219072
```

7.4 Predictive mapping

7.4.1 Permeability

```
Clyde_IrregularGrid$Permeability <- 10^(log10(Clyde_IrregularGrid$Mud*100)*-2.17403 - 10.22917)
```

7.4.2 Porosity

```
## fitted relationship
p1 <- -0.4338169
```

```
p2 <- 0.3013978
p3 <- -1.0376464
p4 <- -0.3096143

Clyde_IrregularGrid$Porosity <- 10^(p1 + p2*(1/(1 + exp(-(log10(Clyde_IrregularGrid$D50) - p3)/p4))))
```

8 Sediment movement

The following section is not run here - calculations are extremely computationally intensive. The code provided loads flow data outputted from a hydrodynamic model for the Clyde and calculates the bed shear stress based on published equations

```
library(devtools)
```

```
## Warning: package 'devtools' was built under R version 4.0.5
```

```
## Loading required package: usethis
```

```
## Warning: package 'usethis' was built under R version 4.0.5
```

```
## devtools::install_github("r4ecology/bedshear", dependencies = TRUE)
library(bedshear)
```

8.1 Load/prepare data objects

```
## Import Latitude and Longitude data
lat <- fread(input = "FVCOM_data/Clyde_Lat.csv")
lon <- fread(input = "FVCOM_data/Clyde_Lon.csv")

## Combine Lat/Lon
Clyde_LonLat <- data.frame(Lat = unlist(lat[1,]), Lon = unlist(lon[1,]))

## Import median grain size data (mm)
Clyde_D50 <- Clyde_IrregularGrid %>% select(Longitude, Latitude, D50)

## Import seabed depth (m)
Clyde_Depth <- Clyde_IrregularGrid %>% select(Latitude, Longitude, Depth)

Clyde_LonLat <- Clyde_LonLat %>%
  left_join(Clyde_D50, by = c("Lon" = "Longitude", "Lat" = "Latitude")) %>%
  left_join(Clyde_Depth, by = c("Lon" = "Longitude", "Lat" = "Latitude"))

## Also, just to make sure that we don't have issues with the data.table format
Clyde_LonLat <- as.data.frame(Clyde_LonLat)

## Check for rows where depth s zero, or D50 is NA.
complete_ind <- complete.cases(Clyde_LonLat)
sum(!complete_ind) ## no cases of NA
```

```
## [1] 0
```

8.2 Calculate pointwise bed shear stress and sediment movement

This section of code is required to calculate the pointwise bed shear stress and Shield stress based on hindcast estimates of tidal current speed for 2005 and 2006 from a high resolution hydrodynamic model of the Clyde.

Further details may be found in the main data paper.

This processes is computationally expensive and is provided here for illustrative purposes but is not run as part of this document.

```
library(fst)

# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

## Load this just to get the number of observation points
nvelocity <- 8735

## Set other variables
q1<- rep(1026.96, nvelocity)  # seawater density
q2<- rep(1.48e-3, nvelocity) # dynamic viscosity
q5<- rep(2650,    nvelocity)     # sediment density
q10<- 0       # switch to set input as mean wave period

## Some modifications so that it is only current only
currentdirection <- rep(40, nvelocity)  # current direction... value is unimportant
waveheight       <- rep(0,  nvelocity)
wavedirection    <- rep(1,  nvelocity)
waveperiod       <- rep(0,  nvelocity)

## Generate folders for outputs
if (!dir.exists("ShieldStress_fst/2005"))   dir.create("ShieldStress_fst/2005")
if (!dir.exists("ShieldStress_fst/2006"))   dir.create("ShieldStress_fst/2006")

if (!dir.exists("ShieldCrit_fst/2005"))     dir.create("ShieldCrit_fst/2005")
if (!dir.exists("ShieldCrit_fst/2006"))     dir.create("ShieldCrit_fst/2006")

if (!dir.exists("BedShearStress_fst/2005")) dir.create("BedShearStress_fst/2005")
if (!dir.exists("BedShearStress_fst/2006")) dir.create("BedShearStress_fst/2006")

## Initialise the loop
pb <- txtProgressBar(style = 3)
for (q in 1:nrow(Clyde_LonLat)) {

  # ----------------------#
  # Create dynamic data objects
  # ----------------------#

  ## extract the time history for a single point in 2005 and 2006
  Current_vec_2005 <- unlist(read_fst(paste0("FVCOM_data/2005/",q,".fst")))
  Current_vec_2006 <- unlist(read_fst(paste0("FVCOM_data/2006/",q,".fst")))

  ## For some stupid reason, the vectors that I'm importing have a

  ## extract the depth for a single point
  Depth <- Clyde_LonLat$Depth[q]

  ## q4 needs to be location specific (convert to metres)
  q4<- Clyde_LonLat$D50[q]/1000   # median grain size 200 microns
```

```
  # ------------------------------#
  # Run the shield stress function
  # ------------------------------#

  result_2005 <- bedshear::shear_stress(bathymetry = Depth,
                                        D50 = q4,
                                        tidal_velocity = Current_vec_2005,
                                        tidal_direction = currentdirection)

  result_2006 <- bedshear::shear_stress(bathymetry = Depth,
                                        D50 = q4,
                                        tidal_velocity = Current_vec_2006,
                                        tidal_direction = currentdirection)

  ## write shield stress
  write_fst(as.data.frame(result_2005$shields_number), paste0("ShieldStress_fst/2005/",q,".fst"))
  write_fst(as.data.frame(result_2006$shields_number), paste0("ShieldStress_fst/2006/",q,".fst"))

  ## write critical shield stress
  write_fst(as.data.frame(result_2005$shields_critical), paste0("ShieldCrit_fst/2005/",q,".fst"))
  write_fst(as.data.frame(result_2006$shields_critical), paste0("ShieldCrit_fst/2006/",q,".fst"))

  ## Write Mean Bed Shear Stress
  write_fst(as.data.frame(result_2005$shear_mean), paste0("BedShearStress_fst/2005/",q,".fst"))
  write_fst(as.data.frame(result_2006$shear_mean), paste0("BedShearStress_fst/2006/",q,".fst"))

  setTxtProgressBar(pb,q/nrow(Clyde_LonLat))
}
```

8.3 Annual bed shear stress summaries

```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

Clyde_BedShearStress <- data.frame()

## Loop to calculate the annual minimum, mean and maximum bed shear stress per point
pb <- txtProgressBar(style = 3)
for (i in 1:39503) {

  ## Calculate summary for each datapoint
  BedShearStress2005 <- read_fst(paste0("BedShearStress2005_fst/",i,".fst")) %>%
    summarise(MeanBedShearStress = mean(`result$shear_mean`),
              MinBedShearStress = min(`result$shear_mean`),
              MaxBedShearStress = max(`result$shear_mean`))

  BedShearStress2006 <- read_fst(paste0("BedShearStress2006_fst/",i,".fst")) %>%
    summarise(MeanBedShearStress = mean(`result$shear_mean`),
              MinBedShearStress = min(`result$shear_mean`),
              MaxBedShearStress = max(`result$shear_mean`))
```

```
  ## combine 2005 and 2006 data
  BedShearStress <- BedShearStress2005 %>% bind_rows(BedShearStress2006) %>%
    summarise_all(mean)

  Clyde_BedShearStress <- rbind(Clyde_BedShearStress, BedShearStress)


  ## update progress bar
  setTxtProgressBar(pb, i/39503)
}

Clyde_BedShearStress_Annual <- cbind(Clyde_LonLat, Clyde_BedShearStress)
```

8.4 Calculate Monthly proportion of time where seabed is disturbed

```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

## Set working directory
setwd(paste0(directory,"/",filepath_water))

## Generate object to store results
Clyde_SedimentMovement_Monthly <- data.frame()

## Define vector of month classes
MonthlyClasses <- c(rep(1, 31*24),
                    rep(2, 28*24),
                    rep(3, 31*24),
                    rep(4, 30*24),
                    rep(5, 31*24),
                    rep(6, 30*24),
                    rep(7, 31*24),
                    rep(8, 31*24),
                    rep(9, 30*24),
                    rep(10,31*24),
                    rep(11,30*24),
                    rep(12,(30*24-1)))

pb <- txtProgressBar(style = 3)
for (i in 1:39503) {

  ## Calculate summary for each data point
  TimeDisturb2005 <- read_fst(paste0("ShieldStress2005_fst/",i,".fst")) %>%
    bind_cols(read_fst(paste0("ShieldCrit2005_fst/",i,".fst"))) %>%
    mutate(Month = MonthlyClasses) %>%
    group_by(Month) %>%
    summarise(PropTimeDisturbed = sum(`result$shields_number` > `result$shields_critical`)/n(),
              MeanShieldStress = mean(`result$shields_number`),
              MeanShieldCrit = mean(`result$shields_critical`))

  TimeDisturb2006 <- read_fst(paste0("ShieldStress2006_fst/",i,".fst")) %>%
    bind_cols(read_fst(paste0("ShieldCrit2006_fst/",i,".fst"))) %>%
```

```
    mutate(Month = MonthlyClasses) %>%
    group_by(Month) %>%
    summarise(PropTimeDisturbed = sum(`result$shields_number` > `result$shields_critical`)/n(),
              MeanShieldStress = mean(`result$shields_number`),
              MeanShieldCrit = mean(`result$shields_critical`))

  StationLonLat <- Clyde_LonLat[i,] %>% mutate(Point = i)

  ## combine 2005 and 2006 data, calculate monthly mean for each point.
  TimeDisturb <- TimeDisturb2005 %>%
    bind_rows(TimeDisturb2006) %>%
    group_by(Month) %>%
    summarise_all(mean) %>%
    mutate(Point = i) %>%
    left_join(StationLonLat, by = c("Point"))

  Clyde_SedimentMovement_Monthly <- rbind(Clyde_SedimentMovement_Monthly, TimeDisturb)

  setTxtProgressBar(pb, i/39503)
}
```

9 Sediment particulate organic carbon and nitrogen content

For clarity, the pre-processing of data, compilation from multiple sources and interpolation of environmental covariates is not shown here but is decribed in detail in the main data paper.

```
library(gamm4)
```

```
## Warning: package 'gamm4' was built under R version 4.0.5
```

```
## This is gamm4 0.2-6
```

9.1 Define functions for cross-validaton and feature selection

9.1.1 sample_POCN()

Function to sample spatially explicit training and test seabed sediment organic carbon and nitrogen data. We assign 2/3 data to training and the remainder as test data.

- **df_nested**: A dataframe nested according to latitude/longitude bins

Return: A list of training data and test data.

```
sample_POCN <- function(df_nested) {

  sample_ok <- FALSE

  ## Loop if no training or test data is sampled
  while(sample_ok == FALSE) {

    ## randomly assign 2/3 data to training
    sample_data <- df_nested %>%
      ungroup() %>%
      mutate(Random = runif(nrow(df_nested))) %>%
      mutate(Train = Random < 0.66)
```

```
    ## Extract training data
    train_data <- sample_data %>%
      filter(Train) %>%
      select(data) %>%
      unnest(cols = c(data))

    ## Extract test data
    test_data <- sample_data %>%
      filter(!Train) %>%
      select(data) %>%
      unnest(cols = c(data))

    ## check that samples are correct
    if (nrow(test_data) != 0 & nrow(train_data) != 0) {sample_ok <- TRUE}
  }

  return(list(train_data = train_data,
              test_data  = test_data))
}
```

9.1.2 Overlay2()

A function to extract data from within grid cells and assign these values to overlaying points.

