Assignment 4

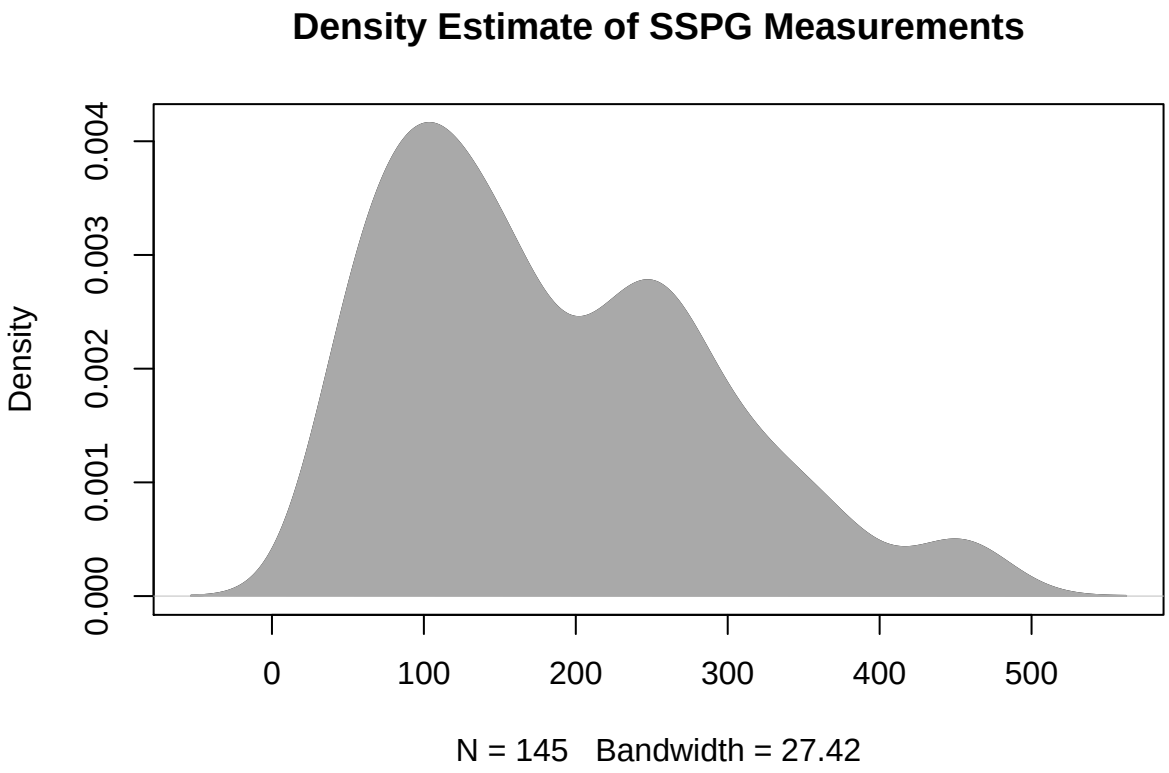
Azoacha Forcheh, 20558994

1. Download the `diabetes` data from the course website. In that file, there is a dataset on various measurements of 145 patients. Once you load this file into your R session (or equivalently, execute its contents there) there will be a data set called `diabetes`.

   The variate `SSPG` stands for steady state plasma glucose which measures the patient's insulin resistance, a pathological condition where the body's cells fail to respond to the hormone insulin.

   (a) (3 marks) Produce a plot of a density estimate of `SSPG` and comment on what you see.

