Assessing landscape heterogeneity in Oregon State

Annie C. Smith

September 5, 2019

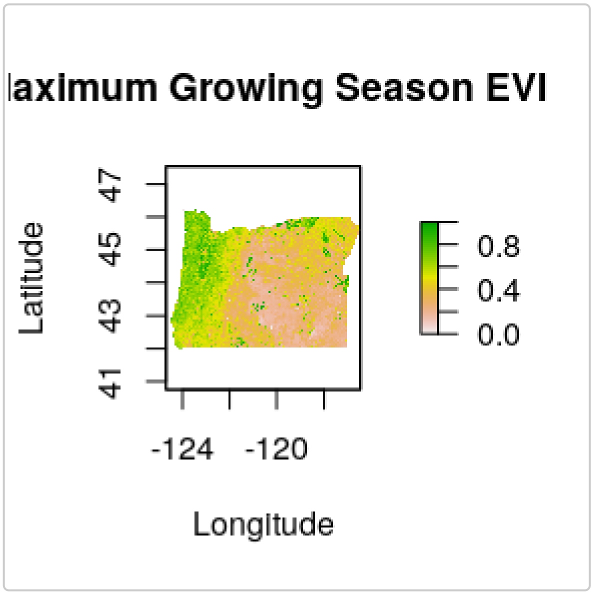
First, import *geodiv* and any other necessary packages.

```
library(geodiv)
#>
#> Attaching package: 'geodiv'
#> The following object is masked from 'package:utils':
#>
#>     str
library(raster)
#> Loading required package: sp
library(mapdata)
#> Loading required package: maps
library(maptools)
#> Checking rgeos availability: TRUE
library(rgeos)
#> rgeos version: 0.5-1, (SVN revision 614)
#>  GEOS runtime version: 3.6.2-CAPI-1.10.2
#>  Linking to sp version: 1.3-1
#>  Polygon checking: TRUE
library(ggplot2)
#> Registered S3 methods overwritten by 'ggplot2':
#>   method         from
#>   [.quosures     rlang
#>   c.quosures     rlang
#>   print.quosures rlang
```

Now, open and plot a raster of the Enhanced Vegetation Index (EVI) for Oregon State.

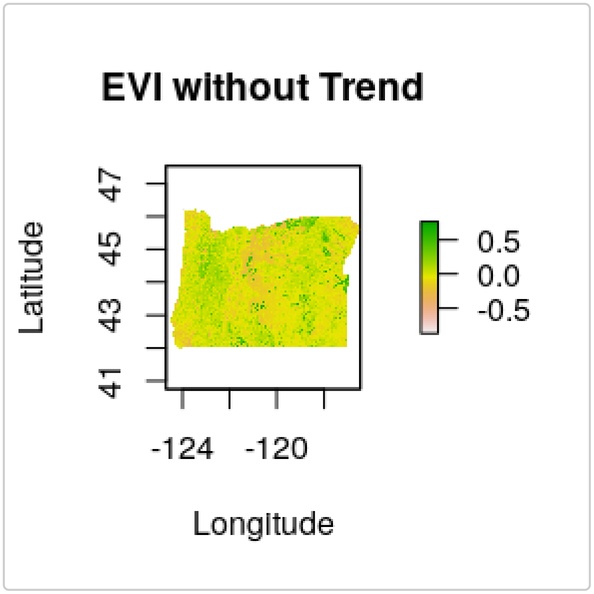
```
# import EVI raster into R
data(oregonEVI)

# look up scalar and apply to data
?oregonEVI
oregonEVI <- oregonEVI * 0.0001

# mask any values that are outside of the state bounds
oregon <- map(database = 'state', regions = 'oregon', fill = TRUE, plot = FALSE)
oregonPoly <- map2SpatialPolygons(oregon, IDs = oregon$names, proj4string =
CRS(proj4string(oregonEVI)))
oregonEVI <- mask(x = oregonEVI, mask = oregonPoly)
```

```
# plot maximum growing season EVI for Oregon
plot(oregonEVI, xlab = 'Longitude', ylab = 'Latitude', main = 'Maximum Growing Season EVI')
```



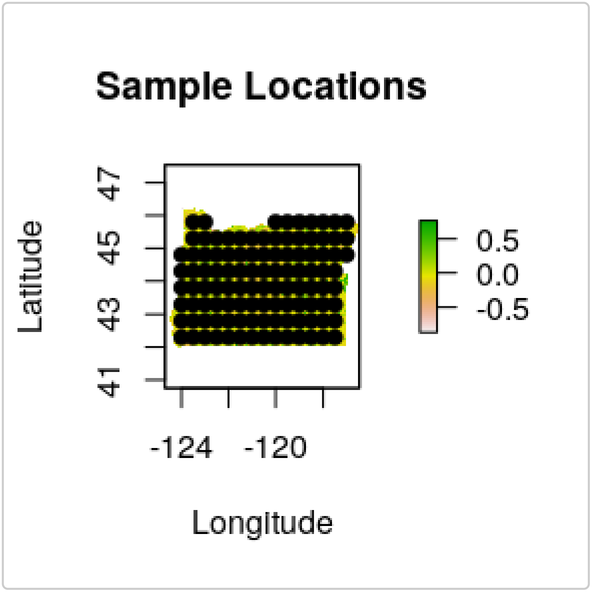
We now want to remove any trend that there may be over the entire raster, because we are only interested in the heterogeneity, not the overall values. This is made easy by a function included in *geodiv*.

```
# remove overall trend in values over raster (this removes the best-fit polynomial plane)
evi <- remove_plane(oregonEVI)
#> [1] "Order of polynomial that minimizes errors: 1"

# plot again to see what the new raster looks like
plot(evi, xlab = 'Longitude', ylab = 'Latitude', main = 'EVI without Trend')
```



Now that we have a clean raster, we need to sample locations at which to calculate metrics. We will select uniformly-distributed points across the state to get a good representation of all ecosystems.

```
# create a grid of points across the state with 1/2 degree spacing
pt_grid <- makegrid(oregonPoly, cellsize = 0.5)
pt_grid <- SpatialPoints(pt_grid, proj4string = CRS(proj4string(evi)))

# cut to state outline
pt_grid <- gIntersection(pt_grid, oregonPoly)

# convert to sample data frame
samp_pts <- as.data.frame(coordinates(pt_grid))

# let's take a look in table form
head(samp_pts)
#>          x    y
#> X1    -124 42.3
#> X1.1 -124 42.8
#> X1.2 -124 43.3
#> X1.3 -124 43.8
#> X1.4 -124 44.3
#> X1.5 -124 44.8

# where are the points located on the map?
plot(evi, xlab = 'Longitude', ylab = 'Latitude', main = 'Sample Locations')
points(x = samp_pts$x, y = samp_pts$y, pch = 19)
```