```
overlay2 <- function(data_df, covariate_df, data_lat, data_lon, cov_lat, cov_lon, cov_res) {

  ## Check if dataframes are dataframes
  if(class(data_df)[1] != "data.frame" | class(covariate_df)[1] != "data.frame") {
    stop("data_df and/or covariate_df are not data.frames")
  }

  ## Check if data_df contains NA in Latitude or Longitude
  if(sum(is.na(data_df[,data_lat])) > 0 | sum(is.na(data_df[,data_lon])) > 0) {
    stop("data_df contains NA in latitude or longitude vectors")
  }

  ## Create results object to store outputs
  extracted_df <- data.frame()

  for(i in 1:nrow(data_df)) {
    ## Calculate distance to each cell
    distance <- sqrt((covariate_df[,cov_lat] - unlist(data_df[i,][data_lat]))^2 +
                       (covariate_df[,cov_lon] - unlist(data_df[i,][data_lon]))^2)

    ## extract row
    extracted_row <- covariate_df[which.min(distance),]

    ## is point withing grid cell boundaries?
    maxlat <- extracted_row[,cov_lat] + cov_res/2
    minlat <- extracted_row[,cov_lat] - cov_res/2
    maxlon <- extracted_row[,cov_lon] + cov_res/2
    minlon <- extracted_row[,cov_lon] - cov_res/2

    within_bound <- unlist(data_df[i,][data_lat]) <= maxlat &
```

```
      unlist(data_df[i,][data_lat]) >= minlat &
      unlist(data_df[i,][data_lon]) <= maxlon &
      unlist(data_df[i,][data_lon]) >= minlon

    ## if extracted row is not within grid cell boundaries replace with NA
    if (within_bound == FALSE) {
      extracted_row[] <- NA
    }

    ## bind with results object
    extracted_df <- rbind(extracted_df,extracted_row)
  }
  return(cbind(data_df, extracted_df))
}
```

9.1.3 Spatial_R2_RF_POC()

A function to fit and spatially cross-validate a Random Forests Models for seabed particulate organic carbon

```
spatial_R2_RF_POC <- function(df_nested, reps = 1000, ...) {

  ## create objects to store results
  results_vector <- data.frame()
  all_r2 <- list()

  ## Loop over iterations
  pb <- txtProgressBar(style = 3)
  for(i in 1:reps){

    # ----------------------------------------#
    # Section 1: sample training and test data
    # ----------------------------------------#

    sample_data <- sample_POCN(df_nested)
    train_data  <- sample_data$train_data
    test_data   <- sample_data$test_data

    # ----------------------------------------#
    # Section 2.1: fit Generalised additive model
    # ----------------------------------------#

    rf_model <- ranger(POC ~ ., importance = "impurity", data = train_data,
                       num.trees = 2500)

    # ----------------------------------------#
    # Section 2.2: Generate predictions & calculate performance statistics
    # ----------------------------------------#

    all_r2[[i]] <- test_data %>%
      mutate(Predict = predict(rf_model,.) %>% predictions()) %>%
      select(POC, Predict) %>%
      summarize(R2 = caret::R2(pred = Predict, obs = POC, formula = "traditional"),
                MSE = mean((POC - Predict)^2)) %>%
      ungroup()
```

```
    setTxtProgressBar(pb, i/reps)
  }

  # ------------------------------------------------------------#
  # Section 3: Collate and summarise model performance statistics
  # ------------------------------------------------------------#

  ## find mean error across iterations and bind to results vector
  results_vector <- all_r2 %>%
    bind_rows() %>%
    mutate_all(function(x){
      cumsum(x)/row_number()})

  result_summary <- results_vector %>%
    summarise_all(function(x){tail(x,1)})

  ## Return the raw, processed and overall performance statistics
  return(list(Rsq_raw = all_r2 %>% bind_rows(),
              Rsq_vec = results_vector,
              Rsq_sum = result_summary))

}
```

9.1.4 Spatial_R2_RF_PON()

A function to fit and spatially cross-validate a Random Forests Models for seabed particulate organic nitrogen

```
spatial_R2_RF_PON <- function(df_nested, reps = 1000, ...) {

  ## create objects to store results
  results_vector <- data.frame()
  all_r2 <- list()

  ## Loop over iterations
  pb <- txtProgressBar(style = 3)
  for(i in 1:reps){

    # ----------------------------------------#
    # Section 1: sample training and test data
    # ----------------------------------------#

    sample_data <- sample_POCN(df_nested)
    train_data  <- sample_data$train_data
    test_data   <- sample_data$test_data

    # ----------------------------------------#
    # Section 2.1: fit Generalised additive model
    # ----------------------------------------#

    rf_model <- ranger(PON ~ ., importance = "impurity", data = train_data,
                       num.trees = 2500)

    # ----------------------------------------#
    # Section 2.2: Generate predictions & calculate performance statistics
```

```
    # ---------------------------------------#

    all_r2[[i]] <- test_data %>%
      mutate(Predict = predict(rf_model,.) %>% predictions()) %>%
      select(PON, Predict) %>%
      summarize(R2 = caret::R2(pred = Predict, obs = PON, formula = "traditional"),
                MSE = mean((PON - Predict)^2)) %>%
      ungroup()

    setTxtProgressBar(pb, i/reps)
  }

  # ---------------------------------------------------------------#
  # Section 3: Collate and summarise model performance statistics
  # ---------------------------------------------------------------#

  ## find mean error across iterations and bind to results vector
  results_vector <- all_r2 %>%
    bind_rows() %>%
    mutate_all(function(x){
      cumsum(x)/row_number()})

  result_summary <- results_vector %>%
    summarise_all(function(x){tail(x,1)})

  ## Return the raw, processed and overall performance statistics
  return(list(Rsq_raw = all_r2 %>% bind_rows(),
              Rsq_vec = results_vector,
              Rsq_sum = result_summary))

}
```

9.1.5 backselect_spatial_POC()

An improved function to iteratively drop variables and run the cross validation. The purpose is to evaluate contribution of each variable to overall model performance in the presence of other explanatory variables.

The function takes as arguments:

- df: a data frame containing:
  - Latitude: Latitude in decimal degrees
  - Longitude: Longitude in decimal degrees
  - source: source of measurement
  - POC: Seabed sediment particulate organic carbon
  - ...: Explanatory variables
- varnames: a vector of explanatory variables names to be tested iteratively dropped
- EliminatedVars: a vector of explanatory variables names that should not be evaluated. These may be variables that have already been elimated through a previous selection process.
- reps: the number of iterations to pass to the cross-validation function
- ...: other arguments that are passed to inner functions

The function returns a data frame containing the following vectors:

- VarNames: a character vector of the variables that were dropped from the "i"th model
- R2: The cross-validation r-squared for the "i"th model
- MSE: The cross-validation mean square error for the "i"th model

```r
backselect_spatial_POC <- function(df,
                                   EliminatedVars = NULL,
                                   varnames,
                                   reps = 1000,
                                   ...) {

  # ------------------------------------------------------------#
  # Section 1: Run initial cross validation with all variables
  # ------------------------------------------------------------#
  # Section 1.1: Spatially bin and nest data
  # ----------------------------------------#

  ## First prepare the nested data frame based on whether "EliminatedVars" is null
  ## or not

  if (is.null(EliminatedVars)) {

    df_nested <- df %>%
      mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
             bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
      group_by(bin_lat, bin_lon) %>%
      select(-Longitude, -Latitude, -source) %>%
      nest()

  } else {

    df_nested <- df %>%
      mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
             bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
      group_by(bin_lat, bin_lon) %>%
      select(-Longitude, -Latitude, -source) %>%
      select(-c(EliminatedVars)) %>%
      nest()

  }

  # ----------------------------------------------#
  # Section 1.2: Carry out spatial cross-validation
  # ----------------------------------------------#

  ## Spatial cross-validation
  RF_R2 <- spatial_R2_RF_POC(df_nested = df_nested, reps = reps, ...)

  ## Extract results from initial cross-validation
  df_result <- data.frame(VarName = "", RF_R2$Rsq_sum)

  # ------------------------------------------------------------------#
  # Section 2: Iteratively drop variables and run cross-validation
  # ------------------------------------------------------------------#
  # Section 2.1: Spatially bin and nest data
  # ----------------------------------------#

  ## Loop for each variable named in "varnames"
```

```
  for(rr in varnames){
    print(rr)

    if (is.null(EliminatedVars)) {

      df_nested <- df %>%
        mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
               bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
        group_by(bin_lat, bin_lon) %>%
        select(-Longitude, -Latitude, -source, -rr) %>%
        nest()

    } else {

      df_nested <- df %>%
        mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
               bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
        group_by(bin_lat, bin_lon) %>%
        select(-Longitude, -Latitude, -source, -rr) %>%
        select(-c(EliminatedVars)) %>%
        nest()

    }

    # ---------------------------------------------#
    #  Section 2.2: Carry out spatial cross-validation
    # ---------------------------------------------#

    RF_R2 <- spatial_R2_RF_POC(df_nested = df_nested, reps = reps, ...)

    df_result <- df_result %>%
      bind_rows(., data.frame(VarName = rr, RF_R2$Rsq_sum))
  }

  return(df_result)
}
```

9.1.6 backselect_spatial_PON()

An improved function to iteratively drop variables and run the cross validation. The purpose is to evaluate contribution of each variable to overall model performance in the presence of other explanatory variables.

The function takes as arguments:

- df: a data frame containing:
  - Latitude: Latitude in decimal degrees
  - Longitude: Longitude in decimal degrees
  - source: source of measurement
  - PON: Seabed sediment particulate organic nitrogen
  - ...: Explanatory variables
- varnames: a vector of explanatory variables names to be tested iteratively dropped
- EliminatedVars: a vector of explanatory variables names that should not be evaluated. These may be variables that have already been elimated through a previous selection process.
- reps: the number of iterations to pass to the cross-validation function
- ...: other arguments that are passed to inner functions

The function returns a data frame containing the following vectors:

- VarNames: a character vector of the variables that were dropped from the "i"th model
- R2: The cross-validation r-squared for the "i"th model
- MSE: The cross-validation mean square error for the "i"th model

```
backselect_spatial_PON <- function(df,
                                   EliminatedVars = NULL,
                                   varnames,
                                   reps = 1000,
                                   ...) {

  # ------------------------------------------------------------#
  # Section 1: Run initial cross validation with all variables
  # ------------------------------------------------------------#
  # Section 1.1: Spatially bin and nest data
  # ----------------------------------------#

  ## First prepare the nested data frame based on whether "EliminatedVars" is null
  ## or not

  if (is.null(EliminatedVars)) {

    df_nested <- df %>%
      mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
             bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
      group_by(bin_lat, bin_lon) %>%
      select(-Longitude, -Latitude, -source) %>%
      nest()

  } else {

    df_nested <- df %>%
      mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
             bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
      group_by(bin_lat, bin_lon) %>%
      select(-Longitude, -Latitude, -source) %>%
      select(-c(EliminatedVars)) %>%
      nest()

  }

  # ------------------------------------------#
  # Section 1.2: Carry out spatial cross-validation
  # ------------------------------------------#

  ## Spatial cross-validation
  RF_R2 <- spatial_R2_RF_PON(df_nested = df_nested, reps = reps, ...)

  ## Extract results from initial cross-validation
  df_result <- data.frame(VarName = "", RF_R2$Rsq_sum)

  # ------------------------------------------------------------#
  # Section 2: Iteratively drop variables and run cross-validation
  # ------------------------------------------------------------#
```

```
  # Section 2.1: Spatially bin and nest data
  # ----------------------------------------#

  ## Loop for each variable named in "varnames"
  for(rr in varnames){
    print(rr)

    if (is.null(EliminatedVars)) {

      df_nested <- df %>%
        mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
               bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
        group_by(bin_lat, bin_lon) %>%
        select(-Longitude, -Latitude, -source, -rr) %>%
        nest()

    } else {

      df_nested <- df %>%
        mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
               bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
        group_by(bin_lat, bin_lon) %>%
        select(-Longitude, -Latitude, -source, -rr) %>%
        select(-c(EliminatedVars)) %>%
        nest()

