KID
Shade Extraction

Fabian Blasch

01.04.2022

Packages

```
# git install opencv
# devtools::install_github("ropenscilabs/opencv", force = T)

# Packages
get.package <- function(package){

  lapply(package, \(x){
    # check if packages are installed and if not install them
    if(!require(x, character.only = T)){
       install.packages(x)
    }
    # call package
    library(x, character.only = T)
  })

}

# exec
get.package(c("png", "jpeg", "tabulizer", "pdftools", "raster", "rgdal", "sp",
              "cluster"))




# since I will use Map() / lapply() alot for plotting I will wrap them in invisible()
invis.Map <- function(f, ...) invisible(Map(f, ...))
invis.lapply <- function(x, f, ...) invisible(lapply(x, f, ...))
```

Import KIDs

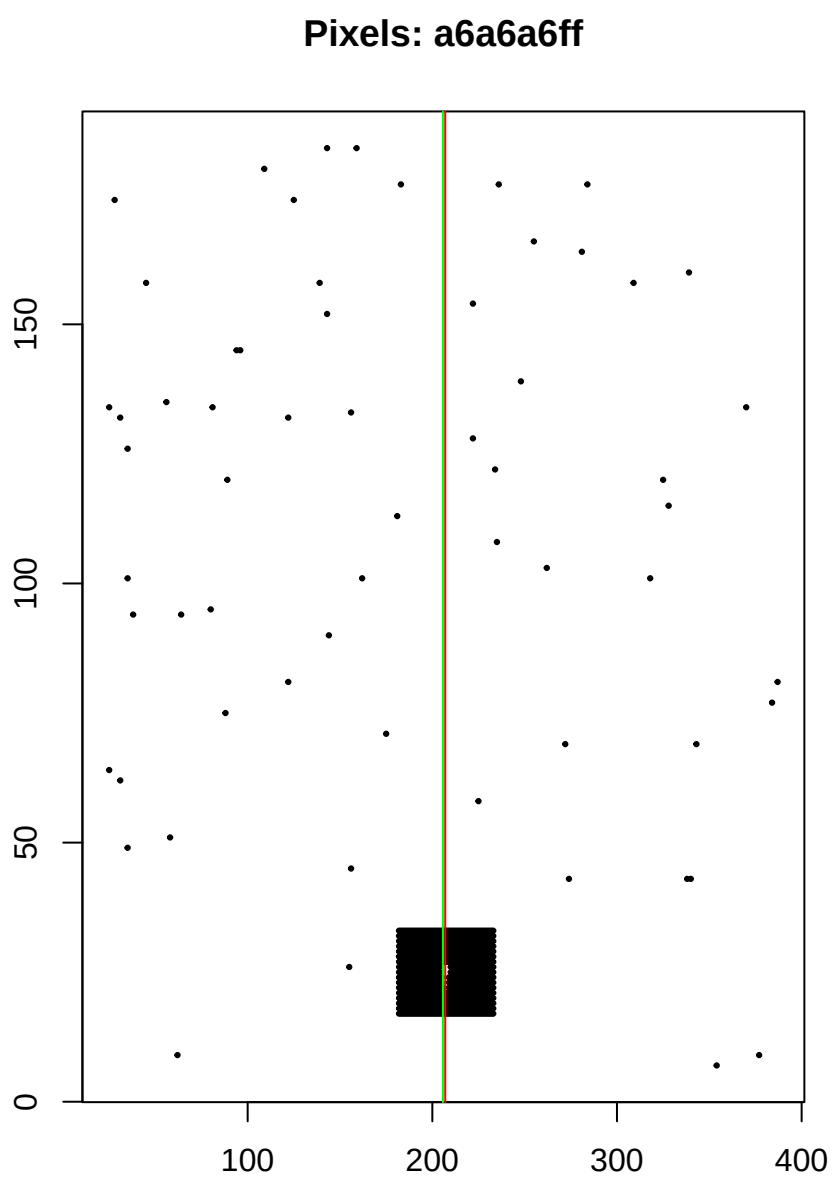
```
# set
setwd("./../../KIDs/Erste")

# all PDF files in the current directory
file_names <- list.files(pattern = ".pdf")

# extract text split by linebreak
lapply(file_names, function(x){
```

```
  # split by line break and extract text
  strsplit(pdftools::pdf_text(x), "\n")

}) -> pdf.text.list

# identofy line that contains "Risiko- und Ertragsprofil"
lapply(pdf.text.list, function(x){

  # obtain relative line on page
  sapply(x, function(y){
    # id
    tmp1 <- grep("Risiko- und Ertragsprofil", y) / length(y)

    # return
    if(length(tmp1) == 0){
      return(NA)
    } else {
      return(tmp1)
    }
  })

}) -> pos.list

# find page that contains desired text
sapply(pos.list, function(x) which(!is.na(x))) -> page.oi

# import PDF and convert to Png
Map(function(x, y, z){

  # convert first page of pdf to bitmap
  tmp1 <- pdftools::pdf_render_page(x, page = y, dpi = 50)

  # subset array
  tmp2 <- round(dim(tmp1)[3] * z)

  # return
  tmp1[ , , -c(tmp2:1)]

}, file_names, page.oi, sapply(pos.list, "[", 1)) -> bitmap.list

# import PDF and convert to Png

# to JPG
jpeg::writeJPEG(bitmap.list[[1]], "test1.jpeg")

# JPEG
imt <- jpeg::readJPEG("test.jpeg")
```