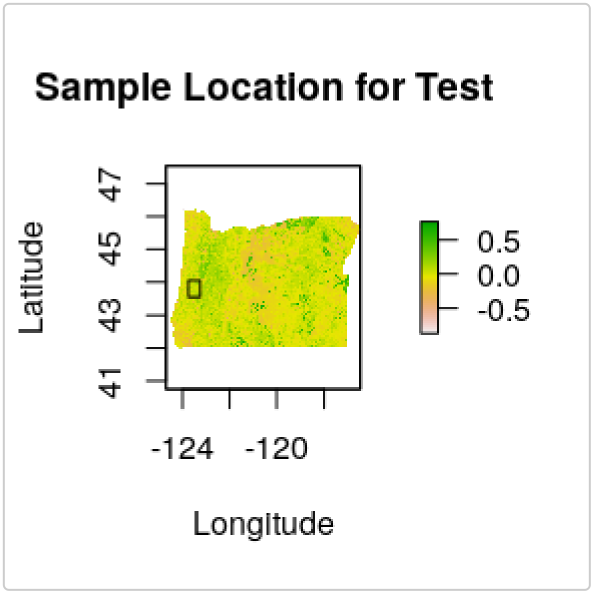
```
# how many points?
nrow(samp_pts)
#> [1] 108
```

This results in 108 sample points. At each of these points, we will calculate all surface metrics. First, we'll do this for a single point.

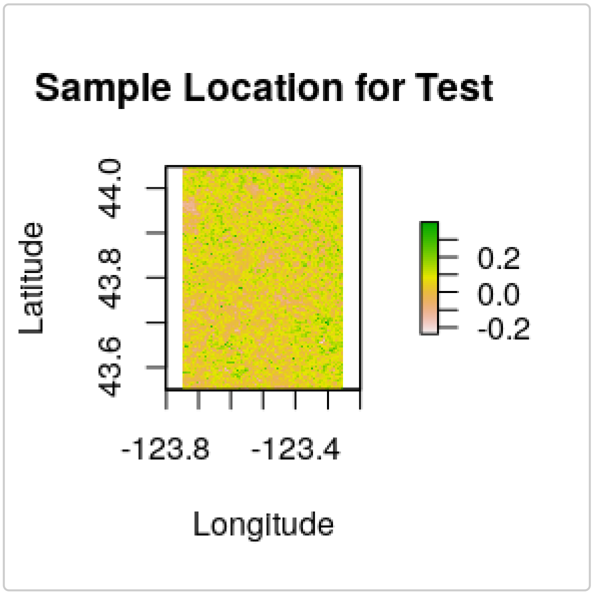
```r
# create function to run all functions and output results
run_all <- function(r){

  # run all functions
  cat('Sa: ', '\n')
  sa <- sa(r)
  print(sa)
  cat('\n')

  cat('Sq: ', '\n')
  sq <- sq(r)
  print(sq)
  cat('\n')

  cat('S10z: ', '\n')
  s10z <-s10z(r)
  print(s10z)
  cat('\n')

  cat('Sdq: ', '\n')
  sdq <- sdq(r)
  print(sdq)
  cat('\n')

  cat('Sdq6: ', '\n')
  sdq6 <- sdq6(r)
  print(sdq6)
  cat('\n')

  cat('Sdr: ', '\n')
  sdr <- sdr(r)
  print(sdr)
  cat('\n')

  cat('Sbi: ', '\n')
  sbi <- sbi(r)
  print(sbi)
  cat('\n')

  cat('Sci: ', '\n')
  sci <- sci(r)
  print(sci)
  cat('\n')

  cat('Ssk (Adjacent): ', '\n')
  ssk_adj <- ssk(r, adj = TRUE)
  print(ssk_adj)
  cat('\n')

  cat('Sku (excess): ', '\n')
  sku_excess <- sku(r, excess = TRUE)
  print(sku_excess)
```

```
        cat('\n')

        cat('Sds: ', '\n')
        sds <- sds(r)
        print(sds)
        cat('\n')

        cat('Sfd: ', '\n')
        sfd <- sfd(as.matrix(r))
        print(sfd)
        cat('\n')

        cat('Srw function - no plotting: ', '\n')
        srwvals <- srw(r, plot = FALSE) # this takes a while
        print(srwvals)
        cat('\n')

        cat('Srw: ', '\n')
        srw <- srwvals[[1]]
        print(srw)
        cat('\n')

        cat('Srwi: ', '\n')
        srwi <- srwvals[[2]]
        print(srwi)
        cat('\n')

        cat('Shw: ', '\n')
        shw <- srwvals[[3]]
        print(shw)
        cat('\n')

        cat('Std function - no plotting: ', '\n')
        stdvals <- std(r, plot = FALSE)
        print(stdvals)
        cat('\n')

        cat('Std: ', '\n')
        std <- stdvals[[1]]
        print(std)
        cat('\n')

        cat('Stdi: ', '\n')
        stdi <- stdvals[[2]]
        print(stdi)
        cat('\n')

        cat('Svi: ', '\n')
        svi <- svi(r)
        print(svi)
        cat('\n')

        cat('Str function: ', '\n')
        strvals <- str(r, threshold = c(0.2, 1 / exp(1)))
        print(strvals)
        cat('\n')

        cat('Str (20%): ', '\n')
        str20 <- strvals[[1]]
        print(str20)
        cat('\n')

        cat('Str (37%): ', '\n')
        str37 <- strvals[[2]]
        print(str37)
        cat('\n')

        cat('Ssc: ', '\n')
        ssc <- ssc(r)
        print(ssc)
        cat('\n')

        cat('Sv: ', '\n')
        sv <- sv(r)
        print(sv)
        cat('\n')

        cat('Sp: ', '\n')
        sp <- sph(r)
        print(sp)
        cat('\n')

        cat('Sk: ', '\n')
        sk <- sk(r)
        print(sk)
        cat('\n')

        cat('Smean: ', '\n')
        smean <- smean(r)
        print(smean)
        cat('\n')

        cat('Spk: ', '\n')
        spk <- spk(r)
        print(spk)
        cat('\n')

        cat('Svk: ', '\n')
        svk <- svk(r)
        print(svk)
        cat('\n')

        cat('Scl function - no plotting: ', '\n')
        sclvals <- scl(r, threshold = c(0.2, 1 / exp(1)), plot = FALSE)
        print(sclvals)
        cat('\n')

        cat('Scl (20%): ', '\n')
```

```
    scl20 <- sclvals[[1]] # never gets to 0.2
    print(scl20)
    cat('\n')

    cat('Scl (37%): ', '\n')
    scl37 <- sclvals[[2]]
    print(scl37)
    cat('\n')

    cat('Sdc 0-5%: ', '\n')
    sdc0_5 <- sdc(r, 0, 0.05)
    print(sdc0_5)
    cat('\n')

    cat('Sdc 50-55%: ', '\n')
    sdc50_55 <- sdc(r, 0.50, 0.55)
    print(sdc50_55)
    cat('\n')

    cat('Sdc 80-85%: ', '\n')
    sdc80_85 <- sdc(r, 0.80, 0.85)
    print(sdc80_85)
    cat('\n')

    return(cat('Finished with raster.', '\n'))
  }

  # select a point
  samp_pt <- samp_pts[10,]

  # calculate smaller extent surrounding point (0.25 degrees on from center to all sides)
  bound_box <- extent(samp_pt$x - 0.25, samp_pt$x + 0.25, samp_pt$y - 0.25, samp_pt$y + 0.25)
  bound_box <- as(bound_box, 'SpatialPolygons')

  # plot the box
  plot(evi, xlab = 'Longitude', ylab = 'Latitude', main = 'Sample Location for Test')
  plot(bound_box, add = TRUE)
```



```
# crop the raster
samp_evi <- crop(evi, bound_box)

# look at the cropped area
plot(samp_evi, xlab = 'Longitude', ylab = 'Latitude', main = 'Sample Location for Test')
```



```
# run all functions on this test area
run_all(samp_evi)
#> Sa:
#> [1] 0.04858378
#>
#> Sq:
#> [1] 0.06203606
#>
#> S10z:
#> [1] 0.5917506
```

```
#>
#> Sdq:
#> [1] 30.8906
#>
#> Sdq6:
#> [1] 29.12409
#>
#> Sdr:
#> [1] 0.5893117
#>
#> Sbi:
#> [1] 0.4111862
#>
#> Sci:
#> [1] 0.09331996
#>
#> Ssk (Adjacent):
#> [1] 0.1824763
#>
#> Sku (excess):
#> [1] 0.8381867
#>
#> Sds:
#> [1] 0.06347126
#>
#> Sfd:
#> [1] 2.000371
#>
#> Srw function - no plotting:
#> [[1]]
#> [1] 0.04532409
#>
#> [[2]]
#> [1] 0.678901
#>
#> [[3]]
#> [1] 0.1233822
#>
#>
#> Srw:
#> [1] 0.04532409
#>
#> Srwi:
#> [1] 0.678901
#>
#> Shw:
#> [1] 0.1233822
#>
#> Std function - no plotting:
#> [[1]]
#> [1] 53
#>
#> [[2]]
#> [1] 0.711894
```

```
#>
#>
#> Std:
#> [1] 53
#>
#> Stdi:
#> [1] 0.711894
#>
#> Svi:
#> [1] 0.006914804
#>
#> Str function:
#> [[1]]
#> [1] 0.6954229
#>
#> [[2]]
#> [1] 0.260353
#>
#>
#> Str (20%):
#> [1] 0.6954229
#>
#> Str (37%):
#> [1] 0.260353
#>
#> Ssc:
#> [1] -0.07907536
#>
#> Sv:
#> [1] 0.2371763
#>
#> Sp:
#> [1] 0.3996962
#>
#> Sk:
#> [1] 0.1540902
#>
#> Smean:
#> [1] 0.07336292
#>
#> Spk:
#> [1] 0.2701434
#>
#> Svk:
#> [1] 0.2126534
#>
#> Scl function - no plotting:
#> [[1]]
#> [1] 0.1306906
#>
#> [[2]]
#> [1] 0.01498128
#>
#> [[3]]
#> [1] 0.1879297
#>
#> [[4]]
#> [1] 0.05754216
#>
#>
#> Scl (20%):
#> [1] 0.1306906
#>
#> Scl (37%):
#> [1] 0.01498128
#>
#> Sdc 0-5%:
#> [1] 0.2488253
#>
#> Sdc 50-55%:
#> [1] 0.007899928
#>
#> Sdc 80-85%:
#> [1] 0.01181408
#>
#> Finished with raster.
```