    }

    # ----------------------------------------------#
    #  Section 2.2: Carry out spatial cross-validation
    # ----------------------------------------------#

    RF_R2 <- spatial_R2_RF_PON(df_nested = df_nested, reps = reps)

    df_result <- df_result %>%
      bind_rows(., data.frame(VarName = rr, RF_R2$Rsq_sum))
  }

  return(df_result)
}
```

9.1.7 backselect_process_POC()

The previously defined functions are very computationally time-consuming, requiring hours to complete. This function automates the process of selecting the optimal model by taking a maximal model, iteratively evaluating the importance of each variable, and dropping the variable that least contributes to model performance before proceeding to the next round of variable evaluation.

This function may take several days to run. Therefore, progress summaries are periodically exported to allow for tracking the process. The function takes as arguments:

- df: a data frame containing:
  - Latitude: Latitude in decimal degrees
  - Longitude: Longitude in decimal degrees
  - source: source of measurement

  - POC: Seabed sediment particulate organic carbon
  - ...: Explanatory variables
- heuristic: a character element defining the metric to be used to compare model performance ("R2" or "MSE")
- reps: the number of iterations to pass to the cross-validation function
- OutputPrefix: Optional prefix used to name the
- ...: other arguments that are passed to inner functions

The function returns a data frame containing the following vectors:

- VarNames: a character vector of the variables that were dropped from the "i"th model in the "j"th round of model selection
- R2: The cross-validation r-squared for the "i"th model
- MSE: The cross-validation mean square error for the "i"th model
- stepcount: the "j"th iteration of model selection

```
backselect_process_POC <- function(df, heuristic = "R2", reps = 1000,
                                   OutputPrefix = "temp_file_backselect_", ...) {

  ## Check that objects supplied are in the correct format...
  if(!is.numeric(reps)) {
    stop("Argument [reps] is not an integer")
  }
  if(heuristic != "R2" & heuristic != "MSE") {
    stop("Argument [heuristic] must be [R2] or [MSE]")
  }
  if(sum(names(df) %in% c("Latitude", "Longitude", "POC", "source")) != 4){
    stop("Dataframe must contain: Latitude, Longitude, POC, source")
  }

  ## prepare tracking objects
  stepcounter <- 1
  EliminatedVars <- vector()

  # ------------------------------------------------------------#
  # Section 1: Initial round of model selection - Maximal model
  # ------------------------------------------------------------#
  #
  # For the first iteration, check the contribution of all candidate explanatory
  # variables to model performance.

  ## define a vector of candidate explanatory variable names
  varnames <- df %>%
    select(-Longitude, -Latitude, -POC, -source) %>%
    names()

  ## evaluate model performance given removal of each variable
  df_result <- backselect_spatial_POC(df = df,
                                      varnames = varnames,
                                      reps = reps,
                                      ...)

  ## Identify the variable that least increases R2 or decreases MSE
  if(heuristic == "R2") {
    dropvar <- df_result %>%
```

```
    filter(VarName != "") %>%
    arrange(desc(R2)) %>%
    select(VarName) %>%
    filter(row_number() == 1) %>% as.character()
} else if (heuristic == "MSE"){
  dropvar <- df_result %>%
    filter(VarName != "") %>%
    arrange(MSE) %>%
    select(VarName) %>%
    filter(row_number() == 1) %>% as.character()
}

## Update modeltrace - export snapshot
df_result$stepcount <- stepcounter
modeltrace <- df_result
fwrite(modeltrace, paste0(OutputPrefix,heuristic,".csv"))

# ----------------------------------------------------------#
# Section 2: Model selection
# ----------------------------------------------------------#
#
# This section comes into effect after the first variable to be removed has
# been identified, and continues to identify and drop "low-performing"
# explanatory variables.

## Begin Dropping process
KeepDropping <- TRUE

## Keep looping as long as "KeepDropping" is true
while (KeepDropping == TRUE) {
  stepcounter <- stepcounter + 1

  ## Add the most recently identified low-performing variable to EliminatedVars
  EliminatedVars <- append(EliminatedVars, dropvar)

  ## define a vector of candidate explanatory variable names (removing eliminated variables)
  varnames <- df %>%
    select(-Longitude, -Latitude, -POC, -source) %>%
    select(-c(EliminatedVars)) %>%
    names()

  ## evaluate model performance given removal of each variable
  df_result <- backselect_spatial_POC(df = df,
                                      EliminatedVars = EliminatedVars,
                                      varnames = varnames,
                                      reps = reps,
                                      ...)

  ## Identify the variable that least increases R2 or decreases MSE
  if(heuristic == "R2") {
    dropvar <- df_result %>%
      filter(VarName != "") %>%
      arrange(desc(R2)) %>%
```

```
        select(VarName) %>%
        filter(row_number() == 1) %>% as.character()
    } else if (heuristic == "MSE"){
      dropvar <- df_result %>%
        filter(VarName != "") %>%
        arrange(MSE) %>%
        select(VarName) %>%
        filter(row_number() == 1) %>% as.character()
    }

    ## Update modeltrace - export snapshot
    df_result$stepcount <- stepcounter
    modeltrace <- rbind(modeltrace, df_result)

    fwrite(modeltrace, paste0(OutputPrefix,heuristic,".csv"))

    ## Continue dropping until only two variables remain
    if(nrow(df_result) <= 2) {KeepDropping <- FALSE}
  }

  return(df_result)
}
```

9.1.8 backselect_process_PON()

The previously defined functions are very computationally time-consuming, requiring hours to complete. This function automates the process of selecting the optimal model by taking a maximal model, iteratively evaluating the importance of each variable, and dropping the variable that least contributes to model performance before proceeding to the next round of variable evaluation.

This function may take several days to run. Therefore, progress summaries are periodically exported to allow for tracking the process. The function takes as arguments:

- df: a data frame containing:
  - Latitude: Latitude in decimal degrees
  - Longitude: Longitude in decimal degrees
  - source: source of measurement
  - PON: Seabed sediment particulate organic nitrogen
  - ...: Explanatory variables
- heuristic: a character element defining the metric to be used to compare model performance ("R2" or "MSE")
- reps: the number of iterations to pass to the cross-validation function
- OutputPrefix: Optional prefix used to name the
- ...: other arguments that are passed to inner functions

The function returns a data frame containing the following vectors:

- VarNames: a character vector of the variables that were dropped from the "i"th model in the "j"th round of model selection
- R2: The cross-validation r-squared for the "i"th model
- MSE: The cross-validation mean square error for the "i"th model
- stepcount: the "j"th iteration of model selection

```
backselect_process_PON <- function(df, heuristic = "R2", reps = 1000,
                                   OutputPrefix = "temp_file_backselect_", ...) {
```

```r
## Check that objects supplied are in the correct format...
if(!is.numeric(reps)) {
  stop("Argument [reps] is not an integer")
}
if(heuristic != "R2" & heuristic != "MSE") {
  stop("Argument [heuristic] must be [R2] or [MSE]")
}
if(sum(names(df) %in% c("Latitude", "Longitude", "PON", "source")) != 4){
  stop("Dataframe must contain: Latitude, Longitude, PON, source")
}

## prepare tracking objects
stepcounter <- 1
EliminatedVars <- vector()

# ------------------------------------------------------------#
# Section 1: Initial round of model selection - Maximal model
# ------------------------------------------------------------#
#
# For the first iteration, check the contribution of all candidate explanatory
# variables to model performance.

## define a vector of candidate explanatory variable names
varnames <- df %>%
  select(-Longitude, -Latitude, -PON, -source) %>%
  names()

## evaluate model performance given removal of each variable
df_result <- backselect_spatial_PON(df = df,
                                    varnames = varnames,
                                    reps = reps,
                                    ...)

## Identify the variable that least increases R2 or decreases MSE
if(heuristic == "R2") {
  dropvar <- df_result %>%
    filter(VarName != "") %>%
    arrange(desc(R2)) %>%
    select(VarName) %>%
    filter(row_number() == 1) %>%
    as.character()
} else if (heuristic == "MSE"){
  dropvar <- df_result %>%
    filter(VarName != "") %>%
    arrange(MSE) %>%
    select(VarName) %>%
    filter(row_number() == 1) %>%
    as.character()
}

## Update modeltrace - export snapshot
df_result$stepcount <- stepcounter
modeltrace <- df_result
```

```
fwrite(modeltrace, paste0(OutputPrefix,heuristic,".csv"))

# ------------------------------------------------------------#
# Section 2: Model selection
# ------------------------------------------------------------#
#
# This section comes into effect after the first variable to be removed has
# been identified, and continues to identify and drop "low-performing"
# explanatory variables.

## Begin Dropping process
KeepDropping <- TRUE

## Keep looping as long as "KeepDropping" is true
while (KeepDropping == TRUE) {
  stepcounter <- stepcounter + 1

  ## Add the most recently identified low-performing variable to EliminatedVars
  EliminatedVars <- append(EliminatedVars, dropvar)

  ## define a vector of candidate explanatory variable names (removing eliminated variables)
  varnames <- df %>%
    select(-Longitude, -Latitude, -PON, -source) %>%
    select(-c(EliminatedVars)) %>%
    names()

  ## evaluate model performance given removal of each variable
  df_result <- backselect_spatial_PON(df = df,
                                      EliminatedVars = EliminatedVars,
                                      varnames = varnames,
                                      reps = reps,
                                      ...)

  ## Identify the variable that least increases R2 or decreases MSE
  if(heuristic == "R2") {
    dropvar <- df_result %>% filter(VarName != "") %>%
      arrange(desc(R2)) %>% select(VarName) %>% filter(row_number() == 1) %>%
      as.character()
  } else if (heuristic == "MSE"){
    dropvar <- df_result %>% filter(VarName != "") %>%
      arrange(MSE) %>% select(VarName) %>% filter(row_number() == 1) %>%
      as.character()
  }

  ## Update modeltrace - export snapshot
  df_result$stepcount <- stepcounter
  modeltrace <- rbind(modeltrace, df_result)

  fwrite(modeltrace, paste0(OutputPrefix,heuristic,".csv"))