Extract SSRI

Bitmap

```
# bitmap of first fund
bitmp1 <- bitmap.list[[1]]

# col
# bitmp1[,587, 690]

# coordinates
coo1 <- which(bitmp1[1,,] == "a6" & bitmp1[2,,] == "a6" & bitmp1[3,,] == "a6" &
              bitmp1[4,,] == "ff", arr.ind = T)

# plot
{
plot(coo1[,1], coo1[, 2], main = "Pixels: a6a6a6ff", pch = 19, cex = 0.3, ylab = "",
     xlab = "")
abline(v = median(coo1[, 1]), col = "red")
abline(v = mean(coo1[, 1]), col = "green")
}
```



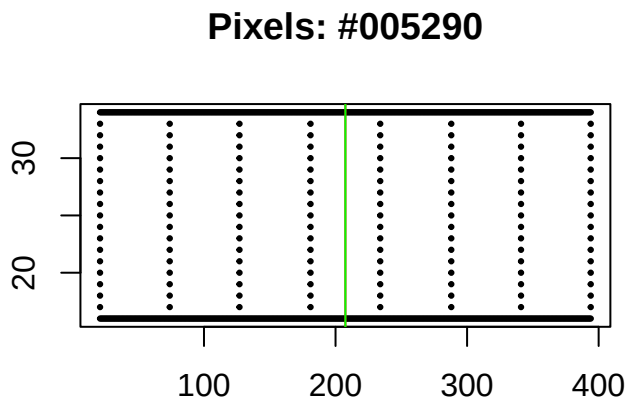

```
# find page margin for now use same color later should be switched to black
# left side margin
lsm <- min(coo1[, 1])
rsm <- max(coo1[, 1])

# scale
int_leng <- (rsm - lsm) / 7

# midpoints
scale <- setNames(cumsum(c(lsm + int_leng / 2, rep(int_leng, 6))), 1:7)
```

```
# only using the median to predict SSRI
which.min(abs(median(coo1[, 1]) - scale))
```

```
## 4
## 4
```

```
# in this case we correctly predict the SSRI, without further classification!
```

```
# check if page is flipped upside down

# coordinates
coob <- which(bitmp1[1,,] == "00" & bitmp1[2,,] == "00" & bitmp1[3,,] == "00",
              arr.ind = T)

# plot
{
plot(coob[,1], coob[, 2], main = "Pixels: #005290", pch = 19, cex = 0.3, xlab = "",
     ylab = "")
abline(v = median(coob[, 1]), col = "red")
abline(v = mean(coob[, 1]), col = "green")
}
```