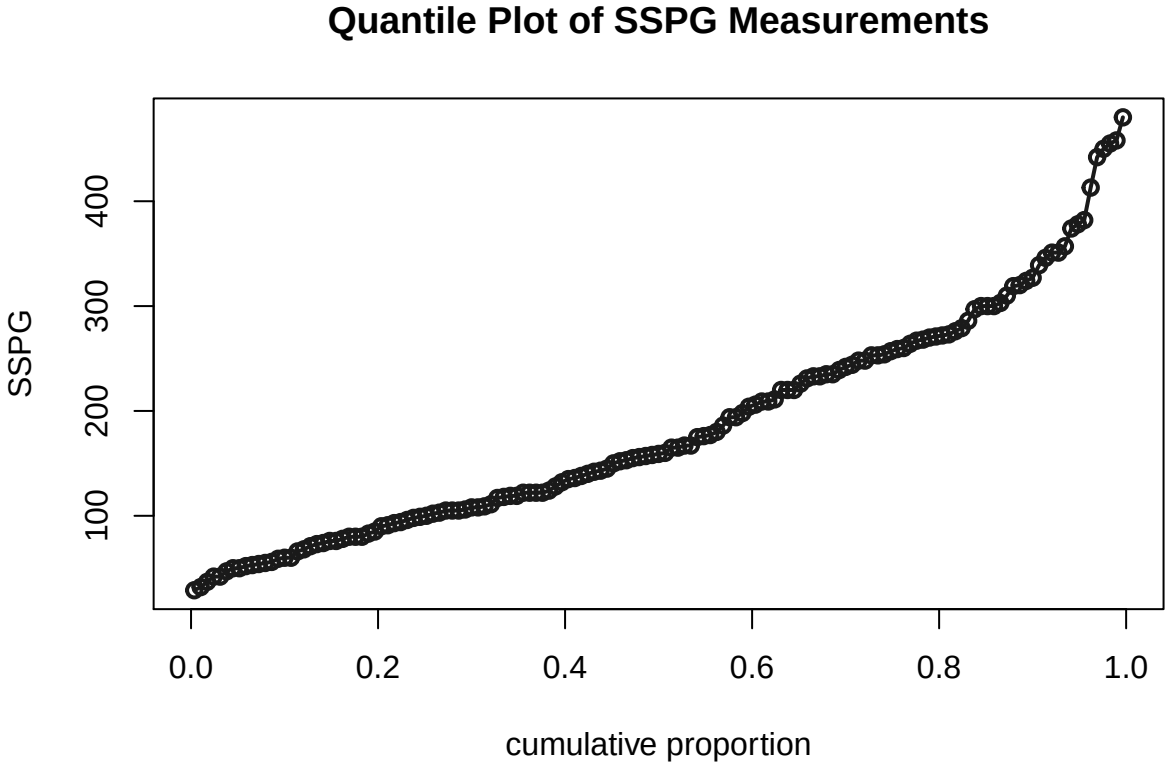
```
sspg_dens = density(diabetes$SSPG, bw="SJ")
plot(sspg_dens, main="Density Estimate of SSPG Measurements")
polygon(sspg_dens, col="dark grey", border="dark grey", xlab="SSPG")
```



   The density is right-skewed and trimodal, with modes at about 100, 250 and 450. The curve rises quickly to the first mode, then gradually decreases. In addition, the majority of patients have an SSG measurement below 250. Lastly, the SSPG seems lacks normality - i.e. it does not seem to have been sampled from the normal - as its density does not resemble the symmetric bell curve of a normal density.

   (b) Construct a quantile plot of `SSPG` and comment on the shape of its distribution.

```
sspg_vals = sort(diabetes$SSPG)
n = length(sspg_vals)
p = ppoints(n) # the proportions

plot(x = p, y = sspg_vals, type="o", lwd=2, col="grey10",
    xlab="cumulative proportion", xlim=c(0,1),
    ylab="SSPG", main="Quantile Plot of SSPG Measurements")
```