  ## Continue dropping
  if(nrow(df_result) <= 2) {KeepDropping <- FALSE}
}
```

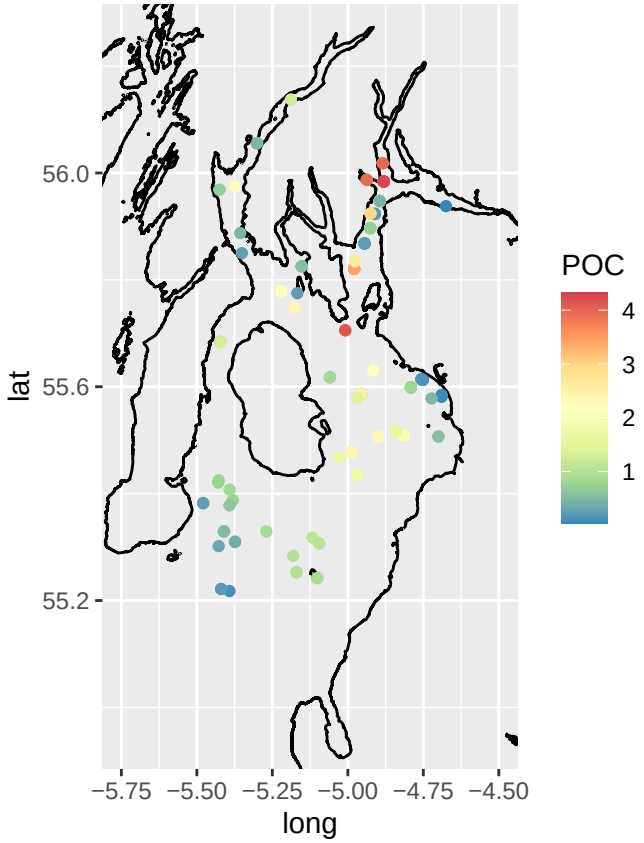
```
  return(df_result)
}
```

9.2 Load compiled Clyde sediment particulate OC & N data

Here I load seabed sediment particulate organic carbon and nitrogen data and generate some quick plots to look at distribution of measurements and correlations among variables.

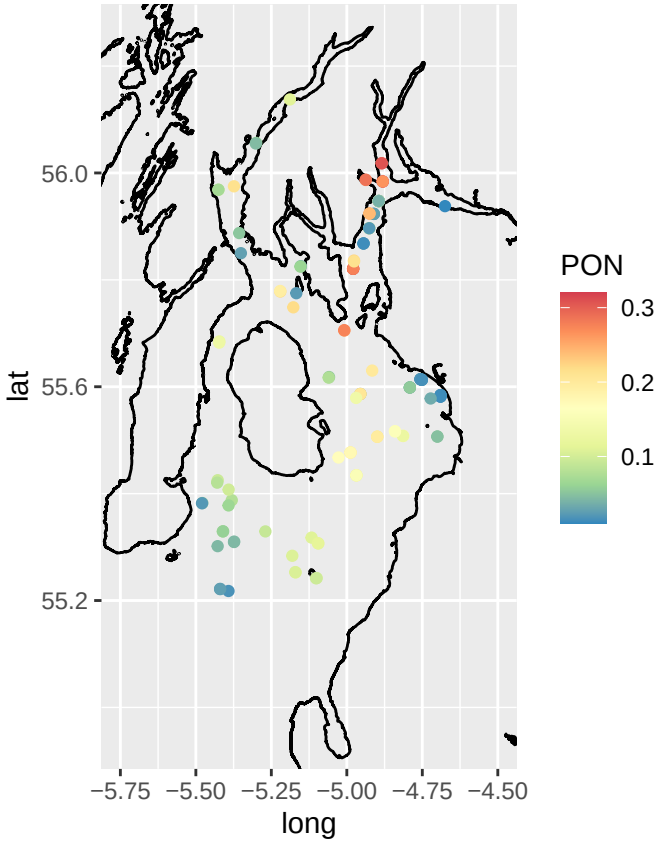
```
Combined_POCN <- fread("data/Clyde_POCN.csv")

Combined_PON <- Combined_POCN %>%
  select(-POC) %>%
  filter(PON > 0, PON < 15) %>%
  drop_na() # n = 117

Combined_POC <- Combined_POCN %>%
  select(-PON) %>%
  filter(POC > 0) %>%
  drop_na() # n = 168

## Particulate organic carbon
Combined_POC %>%
  ggplot() +
  geom_path(aes(x = long, y = lat, group = group), data = fortify(Shape_coast)) +
  geom_point(aes(x = Longitude, y = Latitude, colour = POC)) +
  coord_quickmap(xlim = c(-5.75,-4.50), ylim = c(54.95,56.25)) +
  scale_colour_distiller(palette = "Spectral")
```

```
## Regions defined for each Polygons
```

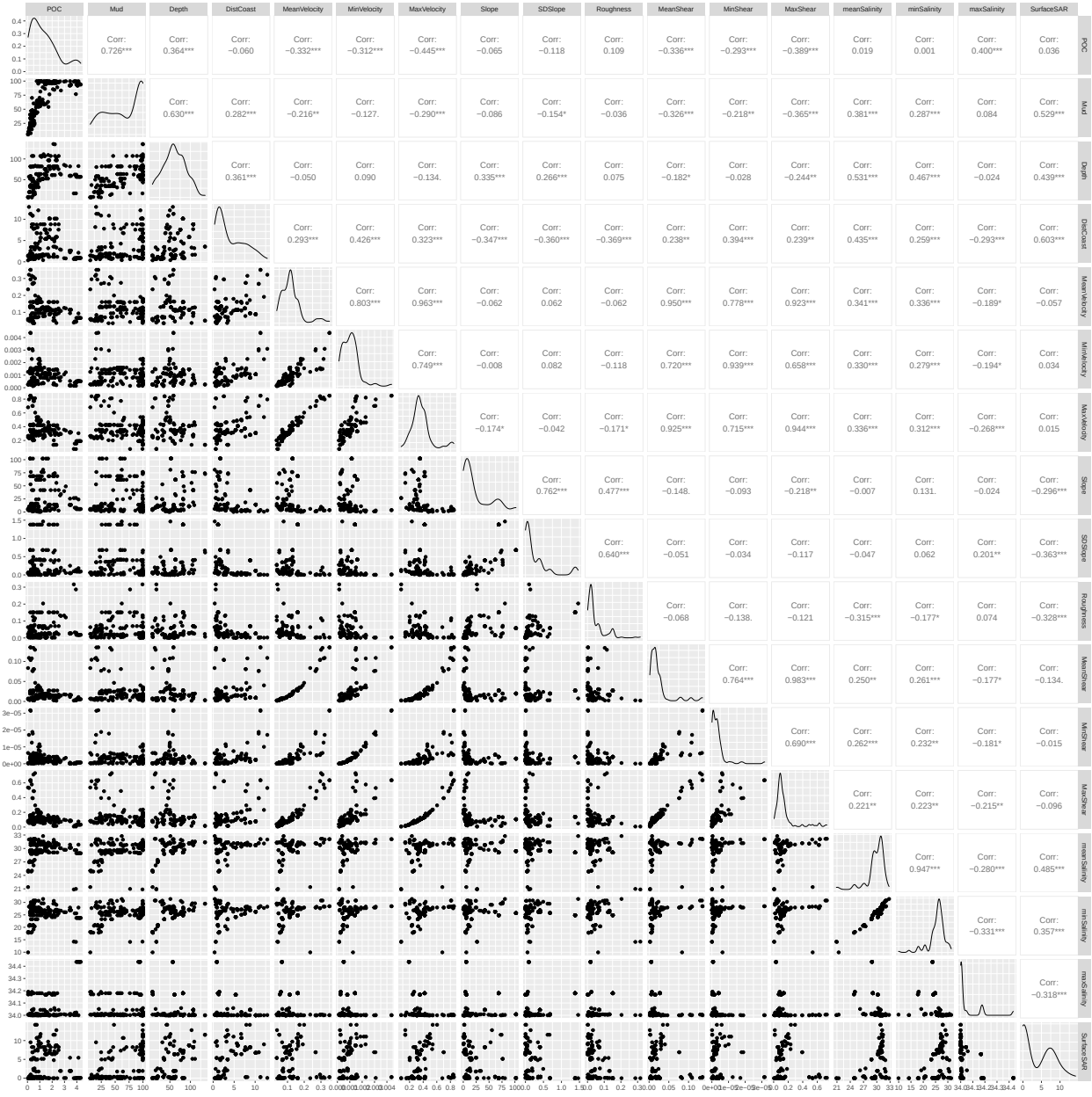



```
## Particulate organic nitrogen
Combined_PON %>%
  ggplot() +
  geom_path(aes(x = long, y = lat, group = group), data = fortify(Shape_coast)) +
  geom_point(aes(x = Longitude, y = Latitude, colour = PON)) +
  coord_quickmap(xlim = c(-5.75,-4.50), ylim = c(54.95,56.25)) +
  scale_colour_distiller(palette = "Spectral")
```

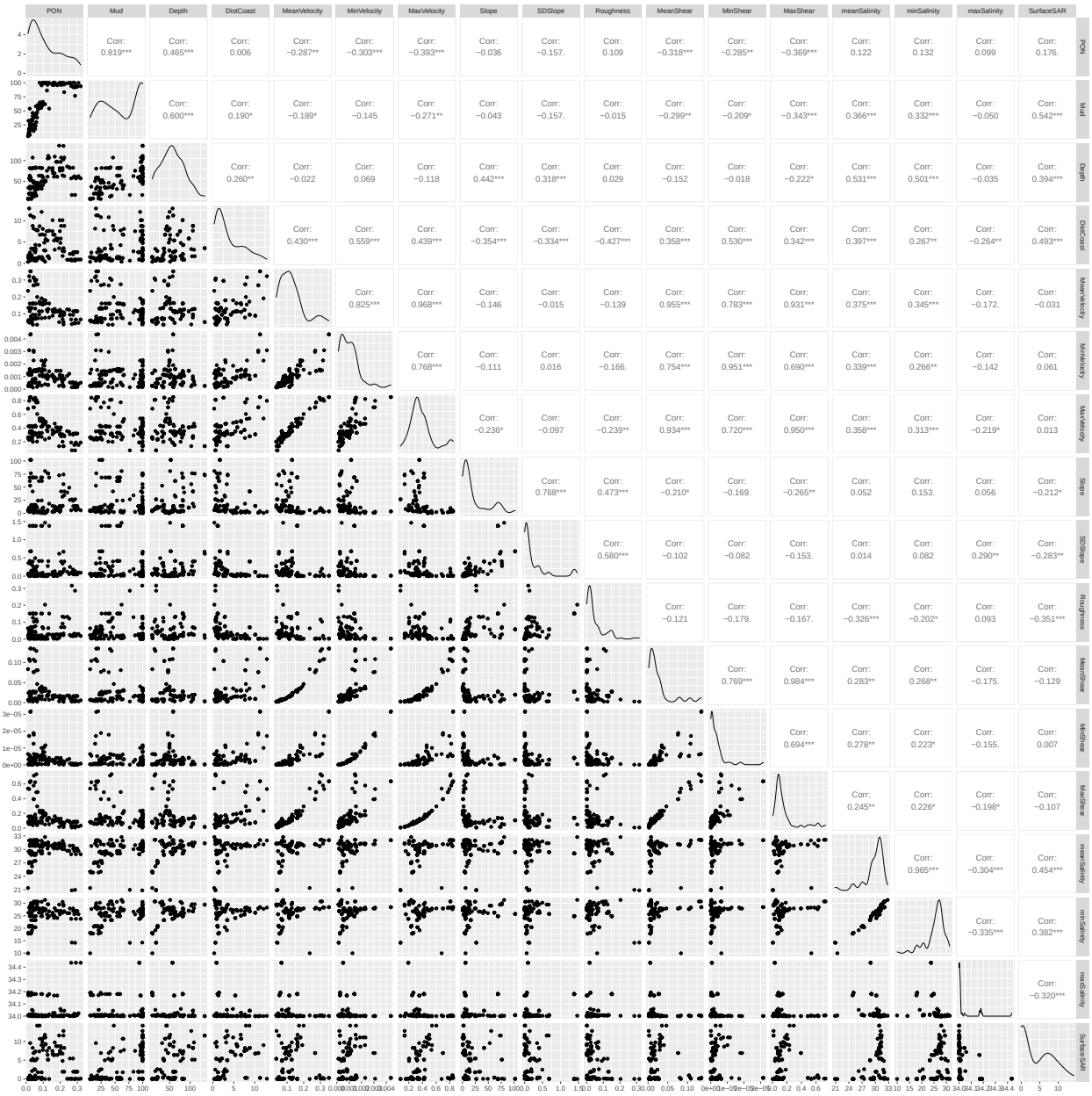
```
## Regions defined for each Polygons
```




```
## Examine pair-wise correlations
GGally::ggpairs(Combined_POC %>% select(-Latitude, -Longitude, -source))
```




```
GGally::ggpairs(Combined_PON %>% select(-Latitude, -Longitude, -source))
```



9.3 Random Forests model fitting

I'm evaluating the following variables as predictors:

- Depth
- Roughness
- Slope
- SDSlope
- DistCoast
- MinVelocity
- MeanVelocity
- MaxVelocity
- MinShear
- MeanShear

- MaxShear
- minSalinity
- meanSalinity
- max Salinity
- SurfaceSAR
- Mud content

```
## Is there any benefit to arcsine transformation?
POC_noTrans <- Combined_POC %>%
  select(-Latitude, -Longitude, -source) %>%
  ranger(POC ~ ., importance = "impurity", data = .)