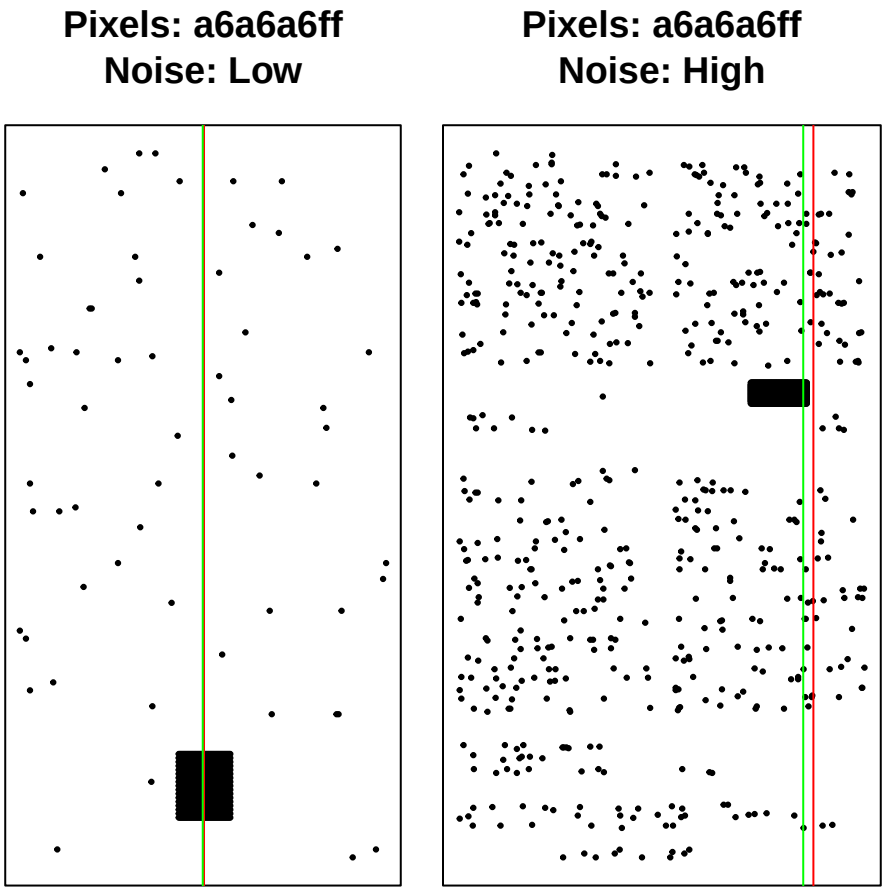



```
# We realize the page is upside down in the bitmap but not mirrored!
```

JPEG

```
# convert by alpha value in this case 255
imt <- imt * 255

# coordinates
    # coo <- which((0 < imt[,, 1]  & imt[,, 1] < 10) &
    #              (75 < imt[,, 2] & imt[,, 2] < 85) &
    #              (130 < imt[,, 3] & imt[,, 3] < 150), arr.ind = T)

coo <- which(imt[,,1] == 166 & imt[,,2] == 166 & imt[,,3] == 166, arr.ind = T)
```

```
# Pixel / Coordinateplot
#save
#pdf(file = "highnoiselownoise.pdf", height = 5)

par(mfrow = c(1, 2), mar = c(1, 1, 4, 1) + 0.1)

{
plot(coo1[,1], coo1[, 2], main = "Pixels: a6a6a6ff\nNoise: Low", pch = 19, cex = 0.3, ylab = "", xlab =
abline(v = median(coo1[, 1]), col = "red")
abline(v = mean(coo1[, 1]), col = "green")
}
par(mar = c(1, -0.1, 4, 1) + 0.1)
{
plot(coo[,2], coo[, 1], main = "Pixels: a6a6a6ff\nNoise: High", pch = 19, cex = 0.3, ylab = "",
     xlab = "", col.lab = "black", col.axis = "white", xaxt = "n", yaxt = "n")
abline(v = median(coo[, 1]), col = "red")
abline(v = mean(coo[, 1]), col = "green")
}
```