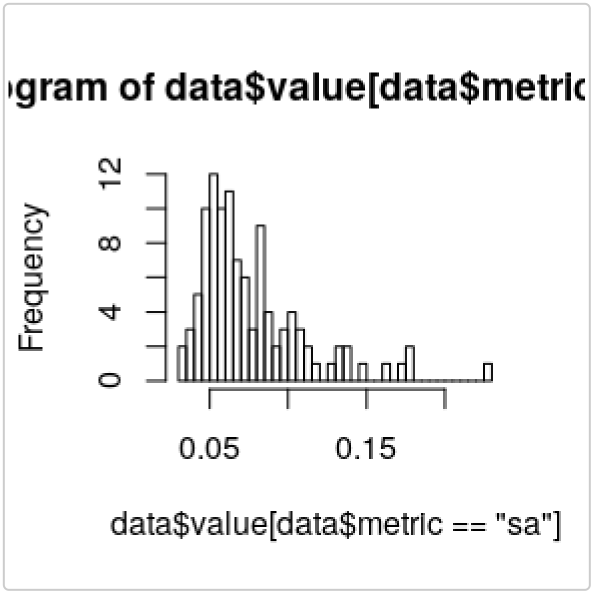
This can take a while, primarily due to the Srw (radial wavelength) and Scl (correlation lengths) functions. However, we want to know how all functions are useful for distinguishing ecosystems, so we'll leave them in for now.

Now we want to run the functions over all points and output the results to a dataframe. This will require re-writing the *run_all* function slightly and looping over all sampled points.