POC_trans <- Combined_POC %>%
  select(-Latitude, -Longitude, -source) %>%
  ranger(asin(sqrt(POC/100)) ~ ., importance = "impurity", data = .)

print(POC_noTrans)
```

```
## Ranger result
##
## Call:
##  ranger(POC ~ ., importance = "impurity", data = .)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      168
## Number of independent variables:  16
## Mtry:                             4
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.1182085
## R squared (OOB):                  0.9086993
```

```
print(POC_trans)
```

```
## Ranger result
##
## Call:
##  ranger(asin(sqrt(POC/100)) ~ ., importance = "impurity", data = .)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      168
## Number of independent variables:  16
## Mtry:                             4
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.000210282
## R squared (OOB):                  0.9096786
```

Although this is carried out using out-of-box error, this is still a level of stochastic sampling and shows that there is no added value to arcsine transformation of the data in this instance.

9.3.1 Backwards Selection using Spatial Cross-validation

Here, I choose a flexible framework for the backward selection of explanatory variables based on model cross-validation performance. However, this is very time consuming as 1000s of models must be fitted. The first step is to check whether the bin size used is appropriate.

9.3.1.1 Check spatial autocorrelation in model residuals

It appears that the out-of-bag sampling within the Random Forests models already disrupts much of the spatial dependency within the data.

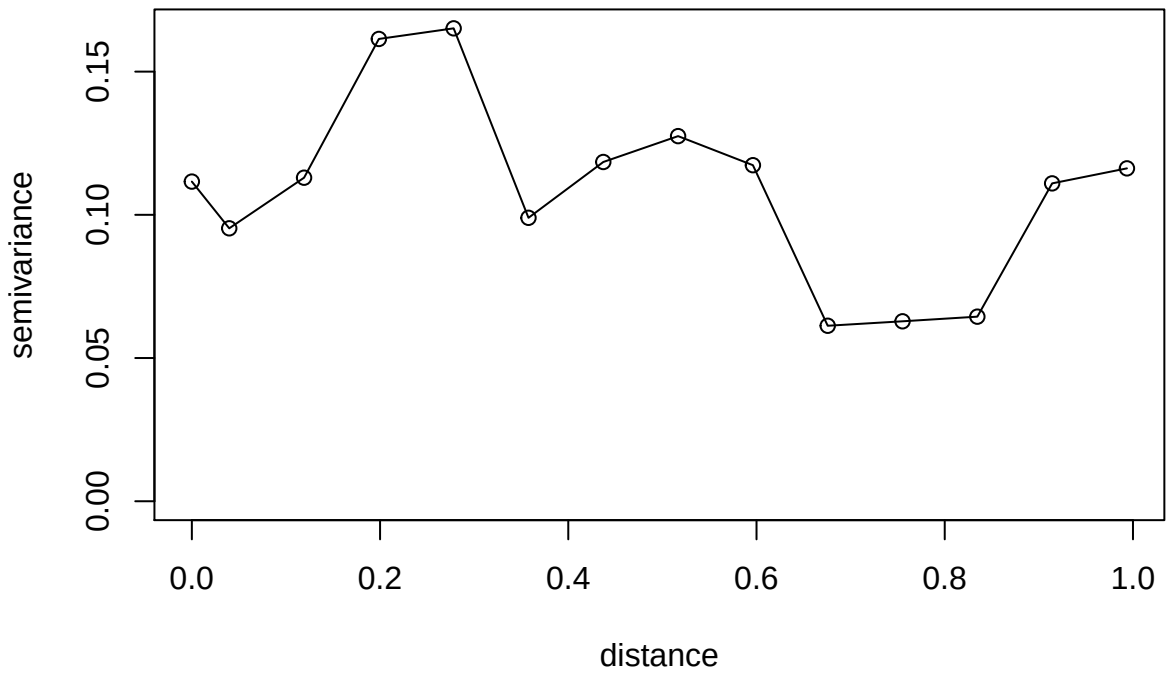
```
Samples_POC <- Combined_POC %>%
    mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
           bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
    group_by(bin_lat, bin_lon) %>%
    select(-source)

Samples_PON <- Combined_PON %>%
    mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
           bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
    group_by(bin_lat, bin_lon) %>%
    select(-source)

models_POC <- ranger(POC ~ . , importance = "impurity",
                       data = Samples_POC %>% select(-Latitude, -Longitude),
                     num.trees = 1000)
models_PON <- ranger(PON ~ . , importance = "impurity",
                       data = Samples_PON %>% select(-Latitude, -Longitude),
                     num.trees = 1000)

## Compute semivariogram using geoR package
Sample_POC_variogram <- geoR::variog(geodata = list(
  coords = matrix(c(Samples_POC$Latitude,
                    Samples_POC$Longitude),
                  ncol = 2),
                  data = (Samples_POC$POC - models_POC$predictions)))
```

```
## variog: computing omnidirectional variogram
## variog: co-locatted data found, adding one bin at the origin
```

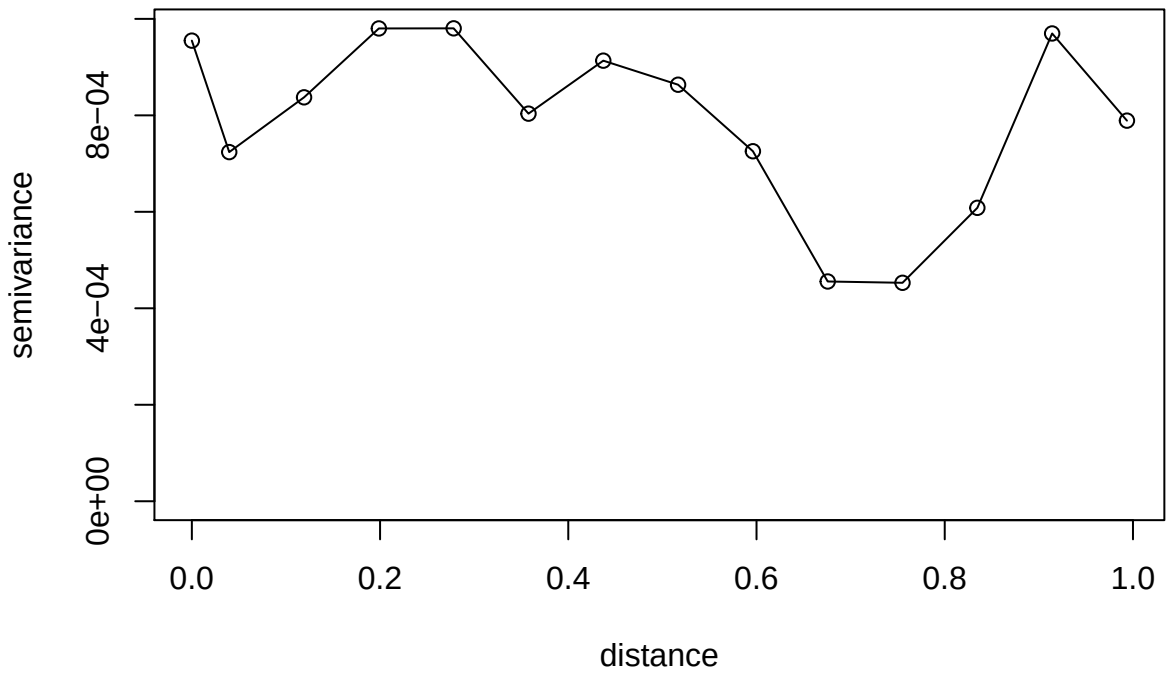
```
Sample_PON_variogram <- geoR::variog(geodata = list(
  coords = matrix(c(Samples_PON$Latitude,
                    Samples_PON$Longitude),
                  ncol = 2),
                  data = (Samples_PON$PON - models_PON$predictions)))
```

```
## variog: computing omnidirectional variogram
## variog: co-locatted data found, adding one bin at the origin
```

```
## spatial autocorrelation in organic carbon model
plot(Sample_POC_variogram, type = "o")
```

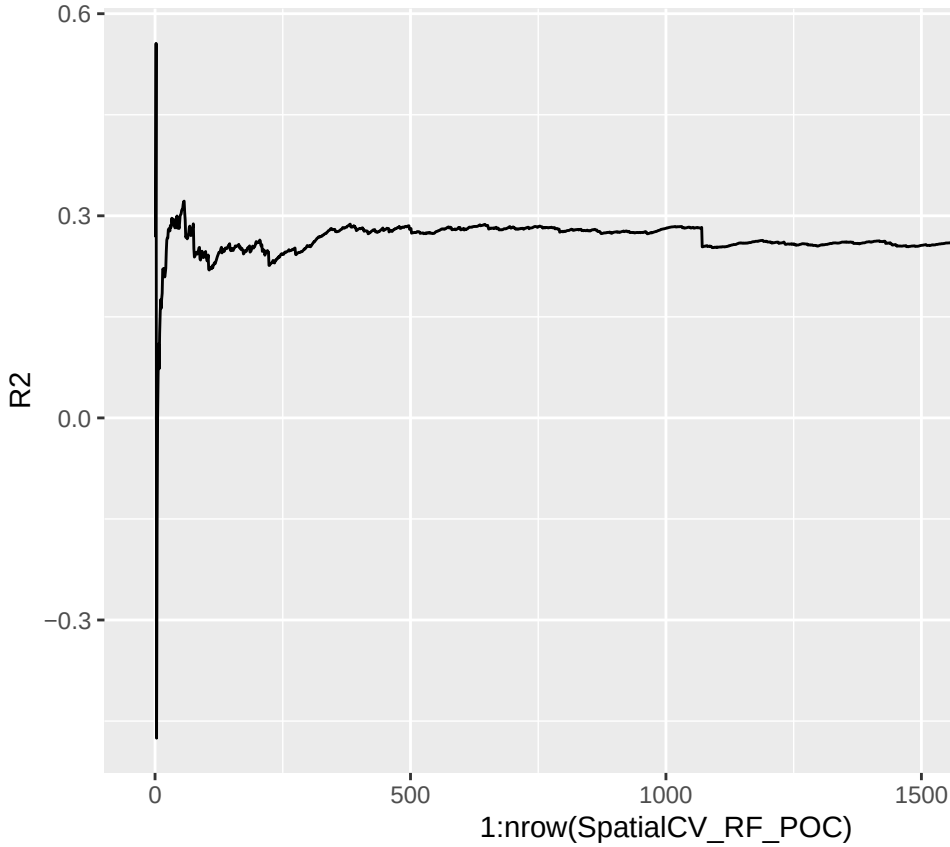