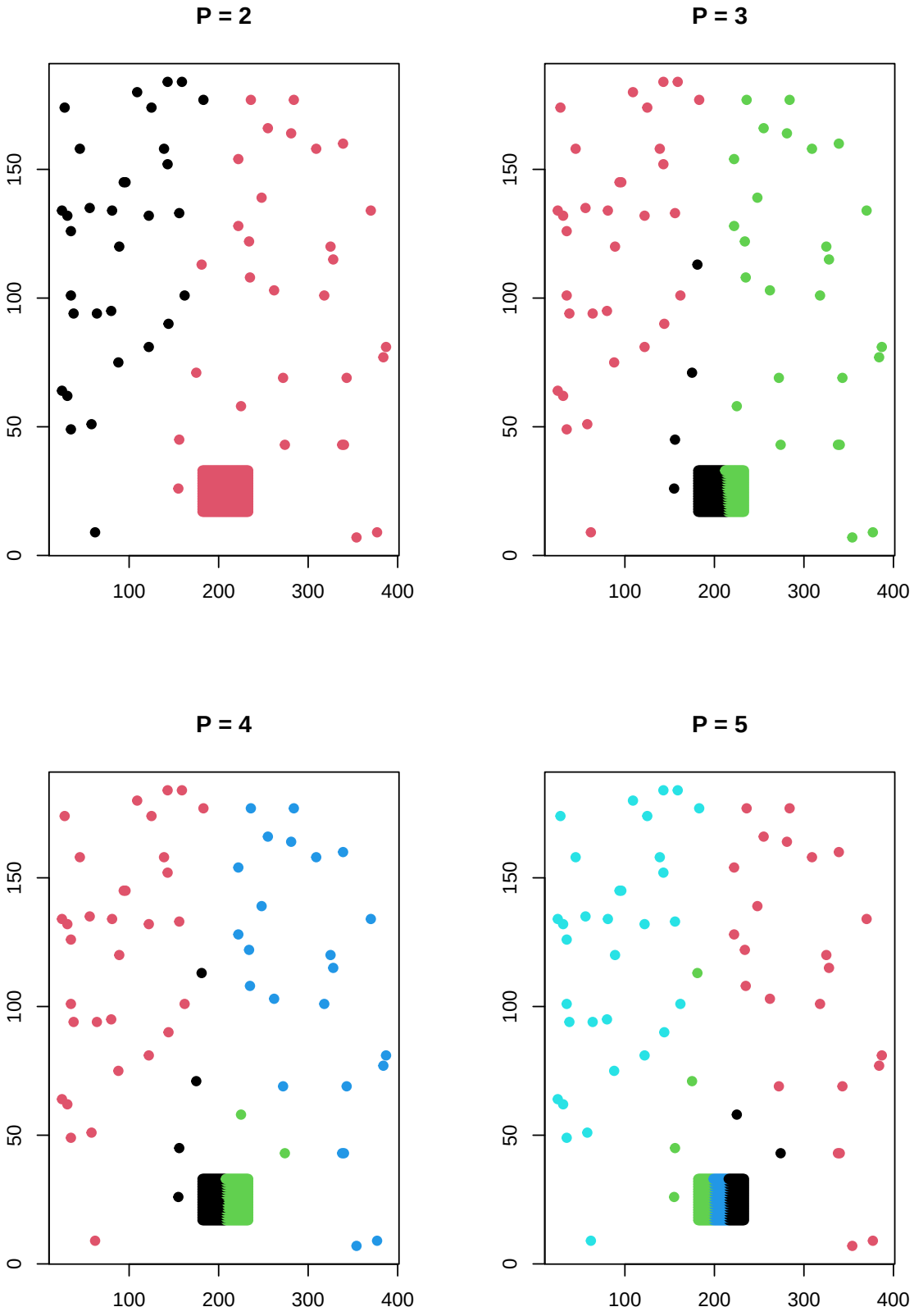

```
#off
#dev.off()
```

Classify utilizing k-means.

```
# Classify with different amount of groups then check for sil coef

# amt of groups
p <- 2:5

# estimate
lapply(p, function(x){

  # merge cluster into df
  dat <- cbind(coo1, kmeans(coo1, x)$cluster) # try specifying centers as closest to median most top le

  # silhouette
  tmp1 <- cluster::silhouette(dat[, ncol(dat)], dist(coo1))

  # return SC and Data
  list(
  "SC" = max(tapply(tmp1[, "sil_width"], tmp1[, "cluster"], mean)),
  "dat" = dat)

}) -> dat.kmeans

# sil coef
sapply(dat.kmeans, "[[", 1)
```

```
## [1] 0.8424069 0.5905349 0.5577094 0.4693019
```

```
# data.frames with the classification of different amt. of groups
# arrange
par(mfrow = c(2, 2))

# plot
invis.Map(function(x, y){

  plot(y[, 1], y[, 2], col = y[, ncol(y)], pch = 19, main = paste("P =", x), xlab = "",
       ylab = "")

}, p, lapply(dat.kmeans, "[[", 2))
```