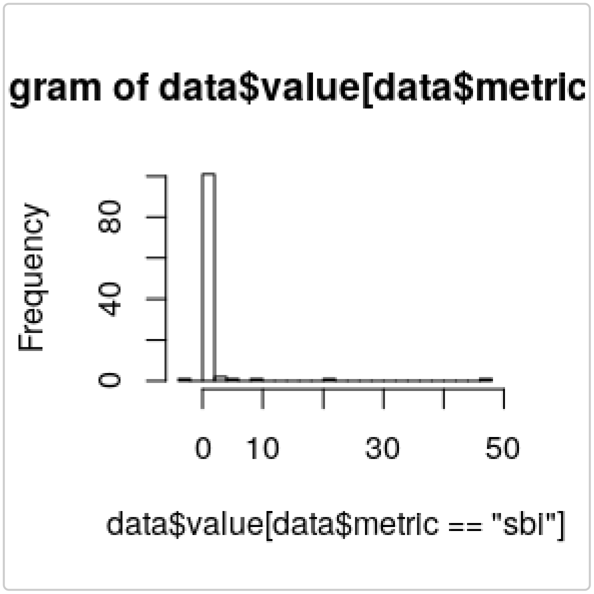
```
# create modified run_all function that outputs results to dataframe
# run_all <- function(r){
#
#   # run all functions
#   sa <- sa(r)
#   sq <- sq(r)
#   s10z <-s10z(r)
#   sdq <- sdq(r)
#   sdq6 <- sdq6(r)
#   sdr <- sdr(r)
#   sbi <- sbi(r)
#   sci <- sci(r)
#   ssk_adj <- ssk(r, adj = TRUE)
#   sku_excess <- sku(r, excess = TRUE)
#   sds <- sds(r)
#   sfd <- sfd(as.matrix(r))
#   srwvals <- srw(r, plot = FALSE)
#   srw <- srwvals[[1]]
#   srwi <- srwvals[[2]]
#   shw <- srwvals[[3]]
#   stdvals <- std(r, plot = FALSE)
#   std <- stdvals[[1]]
#   stdi <- stdvals[[2]]
#   svi <- svi(r)
#   strvals <- str(r, threshold = c(0.2, 1 / exp(1)))
#   str20 <- strvals[[1]]
#   str37 <- strvals[[2]]
#   ssc <- ssc(r)
#   sv <- sv(r)
#   sp <- sph(r)
#   sk <- sk(r)
#   smean <- smean(r)
#   spk <- spk(r)
#   svk <- svk(r)
#   sclvals <- scl(r, threshold = c(0.2, 1 / exp(1)), plot = FALSE)
#   scl20 <- sclvals[[1]]
#   scl37 <- sclvals[[2]]
#   sdc0_5 <- sdc(r, 0, 0.05)
#   sdc50_55 <- sdc(r, 0.50, 0.55)
#   sdc80_85 <- sdc(r, 0.80, 0.85)
#
#   result_df <- data.frame(x = mean(coordinates(r)[, 1]), y = mean(coordinates(r)[, 2]),
#                           metric = c('sa', 'sq', 's10z', 'sdq', 'sdq6', 'sdr', 'sbi',
'sci',
#                                      'ssk', 'sku', 'sds', 'sfd', 'srw', 'srwi', 'shw',
#                                      'std', 'stdi', 'svi', 'str20', 'str37', 'ssc', 'sv',
#                                      'sp', 'sk', 'smean', 'spk', 'svk', 'scl20', 'scl37',
#                                      'sdc0_5', 'sdc50_55', 'sdc80_85'),
#                           value = c(sa, sq, s10z, sdq, sdq6, sdr, sbi, sci, ssk_adj,
#                                     sku_excess, sds, sfd, srw, srwi, shw, std, stdi, svi,
#                                     str20, str37, ssc, sv, sp, sk, smean, spk, svk, scl20,
#                                     scl37, sdc0_5, sdc50_55, sdc80_85))
#   return(result_df)
# }
#
# # loop over all points, cropping raster and running all functions for each
# for (i in seq(1, nrow(samp_pts))){
#   # write out what we're doing
#   cat('Calculating metrics for point: ', i, '\n')
#
#   # select point
#   samp_pt <- samp_pts[i,]
#
#   # calculate smaller extent surrounding point (0.25 degrees on from center to all sides)
#   bound_box <- extent(samp_pt$x - 0.25, samp_pt$x + 0.25, samp_pt$y - 0.25, samp_pt$y +
0.25)
#   bound_box <- as(bound_box, 'SpatialPolygons')
#
#   # crop the raster
#   samp_evi <- crop(evi, bound_box)
#
#   # if irregular, remove rows with NA vals
#   # get rows/cols with nas
#   na_col <- which(colSums(is.na(samp_evi)) > 0)
#   na_row <- which(rowSums(is.na(samp_evi)) > 0)
#   if (length(na_col) + length(na_row) != 0) {
#       # list only good rows/cols
#     good_cols <- seq(1, ncol(samp_evi))[-na_col]
#     good_rows <- seq(1, nrow(samp_evi))[-na_row]
#     # crop raster
#     new_ext <- extent(samp_evi, min(good_rows), max(good_rows), min(good_cols),
max(good_cols))
#     samp_evi <- crop(samp_evi, new_ext)
#   }
#
#   # calculate metrics
#   out <- run_all(samp_evi)
#   out$point <- i
#
#   # add dataframes (or create if first point)
#   if (i == 1){
#     result <- out
#   } else {
#     result <- rbind(result, out)
#   }
#
#   # remove data
#   rm(samp_pt, bound_box, samp_evi, out)
# }
#
# # write out result
# write.csv(result, '~/Documents/surface_metrics/oregon_metric_results.csv', row.names = F)

# read in the future, because the above takes a while
result <- read.csv('~/Documents/surface_metrics/oregon_metric_results.csv')
```

We now have metrics for all of the sampling locations. We want to see if the metrics result in any natural grouping of the sites. We'll start investigating this by exploring the raw results. Let's look at distributions of a few metrics and map those metrics across the state.

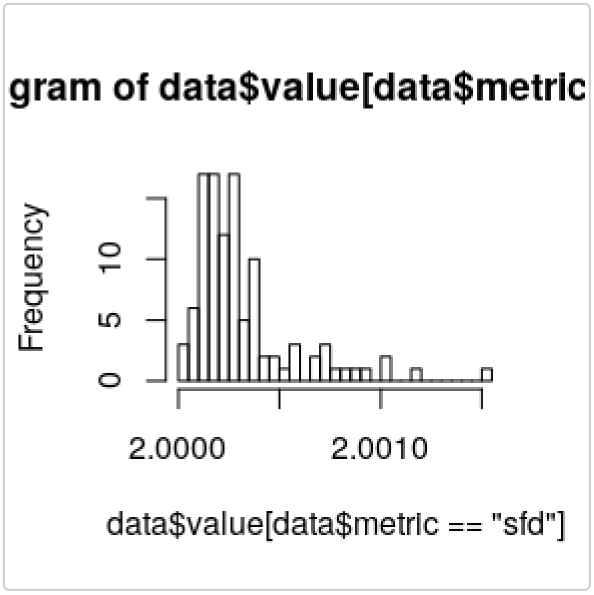
```
data <- result

# distributions of a few variables
hist(data$value[data$metric == 'sa'], breaks = 30)
```