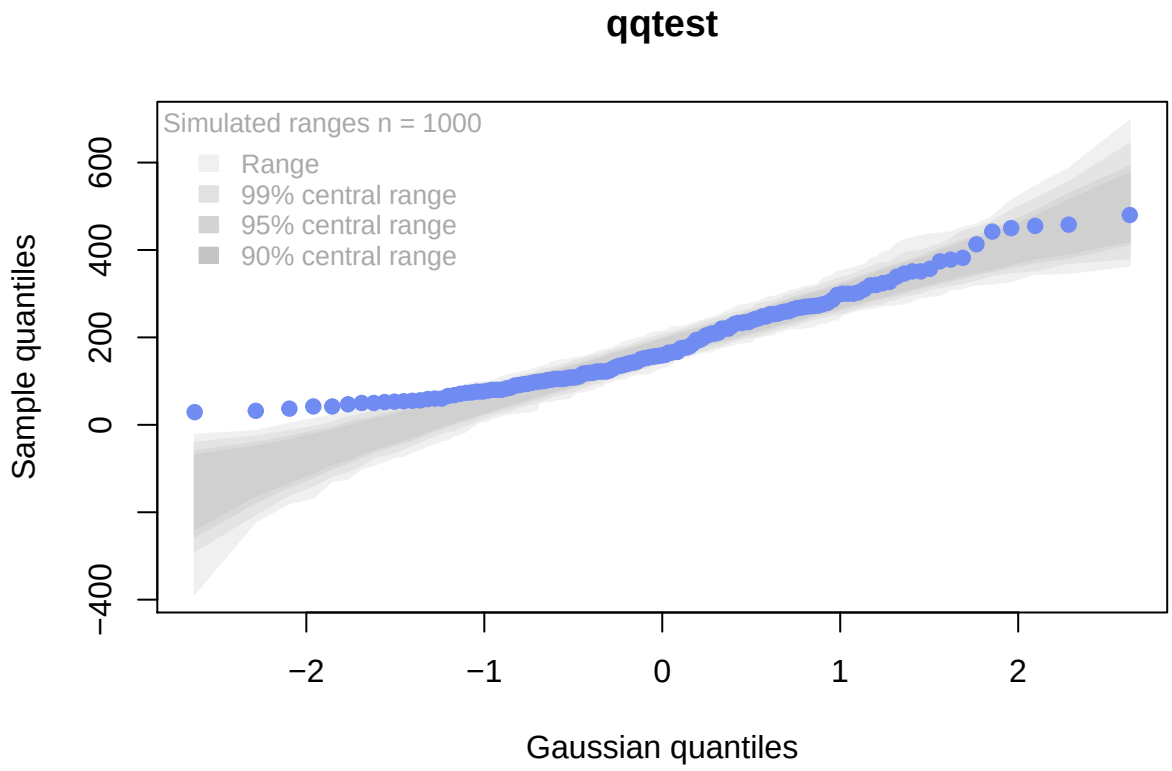


The shape of the quantile plot is nearly linear up until about the 60th percentile, at which point it begins to curve upwards. The data seems to be evenly concentraed throughout the plot, with sparsity at the right tail. This indicates a lot of variation between the extremes as the concentration of points at the left tail of the plot is relatively higher. Overall, the plot has a convex, trough-down shape.

(c) (3 marks) Use `qqtest` to construct a qqplot that compares `SSPG` to a standard normal distribution. Include envelopes in the plot. Comment on the distribution of `SSPG` and whether it might reasonably be regarded as a sample from some normal distribution. Explain your reasoning.

Important: Before every `qqtest` execute `set.seed(3124159)` so that we are all seeing the same plots.

```
library(qqtest)
# Setting the seed
set.seed(3124159)
qqtest(data=diabetes$SSPG)
```