P = 2
P = 3
P = 4
P = 5
0
50
100
150
100
200
300
400

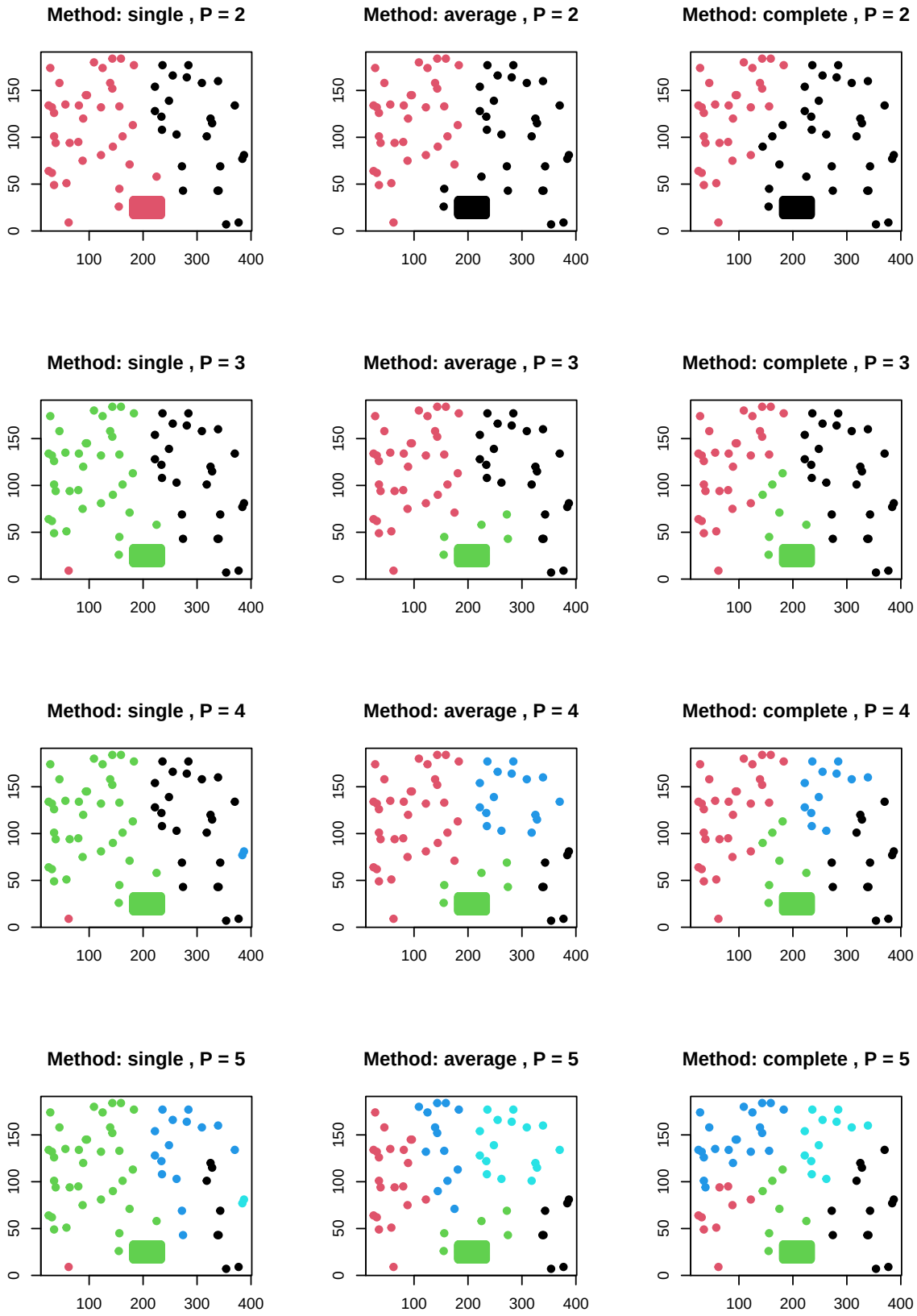
Classify utilizing hierarchical clustering

```
# methods
meth <- c("single", "average", "complete")

# ramp up p
p <- 2:5

# estimate
# over methods
lapply(meth, function(x){

  # over p
  lapply(p, function(y){

    # get grouping
    tmp1 <- agnes(coo1, method = x, diss = F)

    # restrict amnt of groups
    tmp2 <- cutree(tmp1, k = y)

    # bind
    tmp3 <- cbind(coo1, tmp2)

    # calculate coefficients
    tmp4 <- silhouette(tmp3[, ncol(tmp3)], dist(tmp3[, 2:3]))

    # SC
    SC <- max(tapply(tmp4[, "sil_width"], tmp4[, "cluster"], mean))

    # return
    list("Data" = tmp3,
    "SC" = SC,
    "tmp.plot.silhouette" = tmp4)

  }) |> setNames(nm = paste("P =", p))

}) |> setNames(nm = meth) -> Group.list

# SC
lapply(Group.list, \(x){
  sapply(x, "[[", "SC")
})
```

```
## $single
##     P = 2     P = 3     P = 4     P = 5
## 0.8460797 0.3397616 0.9236410 0.9015994
##
## $average
##     P = 2     P = 3     P = 4     P = 5
## 0.8831344 0.9328561 0.8125481 0.8125481
##
## $complete
##     P = 2     P = 3     P = 4     P = 5
```

