# Lab 09 - Regression Analysis

## Contents

## Introduction

The objective of this lab assignment is to examine the use of kriging as an interpolation technique. The student learning outcomes for this lab assignment are:

- Examine the nature of spatial autocorrelation in spatial datasets
- Carry out surface interpolation using kriging
- Critically interpret the results from surface interpolation techniques
- Compare and contrast different surface interpolation techniques
- Validation of interpolated surfaces.

## Part A: Exploring Kriging

The goal of this part is to introduce you to geostatistics including exploratory spatial data analysis, structural analysis (calculation and modeling of surface properties of nearby locations), surface prediction and assessment of the results. The instructions are designed to get you acquainted with the concepts and procedures used in Geostatistical Analyst.

You will use spatial data for Vancouver Island, BC to analyze the spatial structure of water acidity, specifically the distribution of pH (the concentration of hydrogen ions) in rivers.

The data can be downloaded from the Ministry of Energy and Mines, British Columbia web site, but has also been provided as RGS_092F.dbf within the van_island folder of the dataset for this lab.

```
van_island <- read.dbf('~/L/GEOG413/lab09/van_island/RGS_092F.dbf')
cat(paste(count(van_island), "records loaded."))
```

```
## 962 records loaded.
```

```
colnames(van_island)
```

```
##  [1] "MASTER_ID" "MAP"       "YEAR"      "ID"        "UTMZ"      "UTME_27"
##  [7] "UTMN_27"   "UTME_83"   "UTMN_83"   "LAT"       "LONG"      "ELEV"
## [13] "MAT"       "REP"       "SRC"       "ODR"       "TYP"       "PHY"
## [19] "DRN"       "CON"       "WDTH"      "DPTH"      "FLW"       "WAT_COL"
## [25] "BNK"       "BNK_PPT"   "COMP"      "SED_COL"   "SED_PPT"   "CHL_BED"
## [31] "CHL_PTN"   "HGTH"      "COLOR"     "HLTH"      "HOST"      "THICK"
## [37] "DATE"      "PH"        "UW"        "FW"        "SO4"       "ZN"
## [43] "CU"        "PB"        "NI"        "CO"        "AG"        "MN"
## [49] "FE"        "MO"        "U"         "W"         "SN"        "HG"
## [55] "AS"        "SB"        "BA"        "CD"        "V"         "BI"
## [61] "CR"        "SE"        "LOI"       "F"         "AU1"       "WT1"
## [67] "AU2"       "WT2"       "AU_NA"     "AU2_NA"    "SB_NA"     "AS_NA"
## [73] "BA_NA"     "BR_NA"     "CE_NA"     "CS_NA"     "CR_NA"     "CO_NA"
## [79] "HF_NA"     "FE_NA"     "LA_NA"     "LU_NA"     "MO_NA"     "NI_NA"
## [85] "RB_NA"     "SM_NA"     "SC_NA"     "NA_NA"     "TA_NA"     "TB_NA"
## [91] "TH_NA"     "W_NA"      "U_NA"      "YB_NA"     "ZR_NA"     "WT_NA"
```

Review the file format.doc that contains the metadata.

---

**Deliverable 1:**

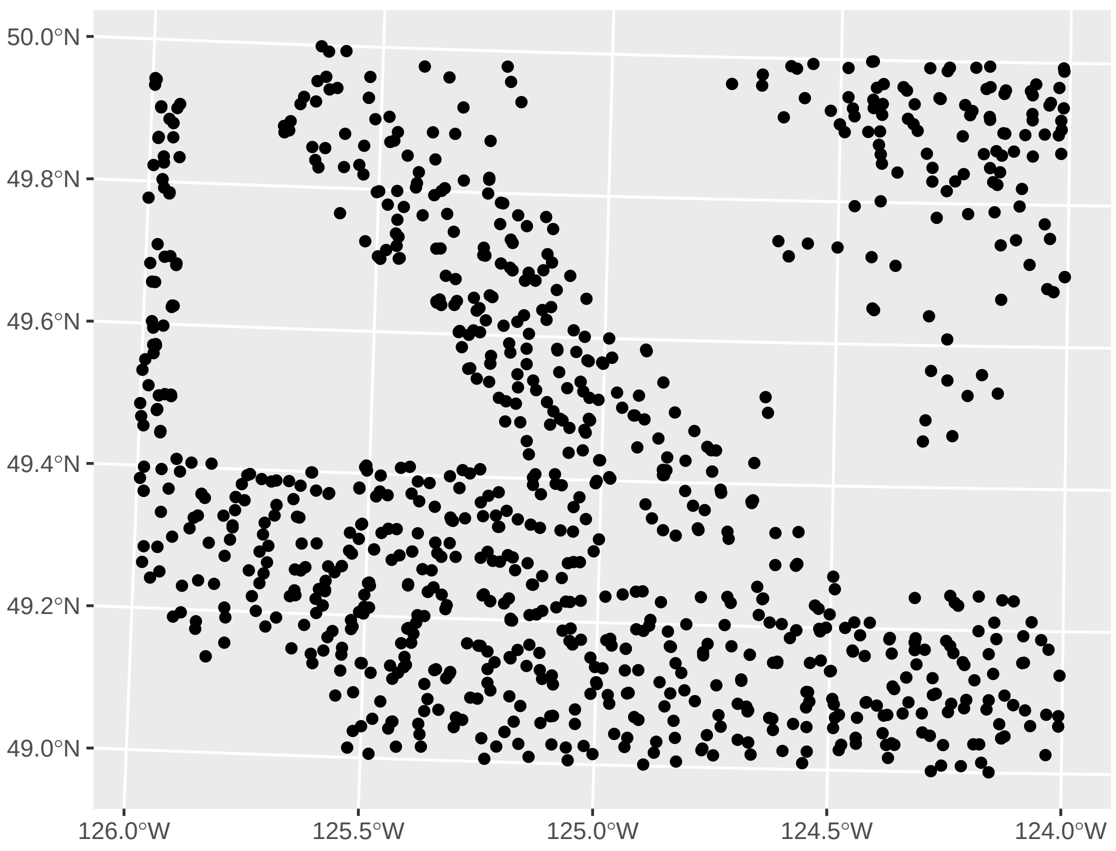
What is the purpose of this dataset, and when was it collected?

---

## Visualize the data

- Convert dataframe to an sf object using the coordinate system NAD 1983 UTM Zone 10N as the coordinate system for your frame. > UTM SRIDS for NAD 1983 can be expressed as 26900 + zone > WGS 84 would be 32600 + zone

```
utmz <- 26900 + van_island$UTMZ[1]
van_island_sf <- van_island %>%
  filter(UTME_83 != '', UTMN_83 != '') %>% # Remove null geometry
  dplyr::select(-UTMZ, -UTME_27, -UTMN_27) %>% # Remove extra coordinates
  #Convert to sf, !!! Assumes all rows have same UTM Zone as 1st row !!!
  st_as_sf(coords = c('UTME_83', 'UTMN_83'), crs = st_crs(utmz))

ggplot(van_island_sf) + geom_sf()
```



- Load the feature class study_area.shp.

```
study_area <- readOGR(dsn = '~/L/GEOG413/lab09/van_island/', layer = 'study_area') %>%
  st_as_sf() %>% st_transform(crs = st_crs(utmz))
```

- Select the records with missing pH data (-1) and delete these from the feature class.

```
van_island_sf_clean <- van_island_sf %>% filter(PH != -1)
```

- Select the points within the study area polygon and export to a new feature class.

```
van_island_study_area <- st_intersection(van_island_sf_clean, study_area)
```

You now have a collection of 728 points within the study area with valid observations for pH. However, some of the observations represent multiple recordings at the exact same monitoring station.

- Use group_by to determine the presence of duplicate locations.

```
# Start of Deliverable 2 code
van_island_study_area %>% group_by(LAT, LONG) %>% summarise(n = n()) %>% arrange(desc(n))
```

---

**Deliverable 2**

How many monitoring stations have more than one observation?

---

**Deliverable 3**

For the pH variable: What are the units of measurement? What is the measurment precision?

---

**Deliverable 4**

Visually inspect the map of sampling locations. Describe how they are distributed in space (e.g., are they uniformly distributed, are all areas sampled).

---

Now it is time to explore and analyze the pH data.