```
## spatial autocorrelation in nitrogen model
plot(Sample_PON_variogram, type = "o")
```



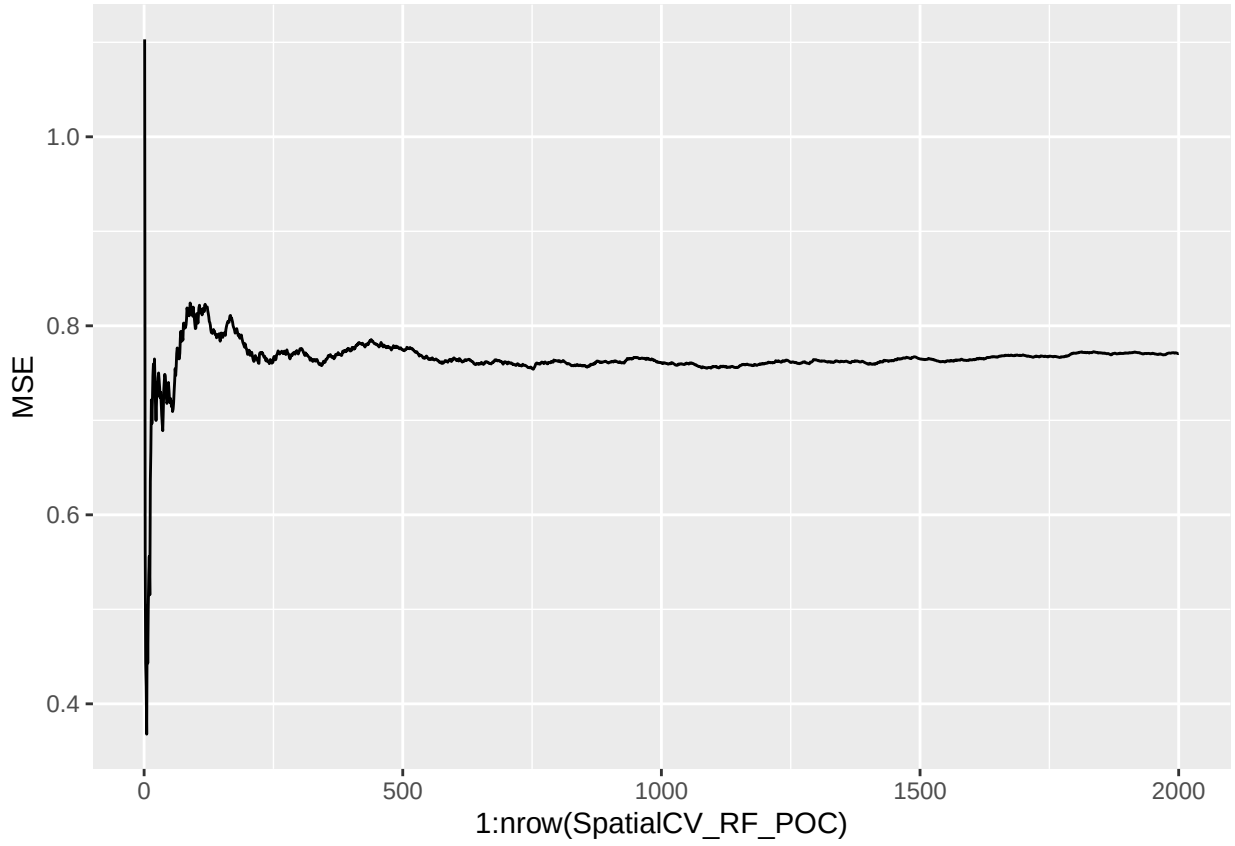

```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

## Initial run to choose number of iterations
SpatialCV_RF_POC <- Combined_POC %>%
  mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
         bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
  mutate(bin_lat = as.character(bin_lat),
         bin_lon = as.character(bin_lon)) %>%
  group_by(bin_lat, bin_lon) %>%
  select(-Longitude, -Latitude, -source) %>%
  nest() %>%
  spatial_R2_RF_POC(df_nested = ., reps = 2000)
```

```
SpatialCV_RF_POC <- fread("OrganicCarbon_RF_SpatialCrossValidation_init_raw.csv")

SpatialCV_RF_POC %>%
    drop_na() %>%
  mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  ggplot(aes(y = R2, x = 1:nrow(SpatialCV_RF_POC))) + geom_line()
```



9.3.1.2 Particulate organic carbon

```
SpatialCV_RF_POC %>%
    drop_na() %>%
    mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  ggplot(aes(y = MSE, x = 1:nrow(SpatialCV_RF_POC))) + geom_line()
```



On the basis of the initial pass, 2000 iterations appears to be suitable for this exercise.

I use a custom function to perform the backwards selection process, with 2000 replicate models fitted for each variable. Because the mean square error shows more consistent behaviour than R2, I use this as the heuristic according to which I select variables.

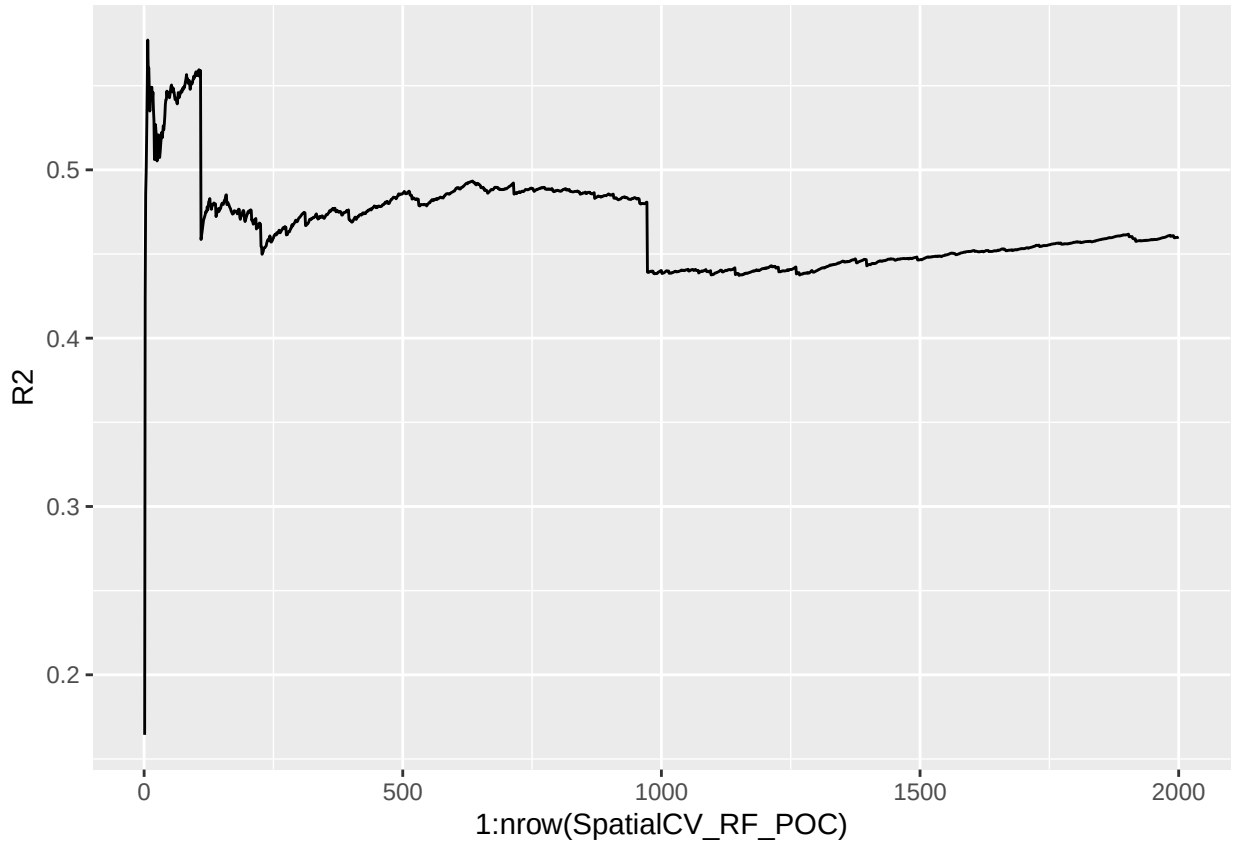
```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

df_result <- Combined_POC %>%
  backselect_process(df = .,
                     heuristic = "MSE",
                     reps = 2000,
                     OutputPrefix = "POC_BackSelection_FVCOMsalinity_v3_1_reps2000_")
```

Final check of selected model performance.

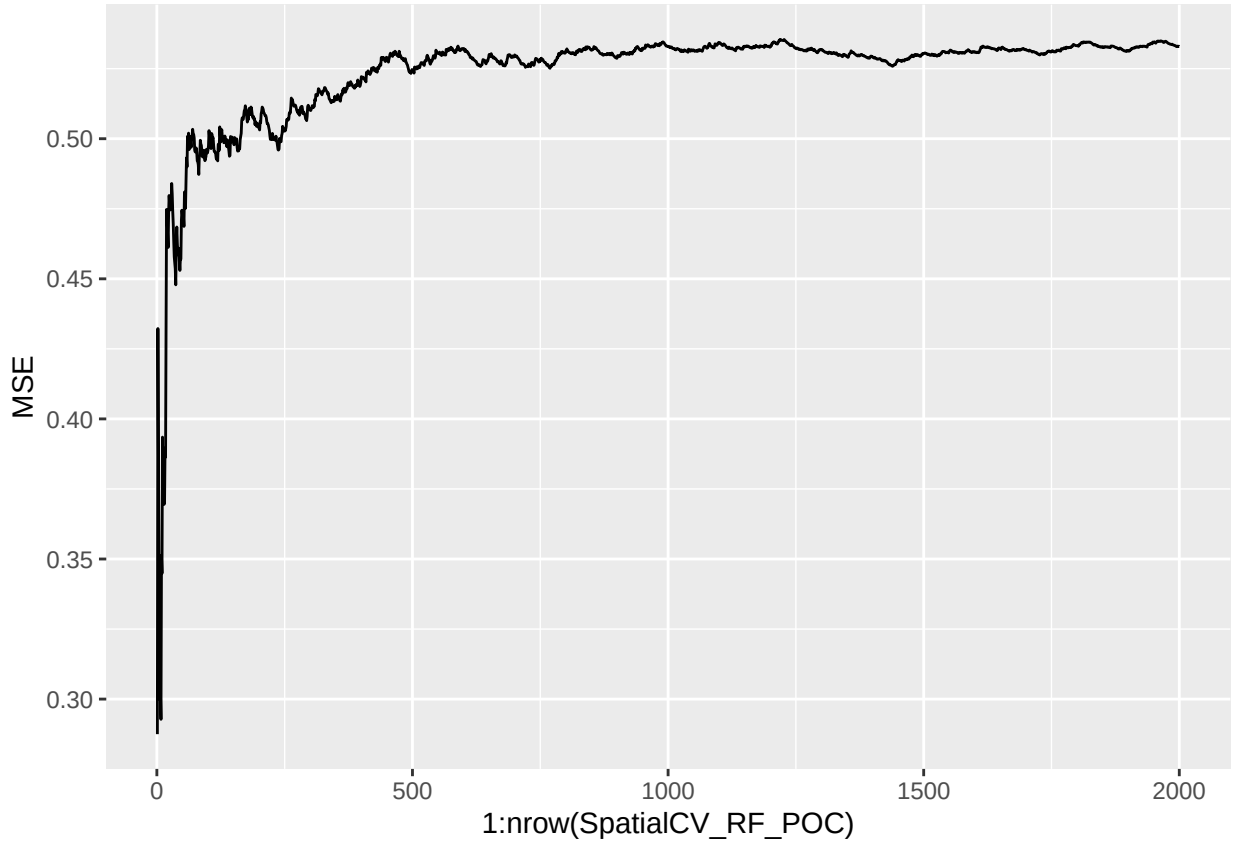
```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

SpatialCV_RF_POC <- Combined_POC %>%
  mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
         bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
  mutate(bin_lat = as.character(bin_lat),
         bin_lon = as.character(bin_lon)) %>%
  group_by(bin_lat, bin_lon) %>%
```

```
  select(-Longitude, -Latitude, -source) %>%
  select(POC, Mud, MaxShear, DistCoast, Roughness) %>%
  nest() %>%
  spatial_R2_RF_POC(df_nested = ., reps = 2000)
```

```
SpatialCV_RF_POC <- fread("OrganicCarbon_RF_SpatialCrossValidation_final_raw.csv")