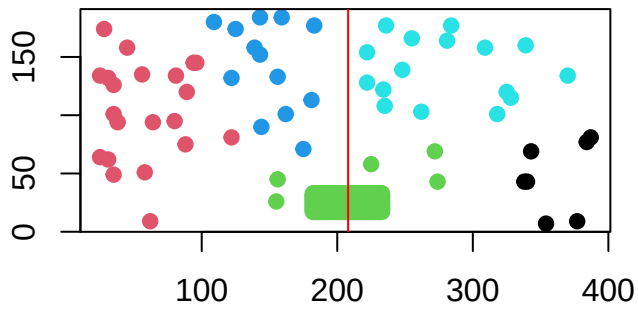
```
## 0.8787131 0.9219419 0.8719915 0.8497295
```

```
# plot
# arrange
par(mfrow = c(4, 3))

invis.lapply(paste("P =", p), \(x){

  # over p
  invis.lapply(meth, \(y){

    # Data
    tmp.plot <- Group.list[[y]][[x]][["Data"]]

    # plot
    plot(tmp.plot[, 1], tmp.plot[, 2], col = tmp.plot[, ncol(tmp.plot)], pch = 19,
    main = paste("Method:", y, ",", x), xlab = "", ylab = "")

  })
})
```


Method: single , P = 2
Method: average , P = 2
Method: complete , P = 2
Method: single , P = 3
Method: average , P = 3
Method: complete , P = 3
Method: single , P = 4
Method: average , P = 4
Method: complete , P = 4
Method: single , P = 5
Method: average , P = 5
Method: complete , P = 5
0
50
100
150
100
200
300
400

Identifying the cluster and estimating SSRI

```
# identify cluster by summed up variance over
dat.p5.ave <- as.data.frame(Group.list[["average"]][["P = 5"]][["Data"]])

# aggregate
which.min(rowSums(aggregate(dat.p5.ave[, 1:2], by = dat.p5.ave[, 3, drop = F],
                            var))) -> rect.grp

med.rect.grp <- median(dat.p5.ave[, 1][which(dat.p5.ave[, 3] == rect.grp)])

# calculate median
{
  plot(dat.p5.ave[, 1], dat.p5.ave[, 2], col = dat.p5.ave[, 3], pch = 19,
       xlab = "", ylab = "")
  abline(v = med.rect.grp,col = "red")
}
```




```
# classify
dif <- abs(med.rect.grp - scale)

# amt pixels off
dif
```

```
##         1         2         3         4         5         6         7
## 157.14286 105.42857  53.71429   2.00000  49.71429 101.42857 153.14286
```

```
# which grp
which.min(dif)
```

```
## 4
## 4
```