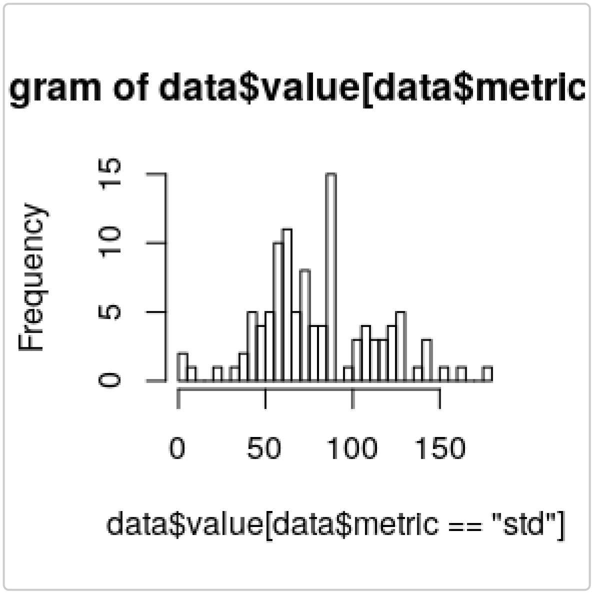
```
hist(data$value[data$metric == 'sbi'], breaks = 30)
```



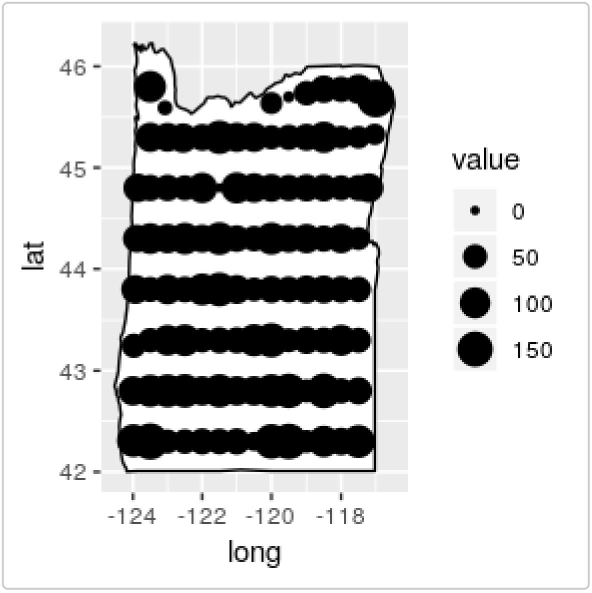
```
hist(data$value[data$metric == 'sfd'], breaks = 30)
```



```r
hist(data$value[data$metric == 'std'], breaks = 30)
```



```r
# map the texture direction index
or_ggpoly <- map_data('state', 'oregon')
ggplot() +
  geom_polygon(data = or_ggpoly, aes(x = long, y = lat, group = region), fill = 'white',
colour = 'black') +
  geom_point(data = data[data$metric == 'std',], aes(x = x, y = y, size = value))
```



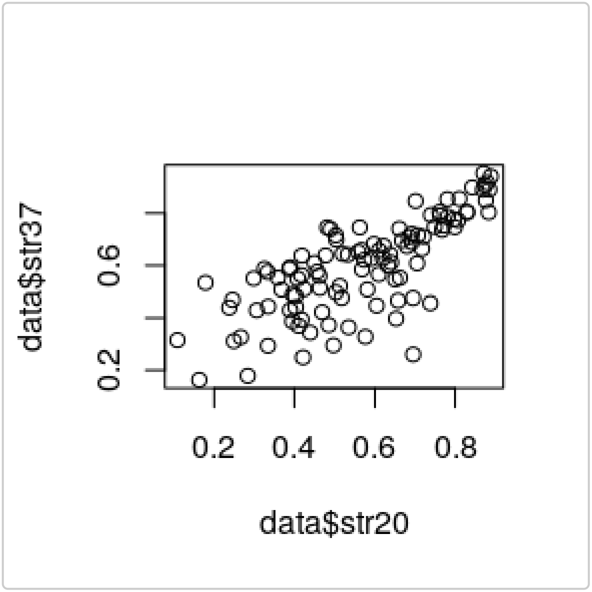
There aren't really any clear patterns in the map of texture direction, but combining metrics may improve our ability to distinguish ecosystems.

We now want to investigate the observed groupings using a statistical approach that simplifies the many metrics into fewer groups, principal components analysis. The composition of the various components provides information on which metrics contain similar information. By observing how the first two or three principal components relate to one another, we can potentially split sites into ecosystems with various characteristics.

```
library(tidyverse)
#> Registered S3 method overwritten by 'cli':
#>   method     from
#>   print.boxx spatstat
#> ── Attaching packages ─────────────────────────────────────── tidyverse
1.2.1 ──
#> ✔ tibble  2.1.3     ✔ purrr   0.3.2
#> ✔ tidyr   0.8.3     ✔ dplyr   0.8.3
#> ✔ readr   1.3.1     ✔ stringr 1.4.0
#> ✔ tibble  2.1.3     ✔ forcats 0.4.0
#> ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
#> ✖ tidyr::extract() masks raster::extract()
#> ✖ dplyr::filter()  masks stats::filter()
#> ✖ dplyr::lag()     masks stats::lag()
#> ✖ purrr::map()     masks maps::map()
#> ✖ dplyr::select()  masks raster::select()

# transform data into correct format (columns for each metric)
data <- spread(data = data, key = metric, value = value)

# order neatly
data <- data[order(data$point),]

# we should now have 108 neatly ordered rows
nrow(data)
#> [1] 108
```