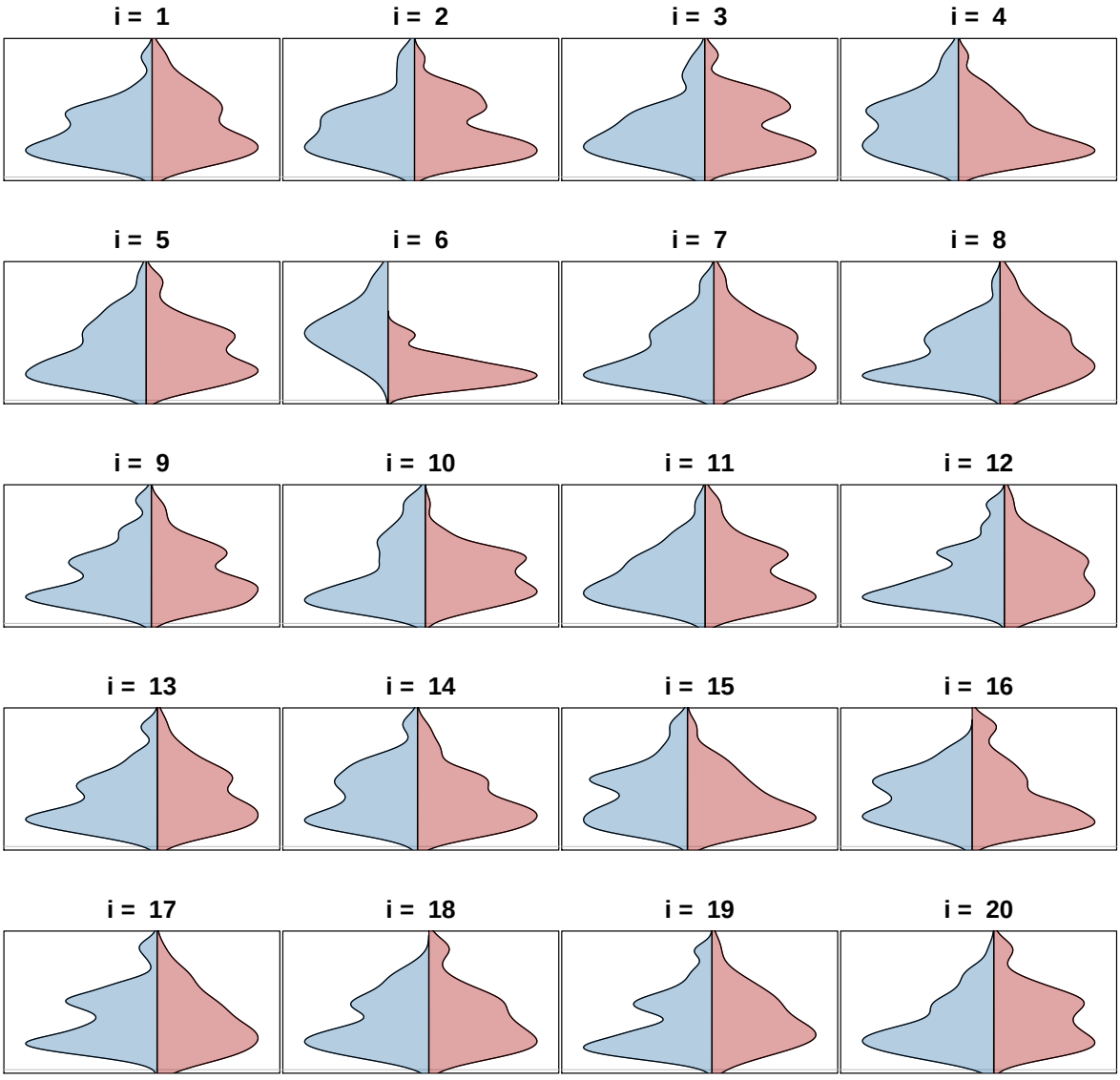
As can be seen in the plot, all the points fall within the envelope, with the exception of the points at the left tail. However, it's on average expected that $n/10 = 14.5 \approx 15$ points would fall outside of the envelope even if the SSPG was from a normal distribution, and the points outside the envelope still lie very close to it. Hence, using the `qqtest` results, it would be reasonable to regard the SSPG values as a sample from a normal distribution.