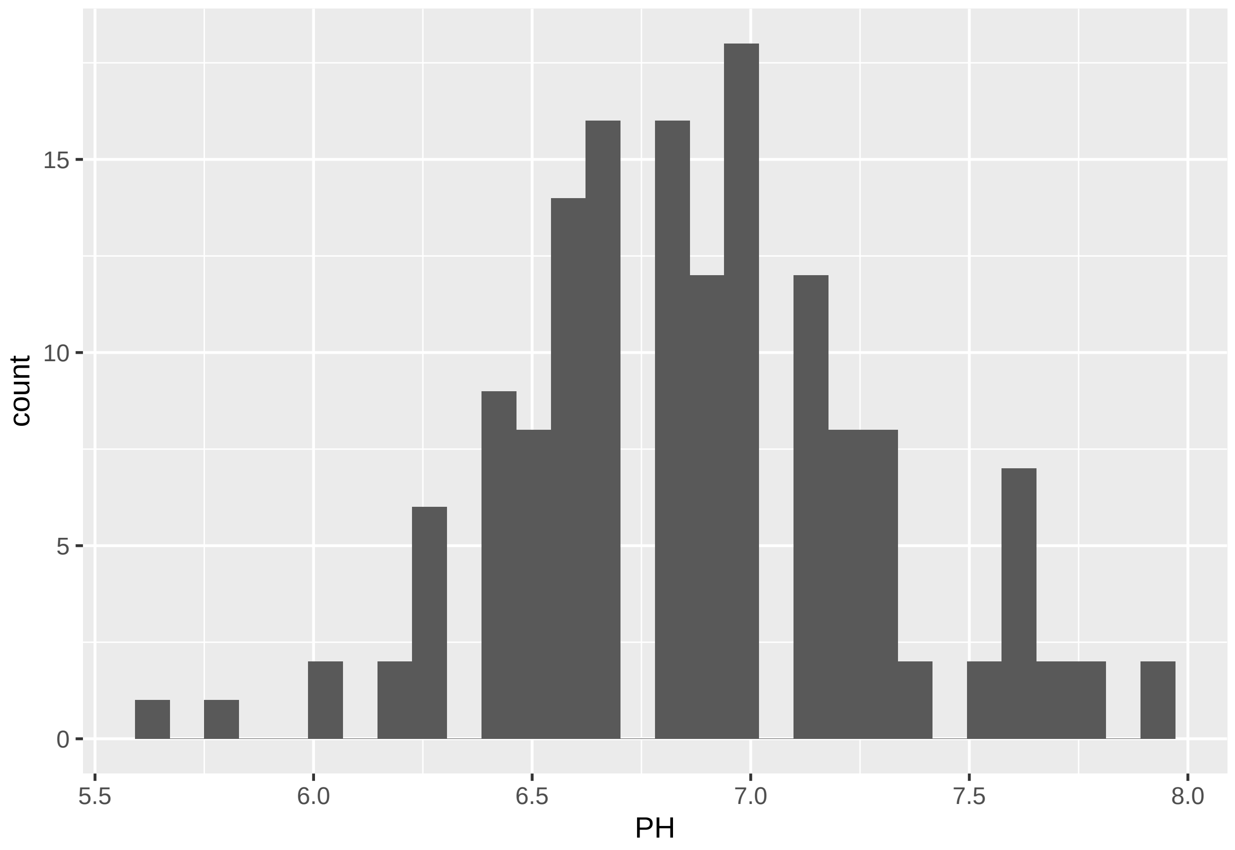
- Remove duplicate values

```
pH_points <- van_island_study_area %>% group_by(LAT, LONG) %>% slice(1) %>% ungroup
pH_points <- pH_points %>% sample_n(min(sub_samp, count(pH_points)[[1]]))
```

- Create a histogram:

```
ggplot(pH_points) + geom_histogram(aes(x = PH))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



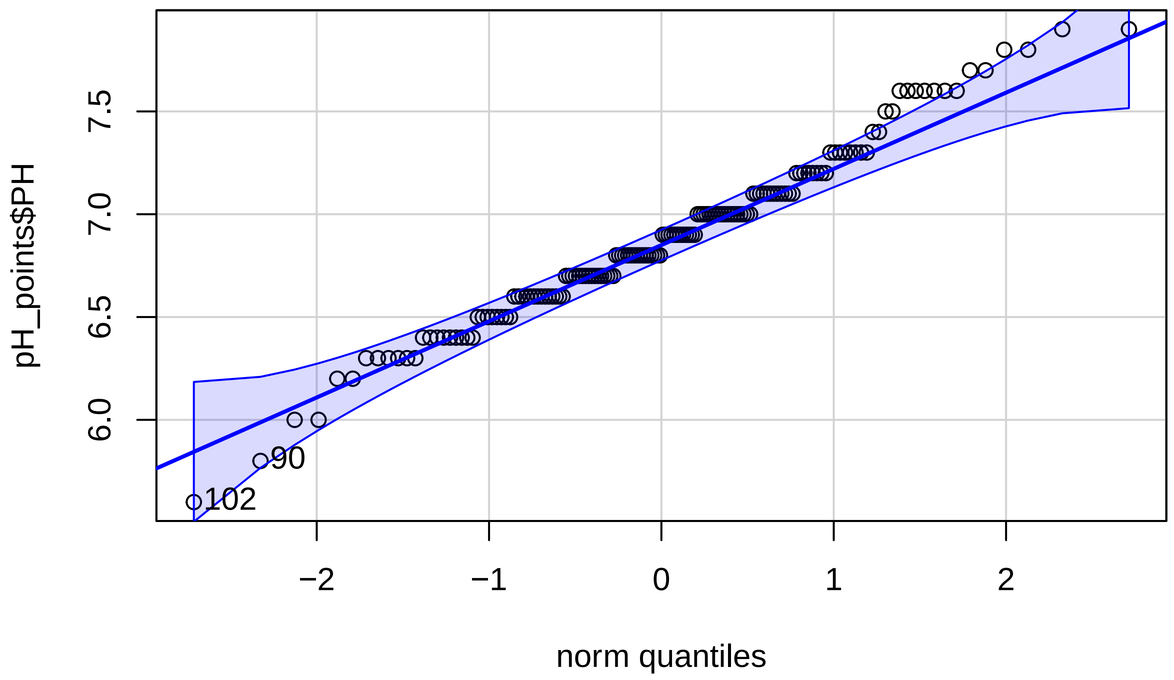
The histogram provides a visual tool to determine whether your data is normally distributed. The indicators of a normal distribution from a histogram are:

- unimodal bell shape
- mean and median values are very close
- kurtosis close to 3

You can also examine normality using a Q-Q plot.

- Create a Q-Q plot:

```
qqPlot(pH_points$PH)
```



```
## [1] 102  90
```

For a normal distribution, the Q-Q plot follows a straight-line. If the data is asymmetric (i.e. far from normal), the points will deviate from the line.

---

**Deliverable 5:**

> Discuss to what extent the data distribution for pH value follows a normal distribution. Use the information from the histogram, descriptive statistics and the Q-Q plot.

## Spatial Auto Correlation

Now it is time to explore the spatial autocorrelation in the pH data. You accomplish this by examining different pairs of sample locations. A semiovariogram is created as follows:

- for a pair of locations, determine the distance between the two locations (called h)