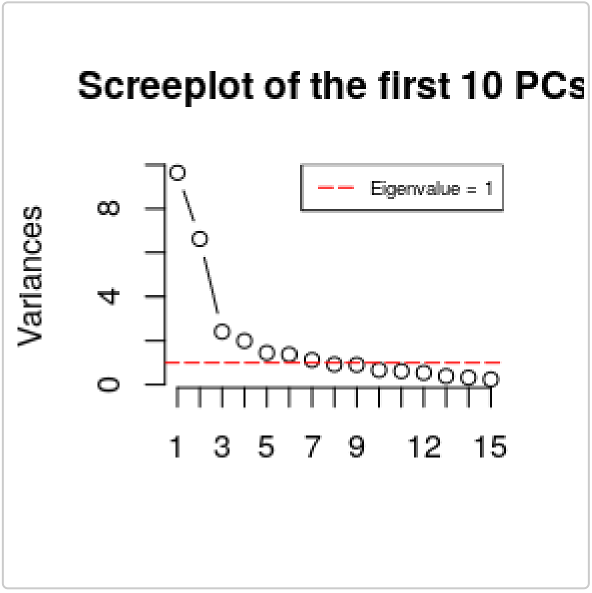
```
head(data)
#>           x        y point      s10z         sa        sbi        sci
#> 2 -123.9866 42.30840     1 0.7240777 0.06244592 -3.9289936 0.11336066
#> 1 -124.0001 42.80023     2 0.4997539 0.05530583  1.2019567 0.10766284
#> 3 -123.9742 43.24602     3 0.9544846 0.04885489  0.6815841 0.08491285
#> 4 -123.9271 43.79961     4 1.0359335 0.06880177  0.8536535 0.10599427
#> 5 -123.8867 44.29930     5 0.5852673 0.06073367  0.9071385 0.11310450
#> 6 -123.8608 44.80011     6 0.7101857 0.05918598  0.7545252 0.09953877
#>        scl20       scl37    sdc0_5    sdc50_55   sdc80_85      sdq
#> 2 0.21466082 0.154474481 0.3580400 0.008785224 0.02113566 26.98778
#> 1 0.14768085 0.095683493 0.2505639 0.009100442 0.01335223 23.75134
#> 3 0.01240277 0.007940060 0.1768070 0.007303348 0.01323875 28.94301
#> 4 0.02773344 0.012402771 0.2182686 0.010272076 0.01750276 35.95720
#> 5 0.04299380 0.008551707 0.2357646 0.009833722 0.01663469 34.67289
#> 6 0.03732673 0.014597623 0.1853310 0.009212850 0.01788011 33.92831
#>       sdq6       sdr         sds      sfd        shw        sk        sku
#> 2 36.93881 0.3102054 0.007153931 2.000243 0.11024778 0.1733075  0.3808235
#> 1 25.27099 0.4102937 0.030064001 2.000285 0.11331022 0.1809745 -0.1740248
#> 3 26.34256 0.2149452 0.057962225 2.000142 0.08733621 0.1434855 18.2045372
#> 4 33.90695 0.2967178 0.053289910 2.000189 0.10546567 0.1953242  9.3869061
#> 5 32.01336 0.7783490 0.044747227 2.000708 0.10850900 0.1970176  0.2395116
#> 6 30.29066 0.4924310 0.058972238 2.000339 0.10433558 0.1699093  1.3093668
#>        smean        sp       spk         sq        srw      srwi
#> 2 0.03983522 0.3377579 0.3853443 0.07968808 0.01469970 0.7262572
#> 1 0.03878259 0.3076593 0.2734495 0.06862617 0.04784209 0.7723231
#> 3 0.04862148 0.2778801 0.1930638 0.06888981 0.03175862 0.7013696
#> 4 0.05840874 0.3316204 0.2265934 0.09676318 0.01917558 0.7032309
#> 5 0.04798783 0.3198276 0.2556198 0.07625678 0.03756081 0.6832798
#> 6 0.05055483 0.2869315 0.2006302 0.07666012 0.05216779 0.7111010
#>           ssc         ssk std      stdi     str20     str37        sv
#> 2 -0.11344251 -0.14091837 123 0.7832487 0.8839647 0.8032296 0.4290753
#> 1 -0.07733982  0.05839043  90 0.7037778 0.5910999 0.6264028 0.2801206
#> 3 -0.04470545 -2.30472301  50 0.7395412 0.5621941 0.7453560 0.7099903
#> 4 -0.05286648 -1.96801963  81 0.6819823 0.3335765 0.5742210 0.7117423
#> 5 -0.10549099  0.08344649  76 0.6210672 0.4030132 0.4411699 0.3013757
#> 6 -0.08004436 -0.55367439  83 0.6058027 0.6801005 0.6731145 0.4785747
#>           svi       svk
#> 2 0.010275656 0.2082056
#> 1 0.007170106 0.1333676
#> 3 0.009603844 0.6513351
#> 4 0.014545209 0.6214802
#> 5 0.008602819 0.1685923
#> 6 0.011145097 0.3949724

# check for missing values
sapply(data[, c(4:30, 32:35)], function(x) sum(is.na(x)))
#>     s10z       sa      sbi      sci    scl20    scl37   sdc0_5 sdc50_55
#>        0        0        0        0        0        0        0        0
#> sdc80_85      sdq     sdq6      sdr      sds      sfd      shw       sk
#>        0        0        0        0        0        0        0        0
#>      sku    smean       sp      spk       sq      srw     srwi      ssc
#>        0        0        0        0        0        0        0        0
#>      ssk      std     stdi    str37       sv      svi      svk
#>        0        0        0        0        0        0        0
```

```
# str20 has one missing value, which makes sense because it means that autocorrelation
# never decreases to 20% in that area
# for now, we will remove str20 because it correlates extremely highly with str37
plot(data$str20, data$str37)
```

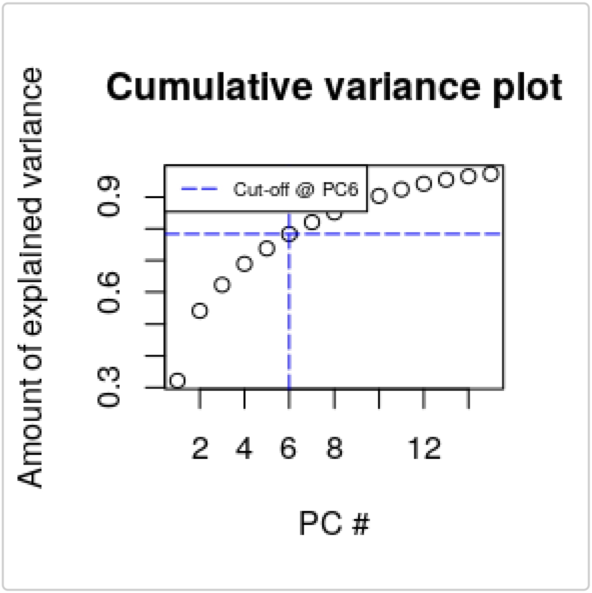


```
# same with scl20, which is Inf for the same location (found by looking at the df)

# calculate principal components
data_prc <- prcomp(data[, c(4:7, 9:30, 32:35)], center = TRUE, scale = TRUE)
summary(data_prc)
#> Importance of components:
#>                           PC1    PC2     PC3     PC4     PC5     PC6
#> Standard deviation     3.1055 2.5734 1.55313 1.41259 1.20406 1.17266
#> Proportion of Variance 0.3215 0.2208 0.08041 0.06651 0.04833 0.04584
#> Cumulative Proportion  0.3215 0.5422 0.62264 0.68915 0.73748 0.78331
#>                            PC7     PC8    PC9    PC10    PC11   PC12
#> Standard deviation     1.05832 0.96004 0.9502 0.81548 0.78344 0.7348
#> Proportion of Variance 0.03733 0.03072 0.0301 0.02217 0.02046 0.0180
#> Cumulative Proportion  0.82065 0.85137 0.8815 0.90363 0.92409 0.9421
#>                           PC13    PC14    PC15    PC16    PC17    PC18
#> Standard deviation     0.62150 0.56497 0.48874 0.45389 0.39231 0.32717
#> Proportion of Variance 0.01288 0.01064 0.00796 0.00687 0.00513 0.00357
#> Cumulative Proportion  0.95497 0.96561 0.97357 0.98044 0.98557 0.98914
#>                           PC19    PC20    PC21    PC22    PC23    PC24
#> Standard deviation     0.30766 0.24380 0.22195 0.19653 0.17126 0.14788
#> Proportion of Variance 0.00316 0.00198 0.00164 0.00129 0.00098 0.00073
#> Cumulative Proportion  0.99229 0.99427 0.99592 0.99720 0.99818 0.99891
#>                           PC25    PC26    PC27    PC28    PC29      PC30
#> Standard deviation     0.11729 0.10851 0.06524 0.05235 0.01397 0.0002605
#> Proportion of Variance 0.00046 0.00039 0.00014 0.00009 0.00001 0.0000000
#> Cumulative Proportion  0.99937 0.99976 0.99990 0.99999 1.00000 1.0000000

# take a look at the components
screeplot(data_prc, type = "l", npcs = 15, main = "Screeplot of the first 10 PCs")
```

```
abline(h = 1, col = "red", lty = 5)
legend("topright", legend = c("Eigenvalue = 1"),
       col = c("red"), lty = 5, cex = 0.6)
```