(d) The last variate, `ClinClass`, represents the classification of each patient according to the 1979 medical criteria into one of three groups: 1 = "Overt Diabetic", 2 = "Chemical Diabetic", and 3 = "Normal".

i. (4 marks) Construct a back to back density line-up plot to assess whether the normal and diabetic (chemical and overt combined) `SSPG` values come from the same distribution. Use `set.seed(3124159)` and show your code. What conclusions do you draw?

```
normal_SSPG = diabetes[diabetes$ClinClass == 3,]$SSPG
diabetic_SSPG = diabetes[diabetes$ClinClass != 3,]$SSPG

back2back <- function(data, suspectNo) {
  ylim = extendrange(c(data$x, data$y)) #c(0,600)
  Xdensity = density(data$x, bw="SJ")
  Ydensity = density(data$y, bw="SJ")
  Ydensity$y = -Ydensity$y
  xlim = extendrange(c(Xdensity$y, Ydensity$y))

  xyswitch <- function(xy_plot) {
    yx_plot = xy_plot
    yx_plot$x = xy_plot$y
    yx_plot$y = xy_plot$x
    yx_plot
  }

  plot(xyswitch(Xdensity), col="firebrick",
       xlab="", ylab="", xaxt="n", yaxt="n",
       main=paste("i = ", suspectNo),# display suspect number
       cex.main = 2, # increase suspect number size
```

```
       xlim=xlim, ylim=ylim)
  polygon(xyswitch(Xdensity), col=adjustcolor("firebrick", 0.4))
  lines(xyswitch(Ydensity), col="steelblue")
  polygon(xyswitch(Ydensity), col=adjustcolor("steelblue",0.4))
}

mixRandomly <- function(data) {
  # Note that data need not be a data frame
  # It is expected to be a list with an x and a y component
  # (possibly of different lengths)
  x <- data$x
  y <- data$y
  n_x <-length(x)
  n_y <-length(y)
  mix <-c(x,y)
  select4x <-sample(1:(n_x+n_y),n_x,replace = FALSE)
  new_x <- mix[select4x] # The mixing occurs
  new_y <- mix[-select4x]
  list(x=new_x, y=new_y)
}

lineup <- function(data, showSuspect=NULL, generateSuspect=NULL,
                   trueLoc=NULL, layout =c(5,4)) {
  # Get the number of suspects in total
  nSuspects <- layout[1] * layout[2]
  if (is.null(trueLoc)) {trueLoc <-sample(1:nSuspects, 1)}
  if (is.null(showSuspect)) {stop("need a plot function for the suspect")}
  if (is.null(generateSuspect)) {stop("need a function to generate suspect")}
  # Need to decide which subject to present
  presentSuspect <- function(suspectNo) {
    if(suspectNo != trueLoc) {data <-generateSuspect(data)}
    showSuspect(data, suspectNo)
  }
  # This does the plotting
  savePar <-par(mfrow=layout,mar=c(2.5, 0.1, 3, 0.1), oma=rep(0,4))
  sapply(1:nSuspects, FUN = presentSuspect)
  par(savePar)

  # Obfuscate location to keep us honest
  possibleBaseVals <- 3:min(2*nSuspects, 50) # remove easy base values
  possibleBaseVals <- possibleBaseVals[possibleBaseVals != 10 & possibleBaseVals != 5]
  base <-sample(possibleBaseVals, 1)
  offset <-sample(5:min(5*nSuspects, 125),1)

  # return obfuscated location
  list(trueLoc =paste0("log(",base^(trueLoc + offset),
                       ", base=",base,") - ", offset))
}

data = list(x=normal_SSPG, y=diabetic_SSPG)
set.seed(3124159)
lineup(data,
       generateSuspect = mixRandomly,
       showSuspect = back2back)
```