```r
point_distance <- matrix(st_distance(pH_points, pH_points, by_element = FALSE), as.integer(count(pH_poi
```

- for that same pair, determine the difference in values, square this difference (so it is always positive) and then divide by two (called y)

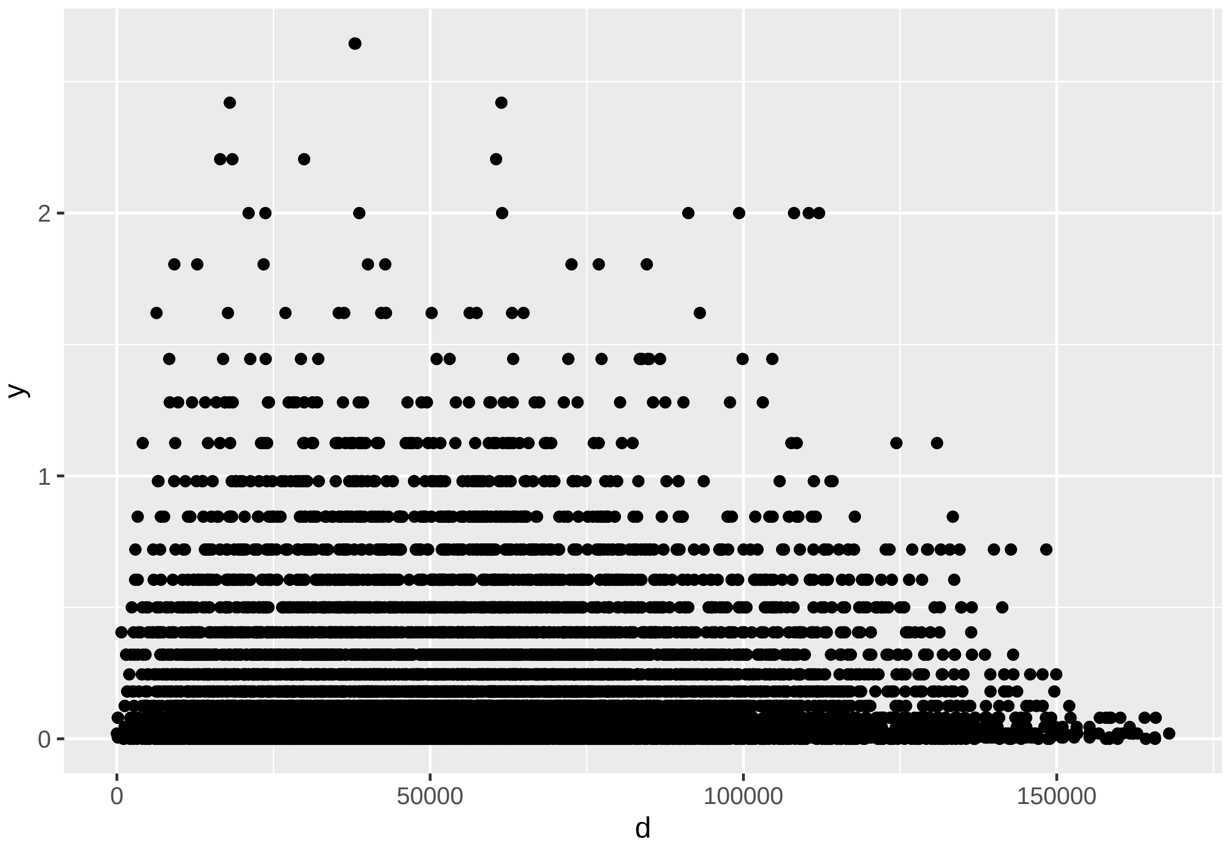
```r
y <- ((pH_points$PH[1] - pH_points$PH[2])^2)/2
```

- repeat this for all possible pairs

```r
# Big-O is n^2!!!

num_points <- count(pH_points)[[1]]

# TODO Can this be CROSS JOIN LATERAL instead of map?
make_map <- function(size) {
  CJ(1:size, 1:size) %>% filter(V1 < V2)
}
map <- make_map(num_points)

num_point_pairs <- as.integer(count(map))

pH_pairs <- data.frame(p1 = integer(), p2 = integer(), d = numeric(), y = numeric())

for (i in 1:num_point_pairs) {
  dist <- point_distance[map$V1[i], map$V2[i]]
  y <- ((pH_points$PH[map$V1[i]] - pH_points$PH[map$V2[i]])^2)/2
  pH_pairs <- rbind(pH_pairs, data.frame(p1 = map$V1[i], p2 = map$V2[i], d = dist, y = y))
  cat("\r", paste0(round(i/num_point_pairs*100, 2), "%       "))
}
```

- create a scatterplot of distance between the locations (X-axis) and half the difference squared

```r
ggplot(pH_pairs, mapping = aes(x = d, y = y)) + geom_point()
```



Each dot in the semivariogram represents a pair of locations, not the individual locations on the map.

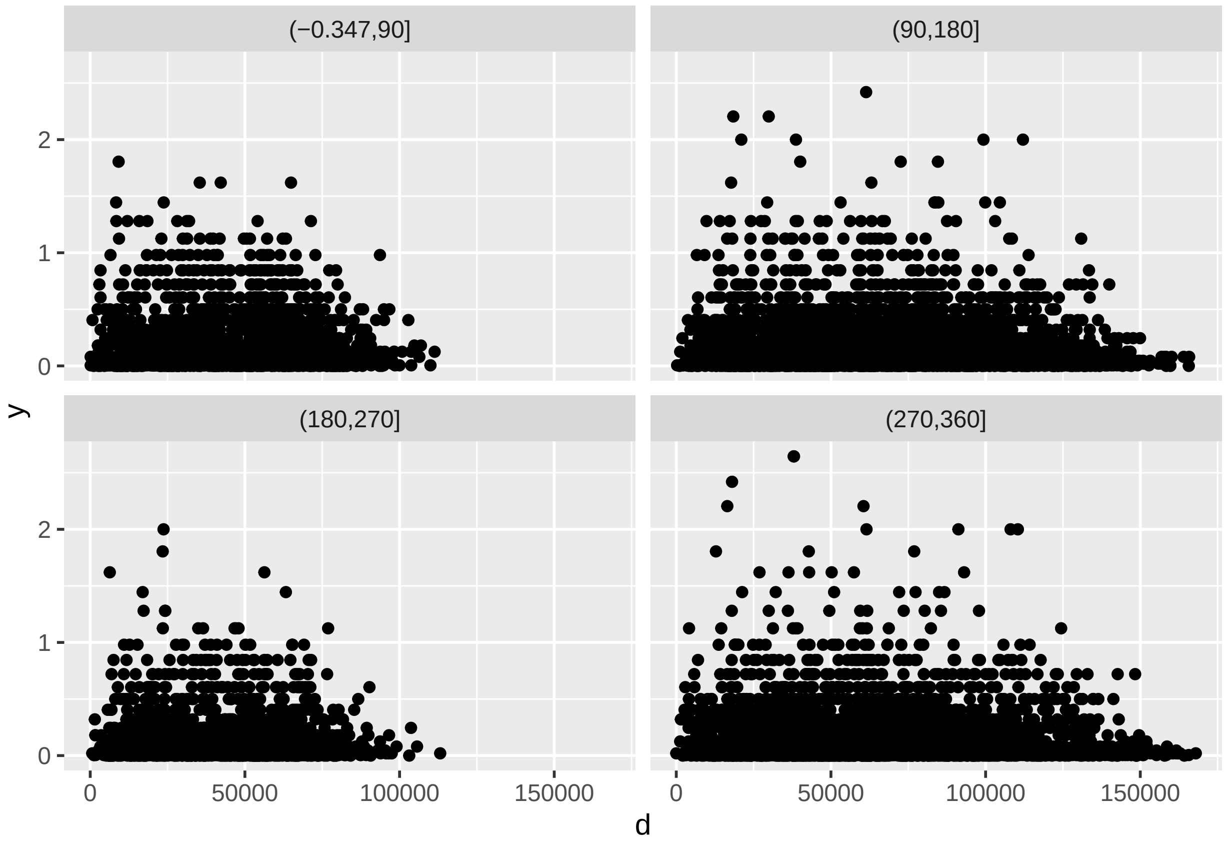
The semivariogram allows you to explore the degree of spatial autocorrelation. If data is spatially dependent, pairs of points that are close together (on the far left of the X-axis) should have fewer differences (or be low on the Y-axis). As points become farther away from each other (moving right on the X-axis), in general, the difference squared should be greater (moving up on the Y-axis). Often there is a certain distance beyond which the squared difference levels out. Pairs of locations beyond this distance are considered uncorrelated.

Spatial autocorrelation may depend only on the distance between two locations, which is called isotropy. However, it is possible that spatial autocorrelation occurs at different distances when considering different directions. Another way to think of this is that things are more alike for longer distances in some directions than in other directions. This directional influence is called anisotropy.