SpatialCV_RF_POC %>%
    drop_na() %>%
  mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  ggplot(aes(y = R2, x = 1:nrow(SpatialCV_RF_POC))) + geom_line()
```

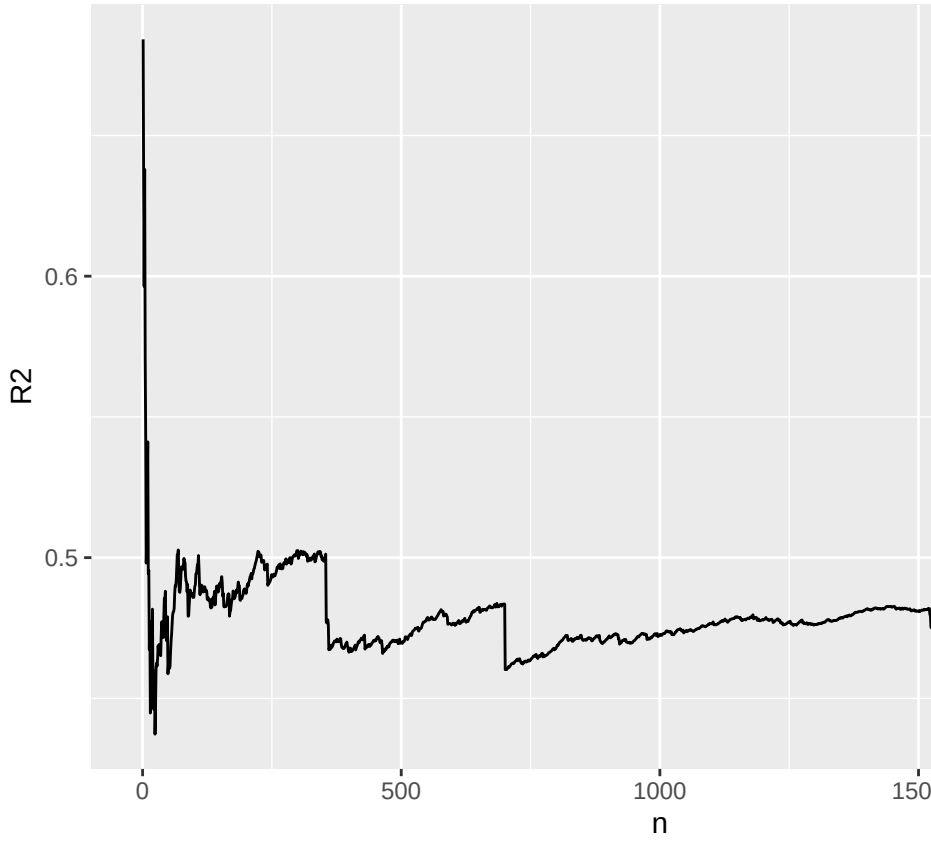



```
SpatialCV_RF_POC %>%
    drop_na() %>%
    mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  ggplot(aes(y = MSE, x = 1:nrow(SpatialCV_RF_POC))) + geom_line()
```



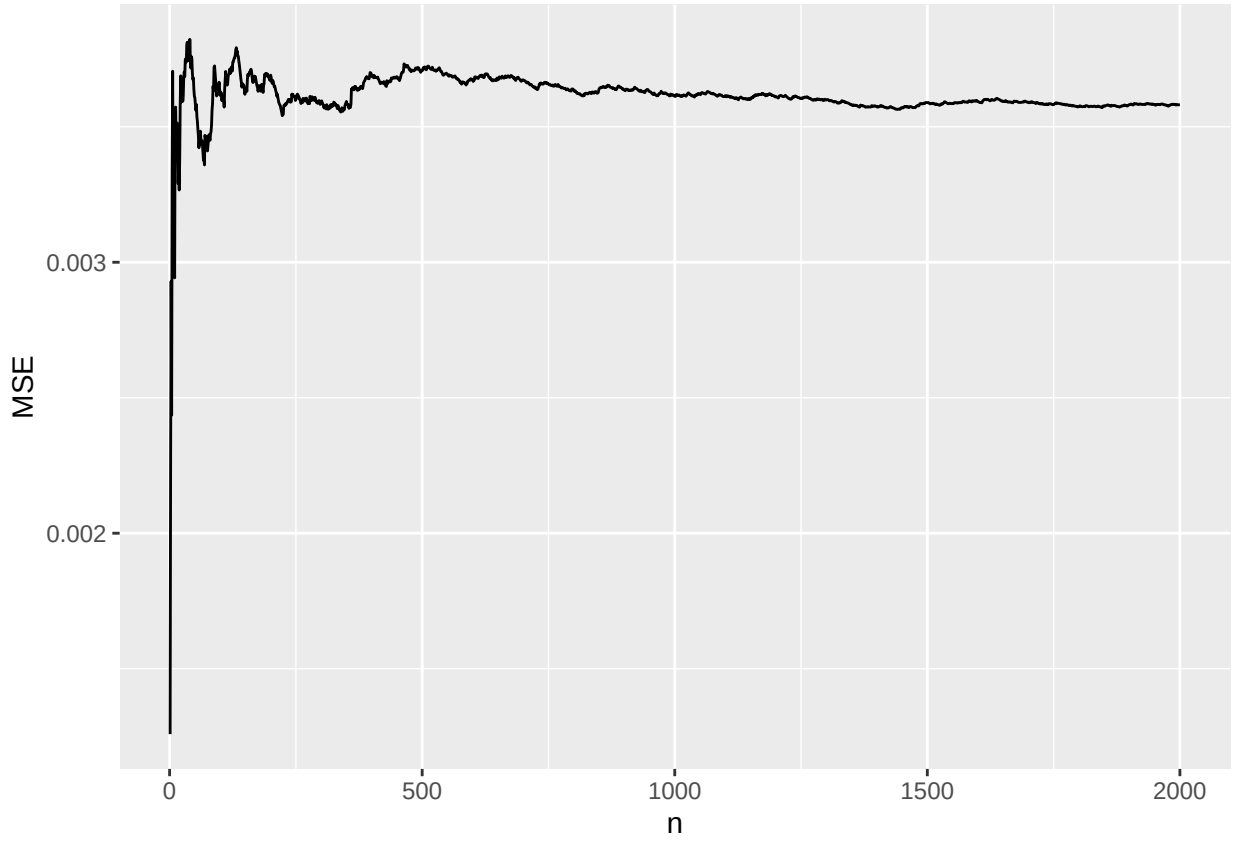

```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

## Initial run to choose number of iterations
SpatialCV_RF_PON <- Combined_PON %>%
  mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
         bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
  mutate(bin_lat = as.character(bin_lat),
         bin_lon = as.character(bin_lon)) %>%
  group_by(bin_lat, bin_lon) %>%
  select(-Longitude, -Latitude, -source) %>%
  nest() %>%
  spatial_R2_RF_PON(df_nested = ., reps = 2000)
```

```
SpatialCV_RF_PON <- fread("OrganicNitrogen_RF_SpatialCrossValidation_init_raw.csv")

SpatialCV_RF_PON %>%
    drop_na() %>%
  mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  mutate(n = row_number()) %>%
  ggplot(aes(y = R2, x = n)) + geom_line()
```



9.3.1.3 Particulate organic nitrogen

```
SpatialCV_RF_PON %>%
    drop_na() %>%
    mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  mutate(n = row_number()) %>%
  ggplot(aes(y = MSE, x = n)) + geom_line()
```



Just as with particulate organic carbon, error stabilised within 2000 iterations.

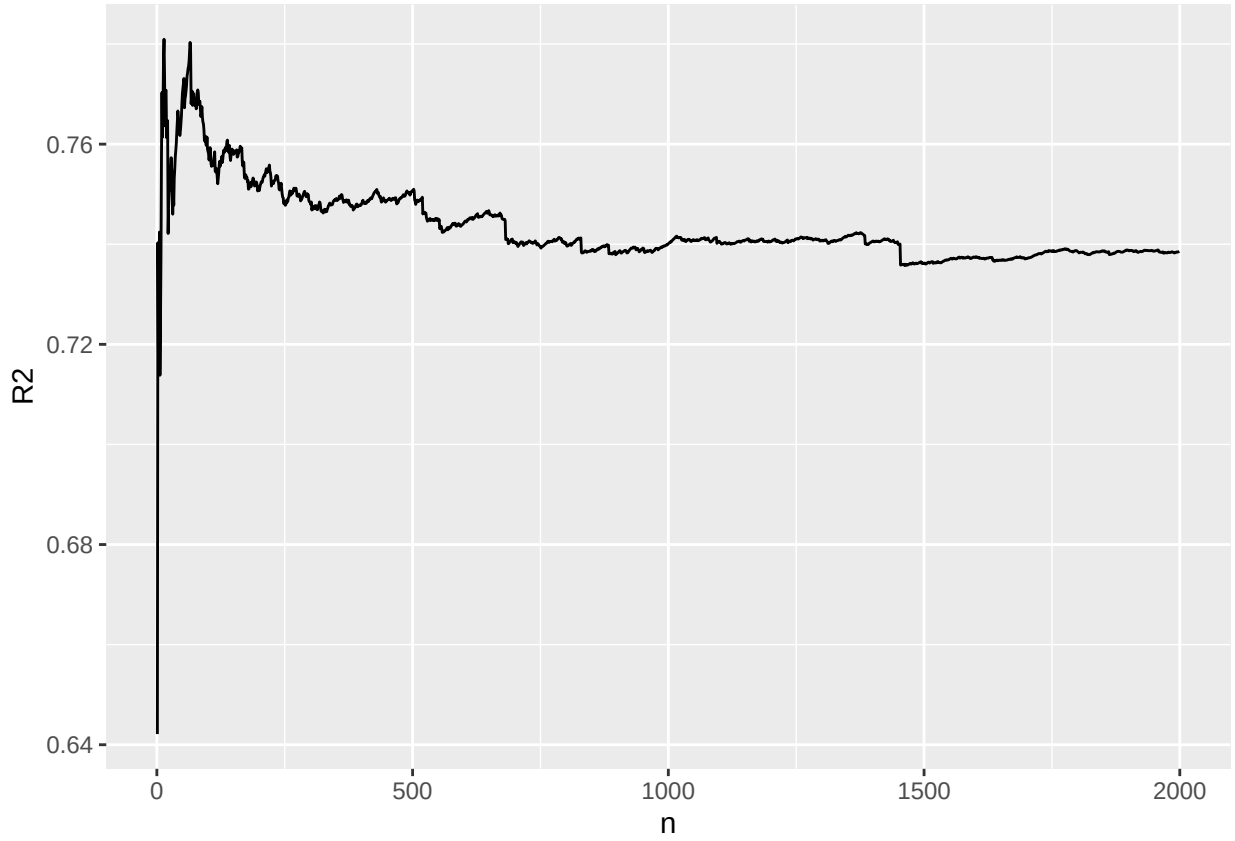
```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