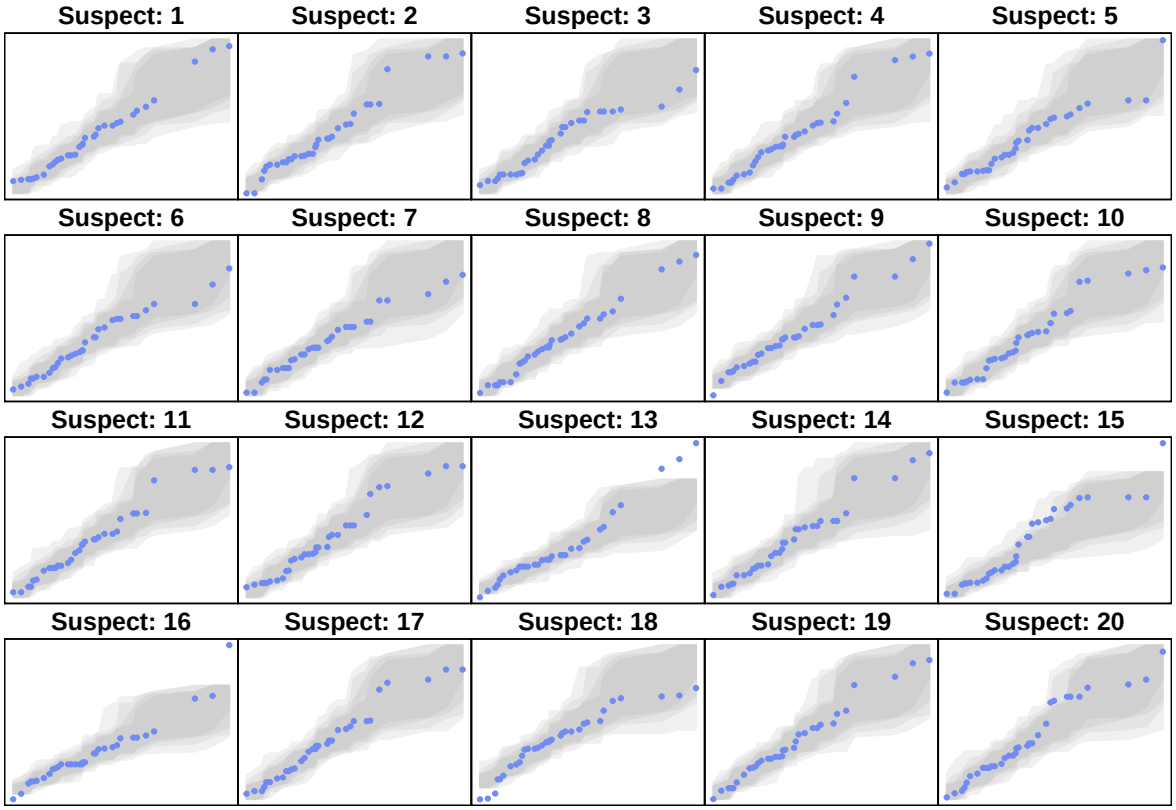



```
## $trueLoc
## [1] "log(7.51141330201283e+30, base=22) - 17"
```

From the lineup test, it is visually clear that the true plot is at $i = 6$ - which is indeed equal to the generated value of `trueLoc` - as this plot stands out more than any of the others. Hence, using this lineup test, there is enough evidence to reject the hypothesis that the two sets come from the same-shaped distribution.

ii. (4 marks) Use `qqtest` to construct a lineup plot making the same assessment, but this time assess whether the overt diabetic values of `SSPG` might have been generated from the same distribution as the normal patients. Use `set.seed(3124159)` and show your code. What conclusions do you draw?

```
set.seed(3124159)
overt_SSPG = diabetes[diabetes$ClinClass == 1,]$SSPG
qqtest(data = overt_SSPG, dataTest=normal_SSPG,
       lineup=TRUE)
```




```
## $TrueLoc
## [1] "log(2.90474294739124e+100, base=26) - 53"
```

It is difficult to see a significant difference between any of the plots. Plot 16 stands out the most personally, but this differs from the `trueLoc` at 18. Hence, using this lineup test, even though in all plots the overt diabetic SSPG values fall within the envelope, there is not enough evidence to reject the hypothesis that the overt diabetic values of `SSPG` might have been generated from the same distribution as the normal patients. The qqtests seem to be producing a type II error.