```
x_diff <- st_coordinates(pH_points[map$V1, ])[,'X']-st_coordinates(pH_points[map$V2, ])[,'X']
y_diff <- st_coordinates(pH_points[map$V1, ])[,'Y']-st_coordinates(pH_points[map$V2, ])[,'Y']
dist <- point_distance[map$V1, map$V2]
```

cart2pol crashes Knitter angle<-cart2pol(x_diff, y_diff, degrees = T)

```
pH_pairs_angle <- cbind(pH_pairs, angle[2])
ggplot(pH_pairs_angle %>% mutate(bin = cut(theta, breaks = 4)), mapping = aes(x = d, y = y)) + geom_poi
```

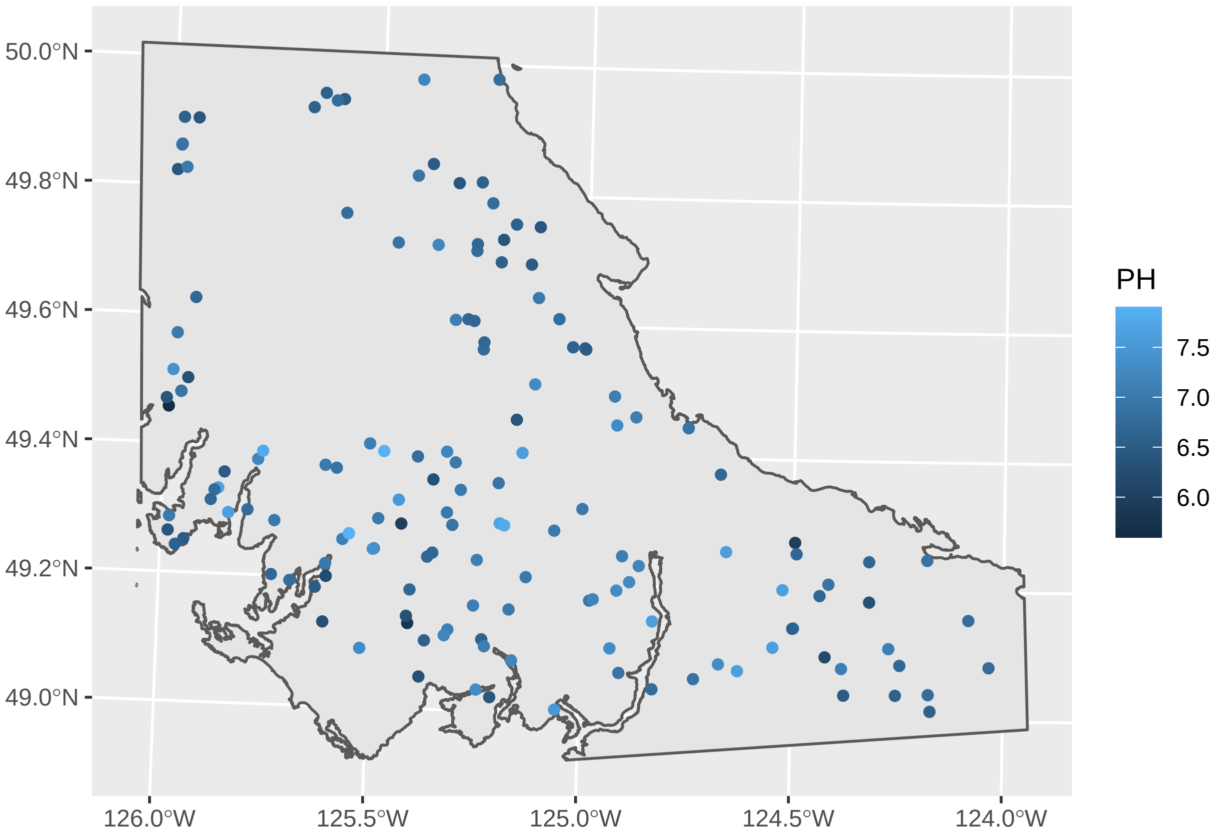


It is important to explore for anisotropy so that you can account for them in the semivariogram. This, in turn, has an effect on the geostatistical prediction method.

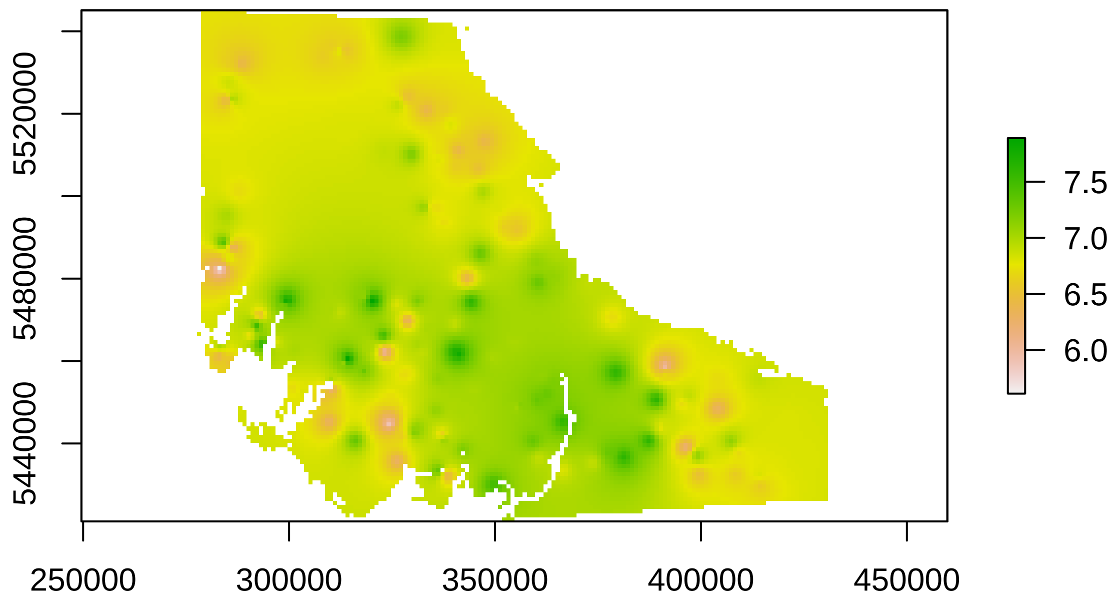
## Inverse Distance Weighting

IDW is an exact, deterministic interpolator with few parameters to set. IDW can be a good way to take a first look at a surface, because no assumptions are made about the data.

```
#Examine Starting Data
ggplot(pH_points) + geom_sf(data = study_area) + geom_sf(aes(col = PH))
```



```
stat <- gstat(formula=PH~1, locations=pH_points, set=list(idp=2))
idw <- interpolate(raster(study_area, res = 1000), stat)
idw_mask <- mask(idw, study_area)
plot(idw_mask)
```



- Notice the RMS of the IDW interpolation, using K folds for testing.

```
idw_rms <- function(points, formula, idp = 2, k = k_folds, ittr = deff_ittr, debug = FALSE){
  rms <- rep(NA, k * ittr)
  for (j in 1:ittr){
    fold <- kfold(nrow(points), k = k)
    for (i in 1:k){
      test_data <- points[fold == i, ]
      train_data <- points[fold != i, ]
      stat <- gstat(formula=formula, locations=train_data, set=list(idp=idp))
      pred <- predict(stat, test_data, debug.level = 0)$var1.pred
      rms[i + ((j-1) * k)] <- sqrt(mean((test_data$PH - pred)^2))
      cat("\r", paste0(round((i + ((j-1) * k))/(k * ittr)*100, 2), "%       "))
    }
  }
  if (debug){
    print(paste('idp:', round(idp, 4), "RMS:", round(mean(rms), 4)))
  }
  round(mean(rms), 4)
}
idw_rms(points = pH_points, formula = PH~1, ittr = deff_ittr, k=k_folds)
```

```
test <- optimize(idw_rms, upper = 5, lower = 1, points=pH_points, formula = PH~1, k = k_folds, ittr=def
```

```
##  25%        50%        75%        100%       [1] "idp: 2.5279 RMS: 0.4204"
##  25%        50%        75%        100%       [1] "idp: 3.4721 RMS: 0.423"
##  25%        50%        75%        100%       [1] "idp: 1.9443 RMS: 0.3845"
##  25%        50%        75%        100%       [1] "idp: 1.5836 RMS: 0.3966"
##  25%        50%        75%        100%       [1] "idp: 1.9305 RMS: 0.3879"
##  25%        50%        75%        100%       [1] "idp: 2.1672 RMS: 0.381"
##  25%        50%        75%        100%       [1] "idp: 2.0637 RMS: 0.3985"
##  25%        50%        75%        100%       [1] "idp: 2.1672 RMS: 0.4175"
##  25%        50%        75%        100%       [1] "idp: 2.1277 RMS: 0.3955"
##  25%        50%        75%        100%       [1] "idp: 2.1521 RMS: 0.4013"
##  25%        50%        75%        100%       [1] "idp: 2.1614 RMS: 0.3854"
##  25%        50%        75%        100%       [1] "idp: 2.165 RMS: 0.3907"
##  25%        50%        75%        100%       [1] "idp: 2.1663 RMS: 0.3921"
##  25%        50%        75%        100%       [1] "idp: 2.1669 RMS: 0.3863"
##  25%        50%        75%        100%       [1] "idp: 2.1671 RMS: 0.4047"
##  25%        50%        75%        100%       [1] "idp: 2.1671 RMS: 0.3892"
##  25%        50%        75%        100%       [1] "idp: 2.1672 RMS: 0.4055"
```