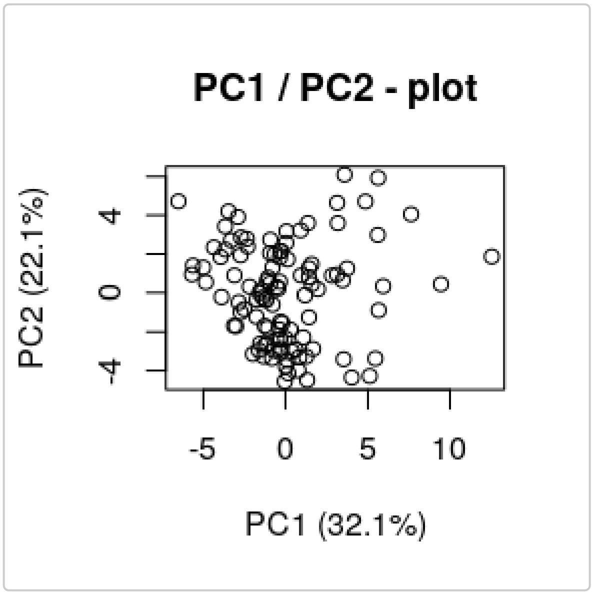


```
cumpro <- cumsum(data_prc$sdev ^ 2 / sum(data_prc$sdev ^ 2))
plot(cumpro[0:15], xlab = "PC #", ylab = "Amount of explained variance", main = "Cumulative
variance plot")
abline(v = 6, col = "blue", lty = 5)
abline(h = 0.7833145, col = "blue", lty = 5)
legend("topleft", legend = c("Cut-off @ PC6"),
       col = c("blue"), lty = 5, cex = 0.6)
```

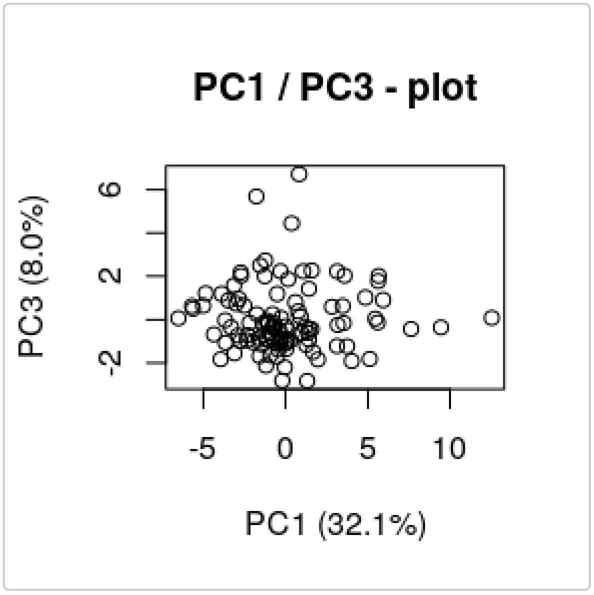


```
# the first 2 components explain 54% of the variance, while the first three explain 62%

# let's plot the first 2
```

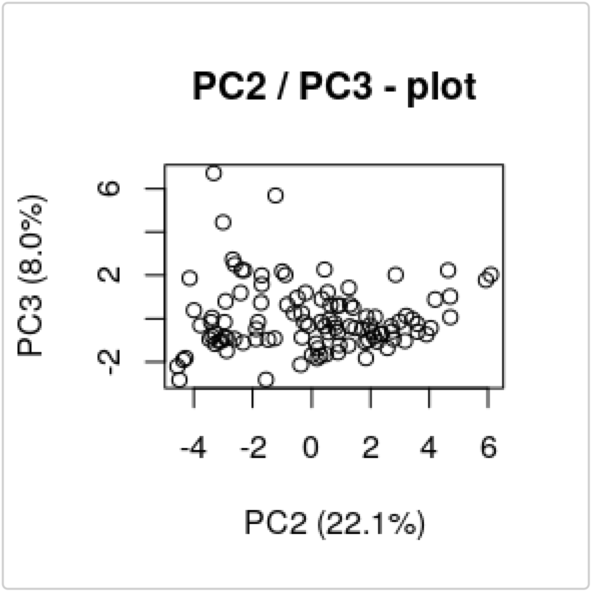
```r
plot(data_prc$x[, 1], data_prc$x[, 2], xlab = "PC1 (32.1%)", ylab = "PC2 (22.1%)", main =
"PC1 / PC2 - plot")
```