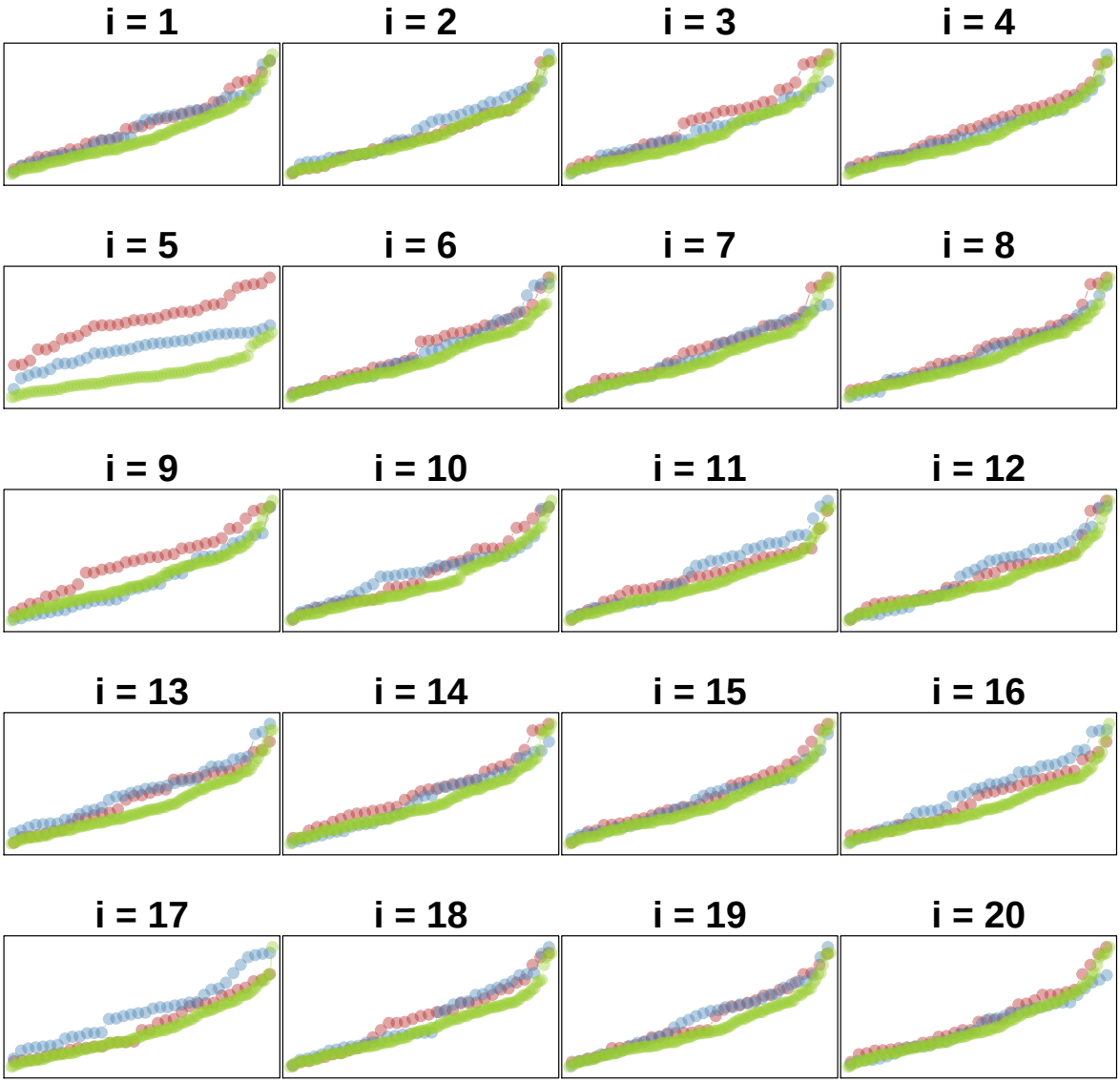
iii. **Grad students, bonus undergraduates** (8 marks) Consider the following code:

```
data <- list(x=x, y=y, z=z)
lineup(data,
generateSuspect = mixRandomly,
showSuspect = myQuantilePlot,
layout=c(5,4))
```