```
print(test)
```

```
## $minimum
## [1] 2.167184
##
## $objective
## [1] 0.4055
```

---

**Deliverable 6:**

> Report the RMSE of the IDW interpolation using an optimized power value and default settings for the other parameters.

---

## Kriging

In Kriging, a predicted value depends on two factors: a trend and an additional element of variability. The trend part of a prediction is simply called the trend. The element of variability is called the spatially-autocorrelated random error. ‘Random’ means that the variability from the trend is not known in advance.

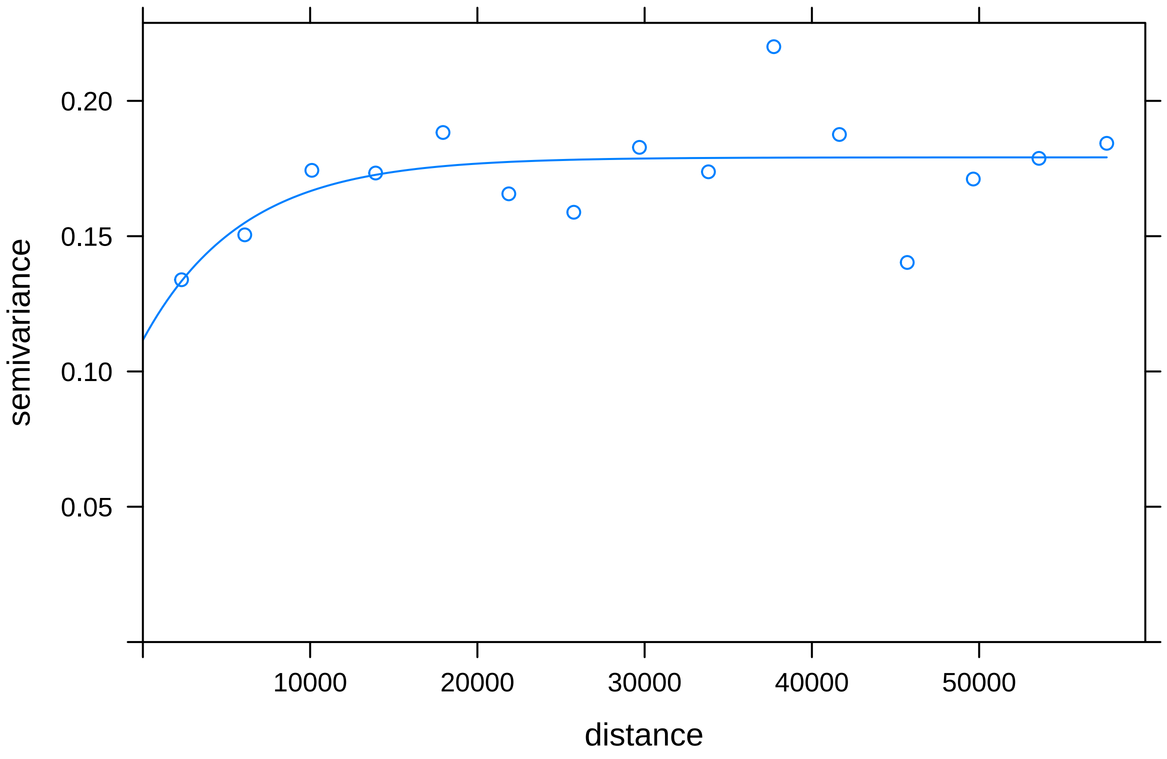
Kriging is a stochastic interpolator. It is very flexible, allows investigation of the spatial autocorrelation in the data, and requires lots of decision-making. It lets you create several kinds of surfaces: prediction, prediction standard error, probability, and quantile. Kriging assumes the data come from a stationary process; some methods require that the data are normally distributed.

Ordinary kriging assumes of model of the form Z(s) = m + e(s), where m is an unknown constant (trend) and e(s) represents the random variability. One of the main issues concerning ordinary kriging is whether the assumption of a constant mean is reasonable. Sometimes there are good scientific reasons to reject this assumption. However, as a simple prediction method, it has remarkable flexibility.

Create an Ordinary Kriging prediction surface: Set Calculate Nugget to True, set Model Type to Exponential and set Lag Size (range) to 1000 meters.

```
# Determine Bounds
coords <- st_coordinates(pH_points)
min_x <- min(coords[, 'X'])
max_x <- max(coords[,'X'])
min_y <- min(coords[, 'Y'])
max_y <- max(coords[,'Y'])
```

```r
# Create exmpty spatial raster for results
width <- seq(min_x, min_y, length.out = grid_dim)
height <- seq(min_y, max_y, length.out = grid_dim)
pH_grid <- expand.grid(x = width, y = height)
pH_grid_sf <- st_as_sf(pH_grid, coords = c('x', 'y'), crs = st_crs(pH_points))



# Make variogram for each variable
ph.vgm <- variogram(PH~1, pH_points)
ph.fit <- fit.variogram(ph.vgm, model = vgm(1, model = 'Exp', nugget = TRUE, range = 1000))

# Plot variogram
plot(ph.vgm, ph.fit)
```



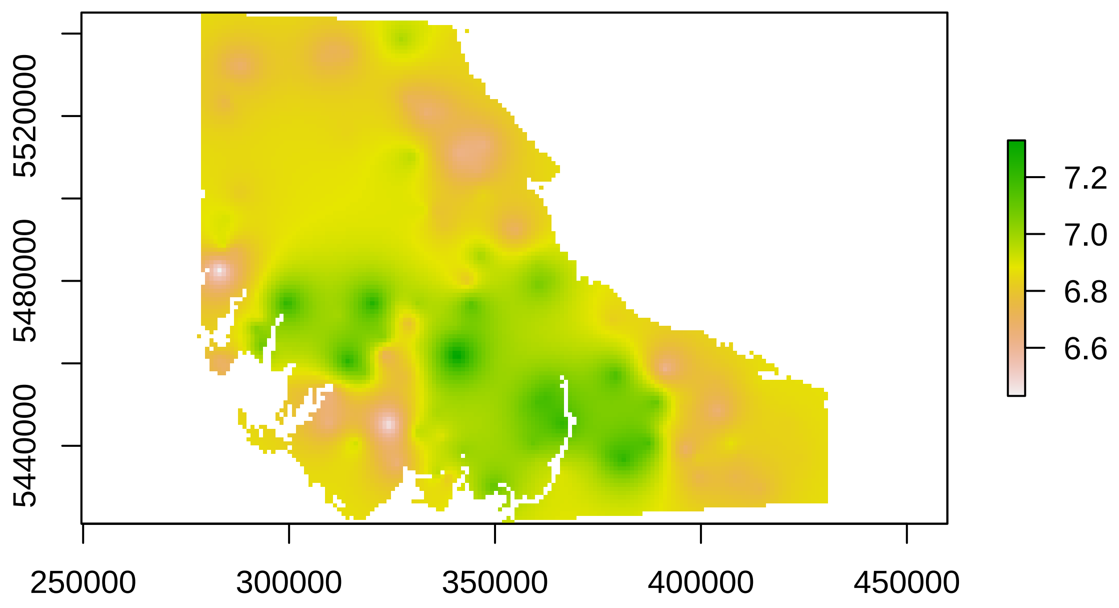
```r
# Make gstat
g <- gstat(NULL, id = "PH", form = PH~1, model=ph.fit, data=pH_points)