df_result_MSE <- Combined_PON %>%
  backselect_process(df = .,
                     heuristic = "MSE",
                     reps = 2000,
                     OutputPrefix = "PON_BackSelection_FVCOMsalinity_v3_1_reps2000_")
```

Final check of selected model performance.

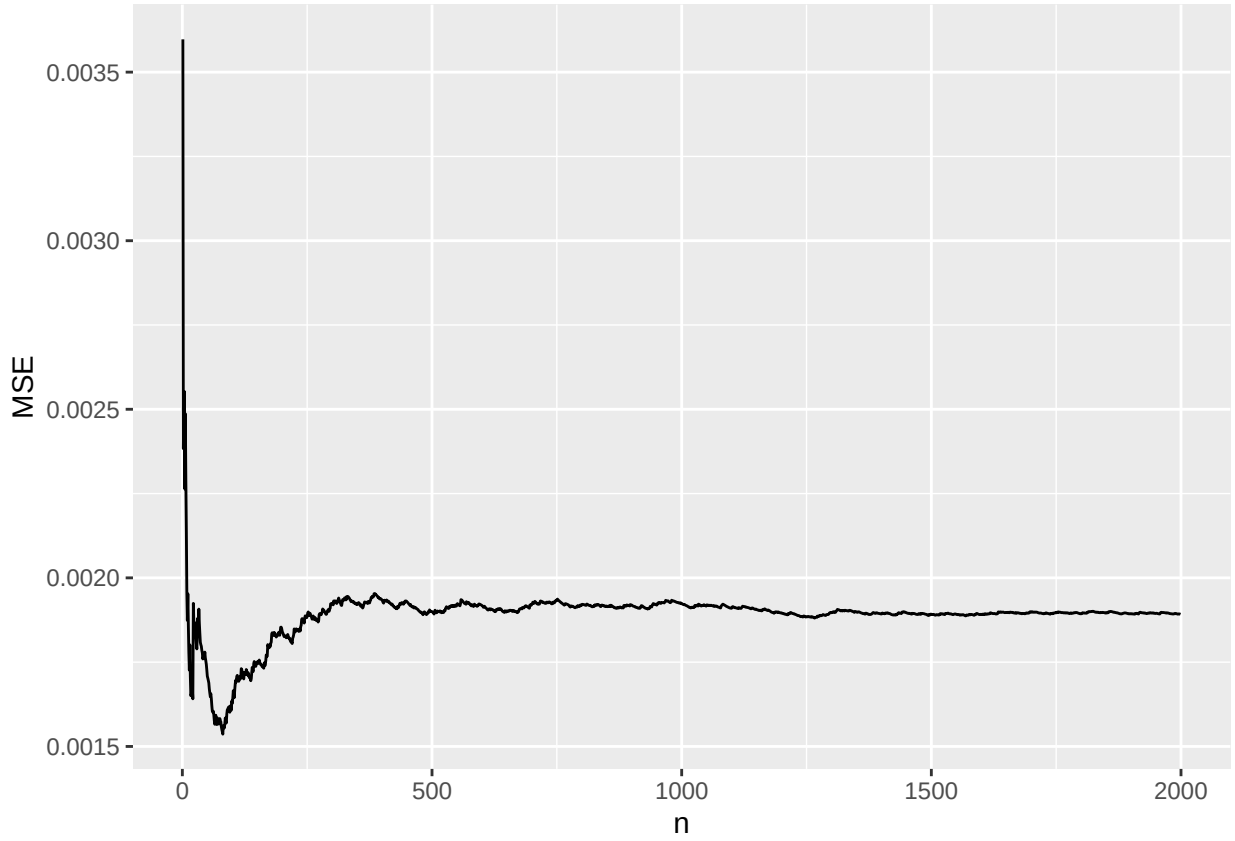
```
# ------------------------------------------------------------#
# NOTE - THE FOLLOWING CODE IS NOT RUN WITHIN THIS DOCUMENT
# ------------------------------------------------------------#

SpatialCV_RF_PON <- Combined_PON %>%
  mutate(bin_lat = cut(Latitude, breaks = seq(55, 57, 0.125)),
         bin_lon = cut(Longitude, breaks = seq(-6, -4, 0.25))) %>%
  mutate(bin_lat = as.character(bin_lat),
         bin_lon = as.character(bin_lon)) %>%
  group_by(bin_lat, bin_lon) %>%
  select(-Longitude, -Latitude, -source) %>%
  select(PON, Mud, MaxShear, MeanShear, DistCoast) %>%
  nest() %>%
```

```
  spatial_R2_RF_PON(df_nested = ., reps = 2000)
```

```
SpatialCV_RF_PON <- fread("OrganicNitrogen_RF_SpatialCrossValidation_final_raw.csv")

SpatialCV_RF_PON %>%
    drop_na() %>%
  mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  mutate(n = row_number()) %>%
  ggplot(aes(y = R2, x = n)) + geom_line()
```




```
SpatialCV_RF_PON %>%
    drop_na() %>%
    mutate_all(function(x){
      cumsum(x)/row_number()}) %>%
  mutate(n = row_number()) %>%
  ggplot(aes(y = MSE, x = n)) + geom_line()
```



9.4 Predictive mapping of seabed organic carbon and nitrogen

We will need to add the relevant environmental predictors to the irregular grid to facilitate predictions - specifically the improved estimates of bed shear stress from this paper.

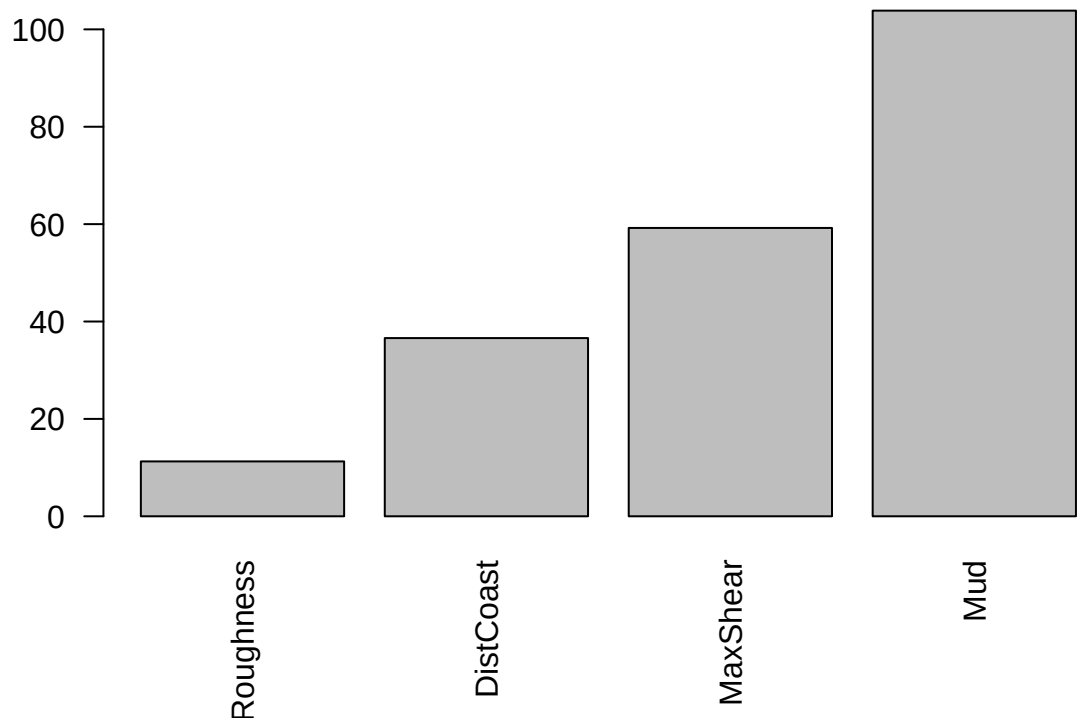
```
## If necessary, re-import Bed Shear Stress data
Clyde_BedShearStress_Annual <- fread(
  "data/Clyde_BedShearStress_IrregularGrid_FINAL_CorrBGSunder60Mud_sMudSandGravel_Annual.csv")

Clyde_IrregularGrid$MeanShear <- interpp(x = Clyde_BedShearStress_Annual$Lon,
                                         y = Clyde_BedShearStress_Annual$Lat,
                                         z = Clyde_BedShearStress_Annual$MeanBedShearStress,
                                         xo = Clyde_IrregularGrid$Longitude,
                                         yo = Clyde_IrregularGrid$Latitude)$z
```

```
## Warning in interpp.old(x, y, z, xo, yo, ncp = 0, extrap = FALSE, duplicate =
## duplicate, : interpp.old() is deprecated, future versions will only provide
## interpp()
```

```
Clyde_IrregularGrid$MaxShear <- interpp(x = Clyde_BedShearStress_Annual$Lon,
                                        y = Clyde_BedShearStress_Annual$Lat,
                                        z = Clyde_BedShearStress_Annual$MaxBedShearStress,
                                        xo = Clyde_IrregularGrid$Longitude,
                                        yo = Clyde_IrregularGrid$Latitude)$z
```

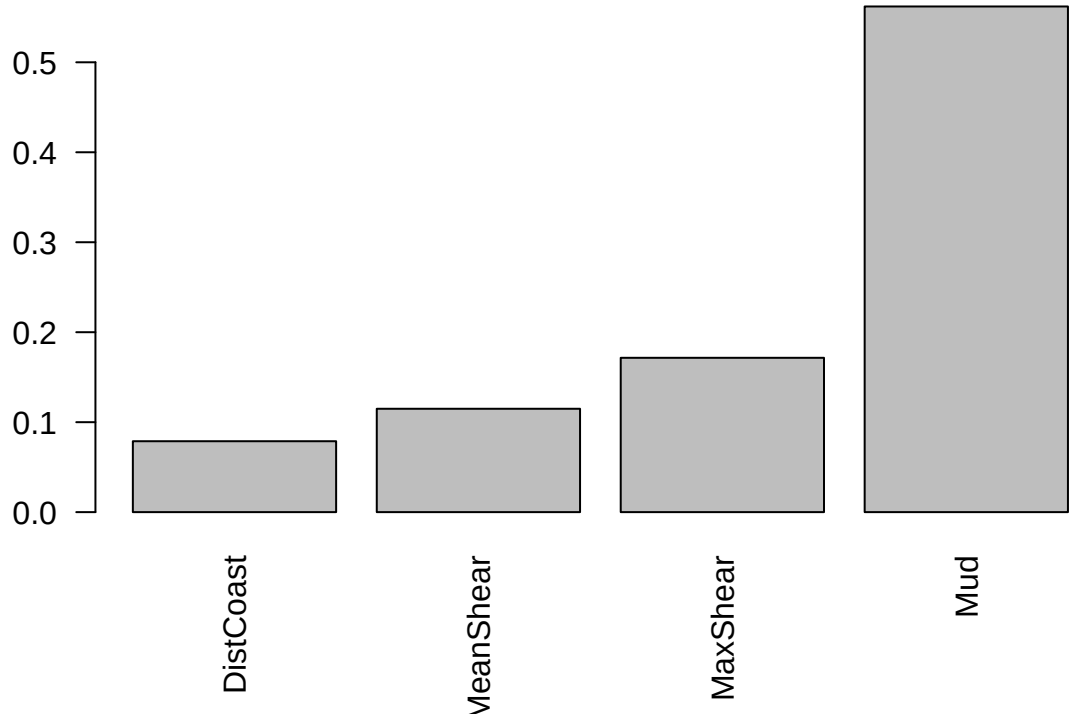
```
## Warning in interpp.old(x, y, z, xo, yo, ncp = 0, extrap = FALSE, duplicate =
## duplicate, : interpp.old() is deprecated, future versions will only provide
```

```
## interpp()
## Rescale distance Variables to km
Clyde_IrregularGrid$DistCoast <- Clyde_IrregularGrid$DistCoast/1000

## Rescale Mud to 0 - 100
Clyde_IrregularGrid$Mud <- Clyde_IrregularGrid$Mud * 100

#' ### Particulate organic carbon

## Fit RF to full dataset
RF_POC <- Combined_POC %>%
  select(POC, Mud, MaxShear, DistCoast, Roughness) %>%
  ranger(POC ~ ., importance = "impurity", data = .)

## plot variable importance
barplot(RF_POC$variable.importance[order(RF_POC$variable.importance)], las = 2)
```




```
## Generate predictive map
Clyde_IrregularGrid$POC <- predict(RF_POC,
                                   data = Clyde_IrregularGrid,
                                   type = "response")$predictions

#' ### Particulate organic nitrogen

## Fit RF to full dataset
RF_PON <- Combined_PON %>%
```

```
  select(PON, Mud, MaxShear, MeanShear, DistCoast) %>%
  ranger(PON ~ ., importance = "impurity", data = .)

## plot variable importance
barplot(RF_PON$variable.importance[order(RF_PON$variable.importance)], las = 2)
```




```
## Generate predictive map
Clyde_IrregularGrid$PON <- predict(RF_PON,
                                   data = Clyde_IrregularGrid,
                                   type = "response")$predictions
```