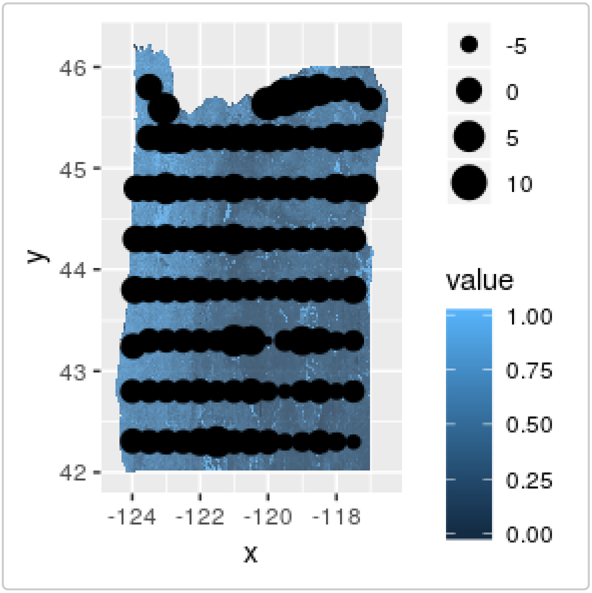
```r
plot(data_prc$x[, 1], data_prc$x[, 3], xlab = "PC1 (32.1%)", ylab = "PC3 (8.0%)", main = "PC1
/ PC3 - plot")
```



```r
plot(data_prc$x[, 2], data_prc$x[, 3], xlab = "PC2 (22.1%)", ylab = "PC3 (8.0%)", main = "PC2
/ PC3 - plot")
```

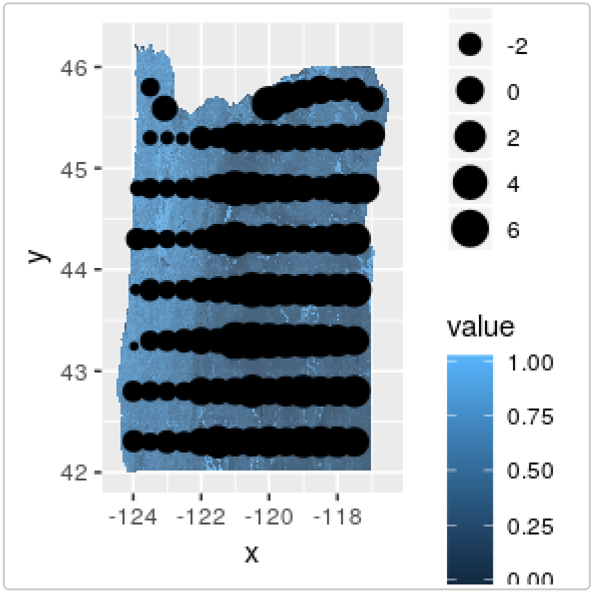


```
# how about mapping - are there any patterns?
data$prc1 <- data_prc$x[, 1]
data$prc2 <- data_prc$x[, 2]
data$prc3 <- data_prc$x[, 3]

# for plotting, convert EVI values to dataframe
evi_pixdf <- as(oregonEVI, "SpatialPixelsDataFrame")
evi_df <- as.data.frame(evi_pixdf)
colnames(evi_df) <- c("value", "x", "y")

# principal component 1
ggplot() +
  geom_tile(data = evi_df, aes(x = x, y = y, fill = value), alpha = 0.8) +
  geom_point(data = data, aes(x = x, y = y, size = prc1))
```

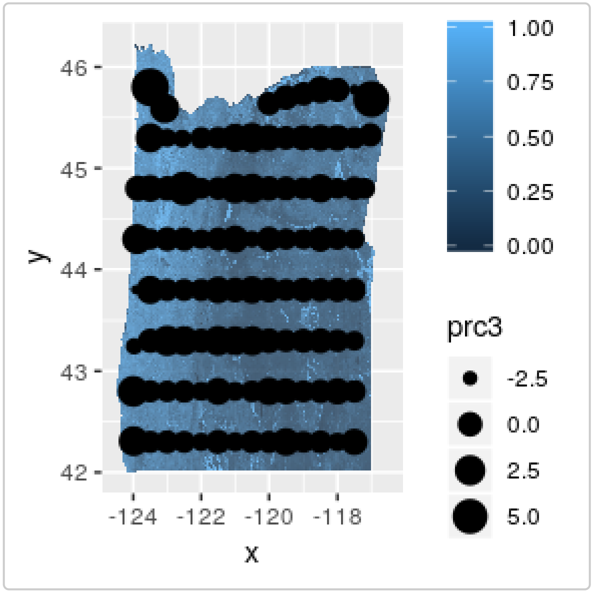


```
# principal component 2
```

```
ggplot() +
  geom_tile(data = evi_df, aes(x = x, y = y, fill = value), alpha = 0.8) +
  geom_point(data = data, aes(x = x, y = y, size = prc2))
```



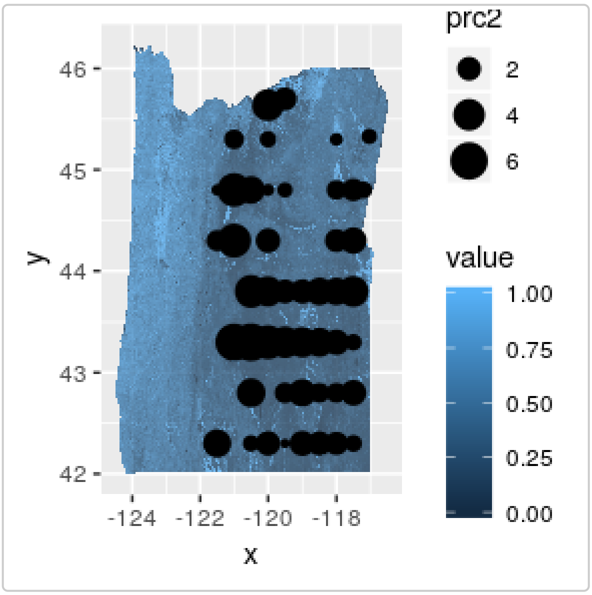
```
# principal component 3
ggplot() +
  geom_tile(data = evi_df, aes(x = x, y = y, fill = value), alpha = 0.8) +
  geom_point(data = data, aes(x = x, y = y, size = prc3))
```



While the individual principal components don't show any clustering, we can see what they represent when we map the results. The first principal component appears to be linked with visible heterogeneity in EVI values, while the second clearly represents the eastern side of the state versus the western side of the state. It isn't clear what the third component represents; however, the primary variables included in that component are the ten-point height of EVI and the summit curvature around the highest peaks. Both variables are negatively represented by the component, so high values of this component should roughly translate to relativley 'flat' areas of EVI (similar greenness values) or areas with high heterogeneity, but low contrast.

We investigate these patterns further by applying thresholds to the map to find low or high areas.

```
# shat do various cutoffs for principal component 2 show?
ggplot() +
  geom_tile(data = evi_df, aes(x = x, y = y, fill = value), alpha = 0.8) +
  geom_point(data = data[data$prc2 >= 0.5,], aes(x = x, y = y, size = prc2))
```



The second principal component is clearly associated with lower and flatter EVI values, as all positive values of the component are in the unforested or relatively unforested areas of Oregon.

discuss results. explain mapping of clusters.

final discussion of results and implications