# Interpolate pH onto new grid
krig <- predict(g, pH_grid_sf)
```

```
## [using ordinary kriging]
```

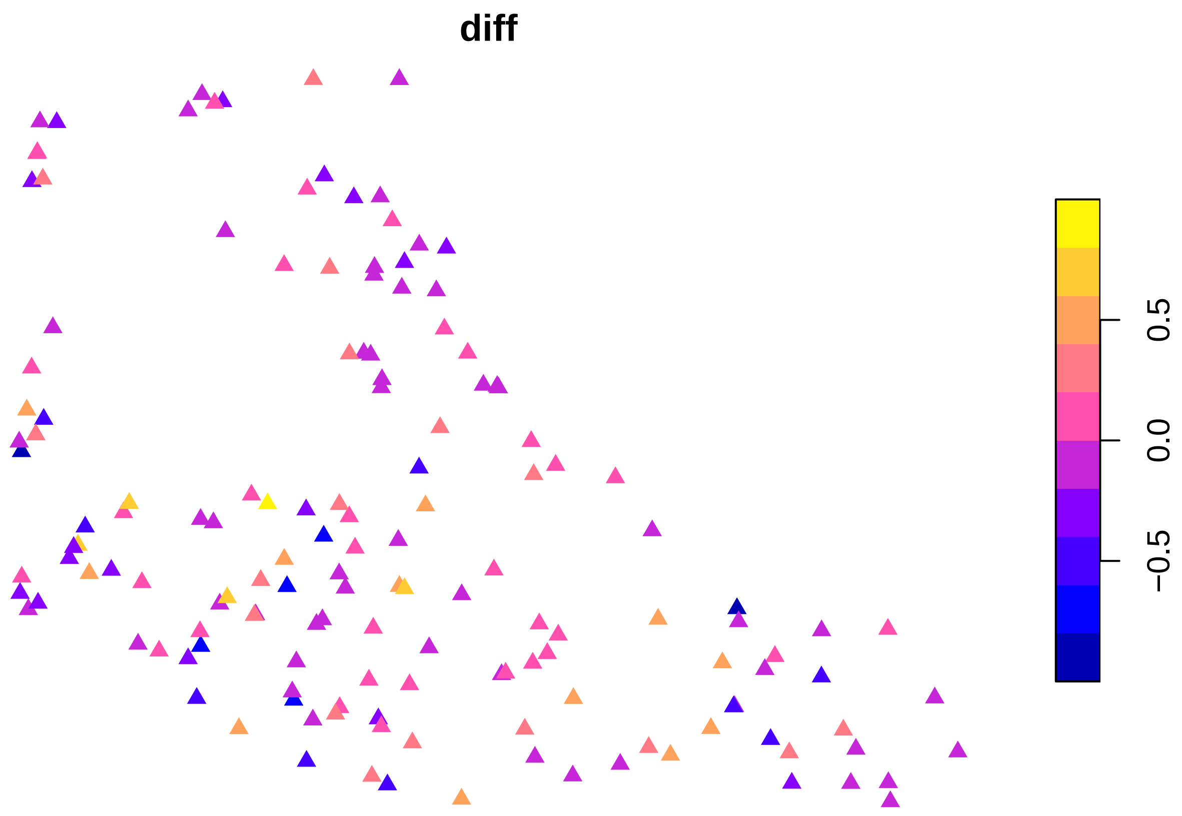
```r
# Rasterize Results
krig_ras <- interpolate(raster(study_area, res = 1000), g)
```

```
## [using ordinary kriging]
```

```
# Mask and plot Raster
krig_mask <-mask(krig_ras$PH.pred, study_area)
plot(krig_mask)
```



```
# Calculate Error
diff = st_join(pH_points, krig, join = st_nearest_feature) %>%
    mutate(diff = PH-PH.pred) %>% dplyr::select(diff)
plot(diff, pch = 17)
```



```r
cat("FRM: ", sqrt(mean(diff$diff^2)))
```

```
## FRM:  0.3127865
```

The semivariogram uses a particular function that fits the distribution of semivariances (or dissimilarity) of data points to the distances that separate them. Lag size is the straight line (vector) that separates any two locations. A lag has length (distance) and direction (orientation). The lag size is the size of a distance class into which pairs of locations are grouped in order to reduce the large number of possible combinations. This is called binning. The lag size controls the size of the bins. Bins are a classification of lags, where all lags that have similar distance and direction are put into the same bin. Bins are commonly formed by a grid or a sector method. In Geostatistical Analyst the empirical semivariogram value in each bin is color-coded and is shown on the semivariogram/covariance surface and graph.

The selection of a lag size has important effects on the empirical semivariogram. For example, if the lag size is too large, short-range autocorrelation may be masked. If the lag size is too small, there may be many empty bins, and sample sizes within bins will be too small to get representative "averages" for bins.

When samples are located on a sampling grid, the grid spacing is usually a good indicator for lag size. However, if the data are acquired using an irregular or random sampling scheme, the selection of a suitable lag size is not so straightforward. A rule of thumb is to multiply the lag size by the number of lags, which should be about half the largest distance among all points. In addition, if the range of the fitted semivariogram model is very small relative to the extent of the empirical semivariogram, you can decrease the lag size. Conversely, if the range of the fitted semivariogram model is large relative to the extent of the empirical semivariogram, you can increase the lag size.

```r
k_rms <- function(points, formula, k = k_folds, ran = 1000, ittr = deff_ittr, debug = FALSE) {
  rms <- rep(NA, k * ittr)
  for (j in 1:ittr){
    fold <- kfold(nrow(points), k = k)
    for (i in 1:k){
      train_data <- points[fold != i, ]
      test_data <- points[fold == i, ]

      width <- seq(min(coords[, 'X']), max(coords[,'X']), length.out = grid_dim)
      height <- seq(min(coords[, 'Y']), max(coords[,'Y']), length.out = grid_dim)
      train_grid <- expand.grid(x = width, y = height)
      train_grid_sf <- st_as_sf(pH_grid, coords = c('x', 'y'), crs = st_crs(pH_points))

      train.vgm <- variogram(formula, train_data)
      train.fit <- tryCatch(fit.variogram(train.vgm, model = vgm(1, model = 'Exp', nugget = TRUE, range
      if (length(train.fit) > 1) {
        test_k <- krige((PH) ~ 1, train_data, train_grid_sf, model=train.fit, debug.level = 0)
        diff = st_join(test_data, test_k, join = st_nearest_feature) %>%
          mutate(diff = PH-var1.pred) %>% select(diff)
        rms <- sqrt(mean(diff$diff^2))
        rms[i + ((j-1) * k)] <- rms
      } else {
        rms[i + ((j-1) * k)] <- NA
      }
      cat("\r", paste0(round((i + ((j-1) * k))/(k * ittr)*100, 2), "%       "))
    }
  }
  if (debug){
    print(paste('Range:', round(ran, 4), "RMS:", round(mean(rms, na.rm=TRUE), 4)))
  }
  round(mean(rms, na.rm=TRUE), 4)
}
k_rms(points = pH_points, formula = PH~1, ran = 1000, k = k_folds, ittr = deff_ittr)
```

```
##  25%        50%        75%        100%
```

```
## [1] 0.4373
```

```r
#test <- optimize(k_rms, upper = 1500, lower = 250, points=pH_points, formula = PH~1, k = k_folds, ittr
#print(test)
```

---

**Deliverable 7:**

> Did the Kriging model improve upon the earlier IDW models, even without considering anisotropy? Try to explain where the improvement (if any) comes from.

---

Now it is time to examine the effect of anisotropy

- Run the Geostatistical Wizard again and create another kriging model.
- Keep all parameters the same but on Step 2 of 5 use the following:
- Set Anisotropy to true, set Calculate Partial to true, and set Direction to 320.

```r
k_rms_anis <- function(points, formula, k = k_folds, ran = 1000, ittr = deff_ittr, debug = FALSE) {
  rms <- rep(NA, k * ittr)
  for (j in 1:ittr){
    fold <- kfold(nrow(points), k = k)
    for (i in 1:k){
      train_data <- points[fold != i, ]
      test_data <- points[fold == i, ]

      width <- seq(min(coords[, 'X']), max(coords[,'X']), length.out = grid_dim)
      height <- seq(min(coords[, 'Y']), max(coords[,'Y']), length.out = grid_dim)
      train_grid <- expand.grid(x = width, y = height)
      train_grid_sf <- st_as_sf(pH_grid, coords = c('x', 'y'), crs = st_crs(pH_points))

      train.vgm <- variogram(formula, train_data)
      train.fit <- tryCatch(fit.variogram(train.vgm, model = vgm(1, model = 'Exp', nugget = TRUE, range
      if (length(train.fit) > 1) {
        test_k <- krige((PH) ~ 1, train_data, train_grid_sf, model=train.fit, debug.level = 0)
        diff = st_join(test_data, test_k, join = st_nearest_feature) %>%
          mutate(diff = PH-var1.pred) %>% select(diff)
        rms <- sqrt(mean(diff$diff^2))
        rms[i + ((j-1) * k)] <- rms
      } else {
        rms[i + ((j-1) * k)] <- NA
      }
      cat("\r", paste0(round((i + ((j-1) * k))/(k * ittr)*100, 2), "%        "))
    }
  }
  if (debug){
    print(paste('Range:', round(ran, 4), "RMS:", round(mean(rms), 4)))
  }
  round(mean(rms, na.rm = TRUE), 4)
}

k_rms_anis(pH_points, formula = PH~1, ran = 1000, ittr = deff_ittr, k = k_folds)
```

```
##  25%         50%         75%         100%
```

```
## [1] 0.3977
```

---

**Deliverable 8:**

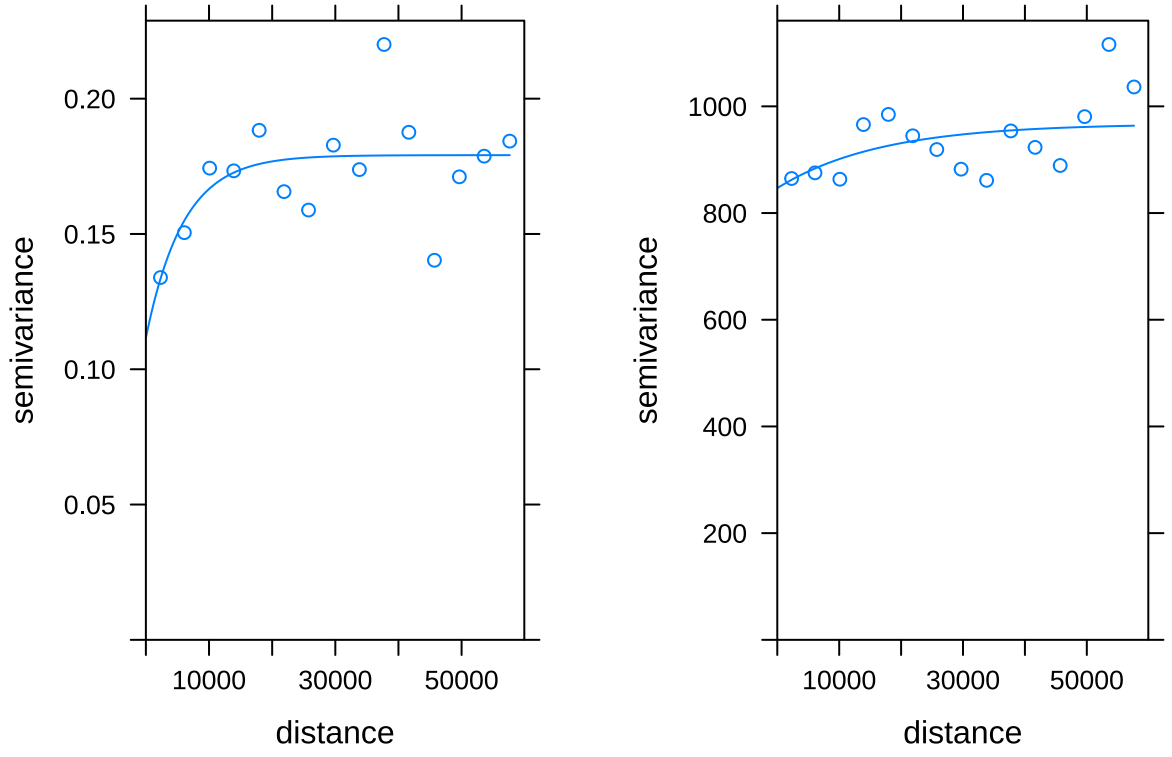
Did anisotropy improve the kriging model? Briefly explain your answer.

---

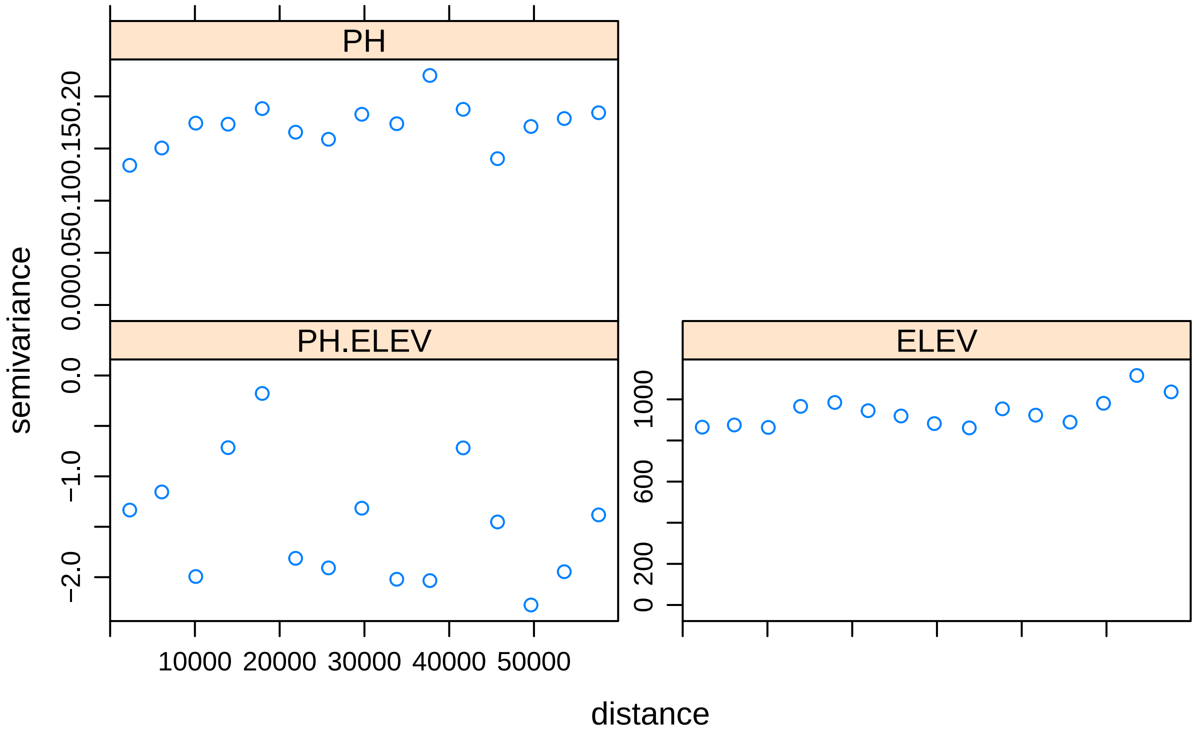
## Use CoKriging

There are many different types of kriging. Here we will examine the use of cokriging. Like kriging, cokriging is a stochastic interpolator that is exact if the data has no measurement error, and smooth if it has measurement error. It can use information from multiple data sets.

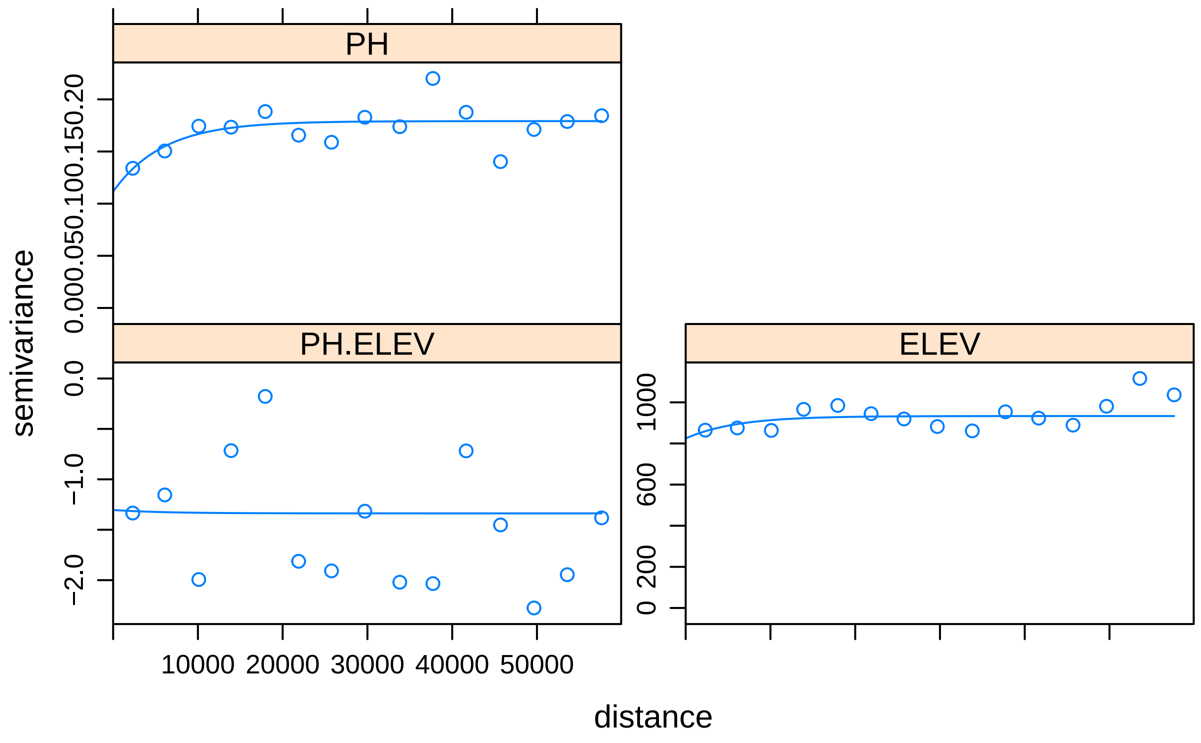
```r
# Determine Bounds
coords <- st_coordinates(pH_points)
min_x <- min(coords[, 'X'])
max_x <- max(coords[,'X'])
min_y <- min(coords[, 'Y'])
max_y <- max(coords[,'Y'])

# Create exmpty spatial raster for results
width <- seq(min_x, min_y, length.out = grid_dim)
height <- seq(min_y, max_y, length.out = grid_dim)
pH_grid <- expand.grid(x = width, y = height)
pH_grid_sf <- st_as_sf(pH_grid, coords = c('x', 'y'), crs = st_crs(pH_points))

# Make variogram for each variable
ph.vgm <- variogram(PH~1, pH_points)
ph.fit <- fit.variogram(ph.vgm, model = vgm(1, model = 'Exp', nugget = TRUE, range = 1000))
elev.vgm <- variogram(ELEV~1, pH_points)
elev.fit <- fit.variogram(elev.vgm, model = vgm(1, model = 'Exp', nugget = TRUE))

# Plot variograms
ph.plt <- plot(ph.vgm, ph.fit)
elev.plt <- plot(elev.vgm, elev.fit)
grid.arrange(ph.plt, elev.plt, ncol = 2)
```



```
# Make Cross variogram
g <- gstat(NULL, id = "PH", form = PH~1, data=pH_points)
g <- gstat(g, id = "ELEV", form = ELEV~1, data=pH_points)
v.cross <- variogram(g)

# View Cross variogram
plot(v.cross)
```



```
# Fit variogram
g <- gstat(g, id = "PH", model = ph.fit, fill.all=T)
g <- fit.lmc(v.cross, g)
plot(variogram(g), model=g$model)
```



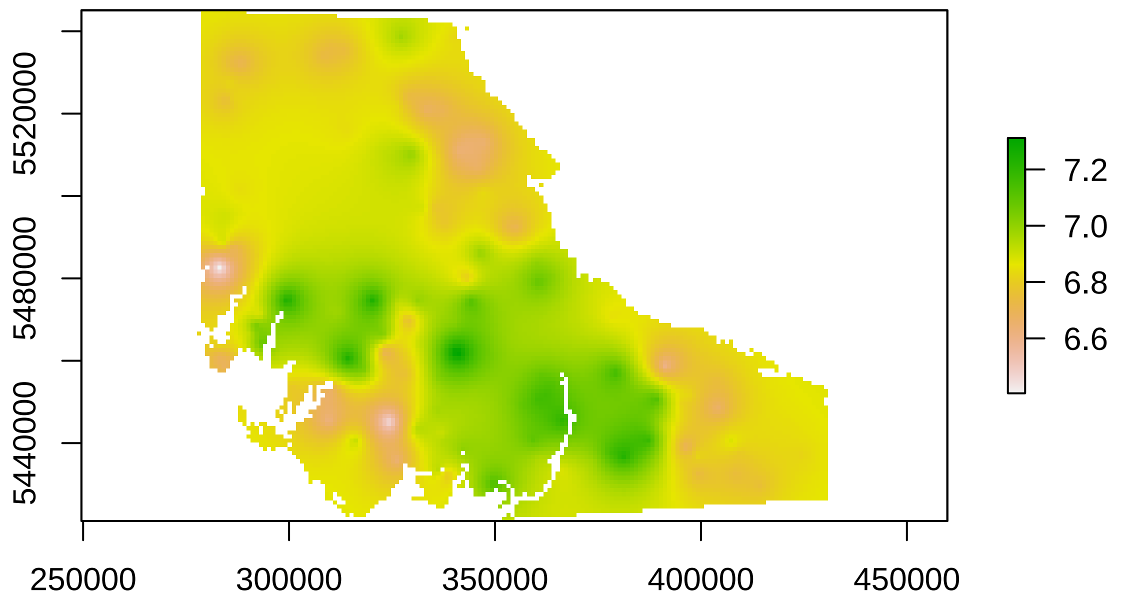
```r
# Interpolate pH onto new grid
krig <- predict(g, pH_grid_sf)
```

```
## Linear Model of Coregionalization found. Good.
## [using ordinary cokriging]
```

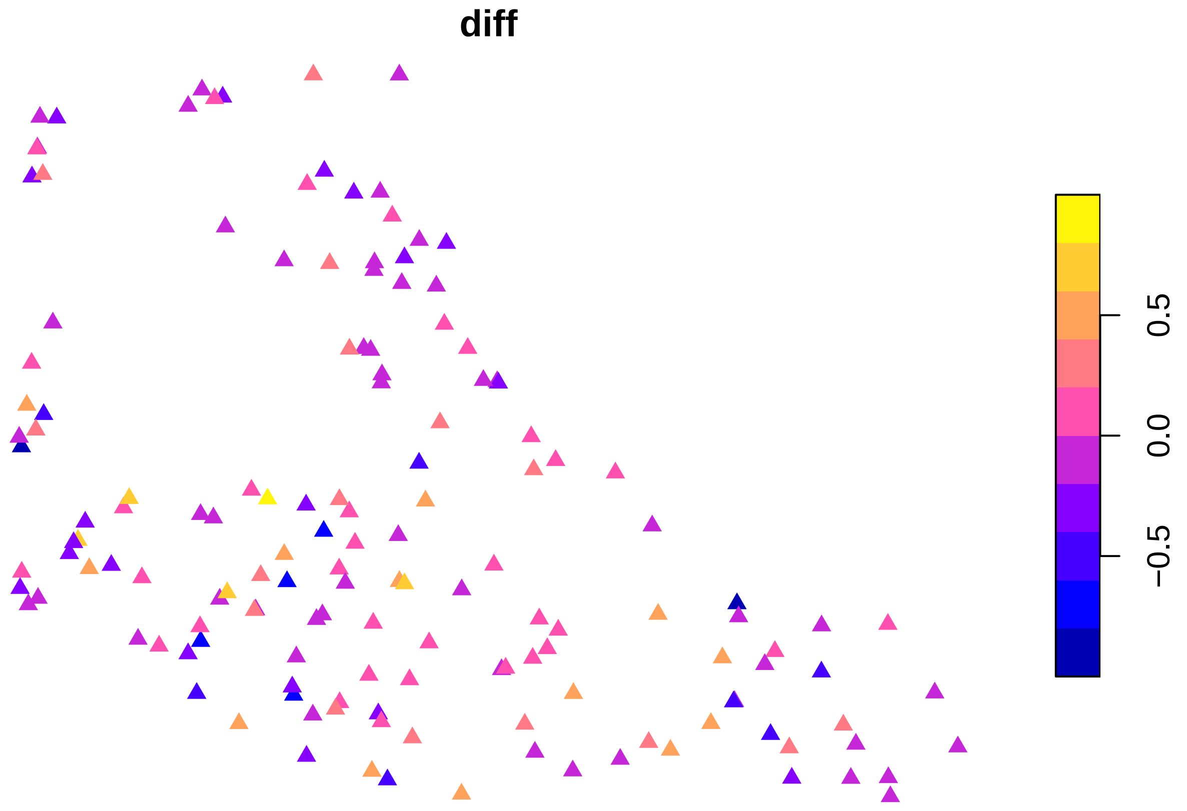
```r
# Rasterize Results
krig_ras <- interpolate(raster(study_area, res = 1000), g)
```

```
## Linear Model of Coregionalization found. Good.
## [using ordinary cokriging]
```

```r
# Mask and plot Raster
krig_mask <-mask(krig_ras$PH.pred, study_area)
plot(krig_mask)
```



```
# Calculate Error
diff = st_join(pH_points, krig, join = st_nearest_feature) %>%
    mutate(diff = PH-PH.pred) %>% dplyr::select(diff)
plot(diff, pch = 17)
```



```
cat("FRM: ", sqrt(mean(diff$diff^2)))
```

```
## FRM:  0.3129141
```

**Deliverable 9:**

Did using elevation data result in an improved prediction surface? Briefly explain your answer.