The function `mixRandomly` will need to be rewritten to handle `data` being a list of three samples. Write the function `myQuantilePlot` so that it overlays the sample quantile functions of each of `x`, `y`, and `z` in the same display using different colours. Hand in your code for these two functions and illustrate the outcome (using `set.seed(314159)`) on `SSPG` for the three different clinical classes. Comment on your findings.

```
mixRandomly <- function(data) {
  # Note that data need not be a data frame
  # It is expected to be a list with an x, a y and a z component
  # (possibly of different lengths)
  x = data$x
  y = data$y
  z = data$z
  n_x = length(x)
  n_y = length(y)
  n_z = length(y)
```

```
  mix = c(x,y,z)
  full_range = 1:(n_x+n_y+n_z)
  select4x = sample(full_range,n_x,replace = FALSE)
  # remove indices in select4x from sampling pool and sample indices for y
  select4y = sample(full_range[-select4x],n_y,replace = FALSE)
  new_x = mix[select4x] # The mixing occurs
  new_y = mix[select4y]
  new_z = mix[-select4y]
  list(x=new_x, y=new_y, z=new_z)
}

# should there be an option to specify colors or can we choose any?
myQuantilePlot <- function(data, suspectNo) {
  ylim = extendrange(c(data$x, data$y, data$z))
  n_x = length(data$x)
  n_y = length(data$y)
  n_z = length(data$z)
  p_x = ppoints(n_x)
  p_y = ppoints(n_y)
  p_z = ppoints(n_z)
  plot(p_x, sort(data$x), type="b", col=adjustcolor("firebrick", 0.4),
       pch=19, cex=2, ylim = ylim,
       main=paste("i =", suspectNo), # display suspect number
       cex.main = 3,# increase suspect number size
       ylab="", xlab="", xaxt="n", yaxt="n")
  points(p_y,sort(data$y), type="b",
         col=adjustcolor("steelblue", 0.4),  pch=19, cex=2)
  points(p_z,sort(data$z), type="b",
         col=adjustcolor("yellowgreen", 0.4),  pch=19, cex=2)
}

chemical_SSPG = diabetes[diabetes$ClinClass == 2,]$SSPG
sspg_data = list(x=overt_SSPG, y=chemical_SSPG, z=normal_SSPG)
set.seed(314159)
lineup(sspg_data,
       generateSuspect = mixRandomly,
       showSuspect = myQuantilePlot,
       layout=c(5,4))
```




```
## $trueLoc
## [1] "log(6.6790315523026e+66, base=39) - 37"
```

From the lineup test, it is visually clear that the true plot is at $i = 5$ - which is indeed equal to the generated value of `trueLoc` - as in this plot, none of the graphs never overlap unlike every other plot. Hence, using this lineup test, there is enough evidence to reject the hypothesis that the three sets come from the same-shaped distribution.

2. Download the `olive` data from the course website. In that file, there is a dataset on the fatty acid content of 572 different Italian olive oils. In total eight different fatty acids are measured. All olive oils come from one of nine different olive growing regions in Italy.

Once you load this file into your R session there will be a data set called `olive`.

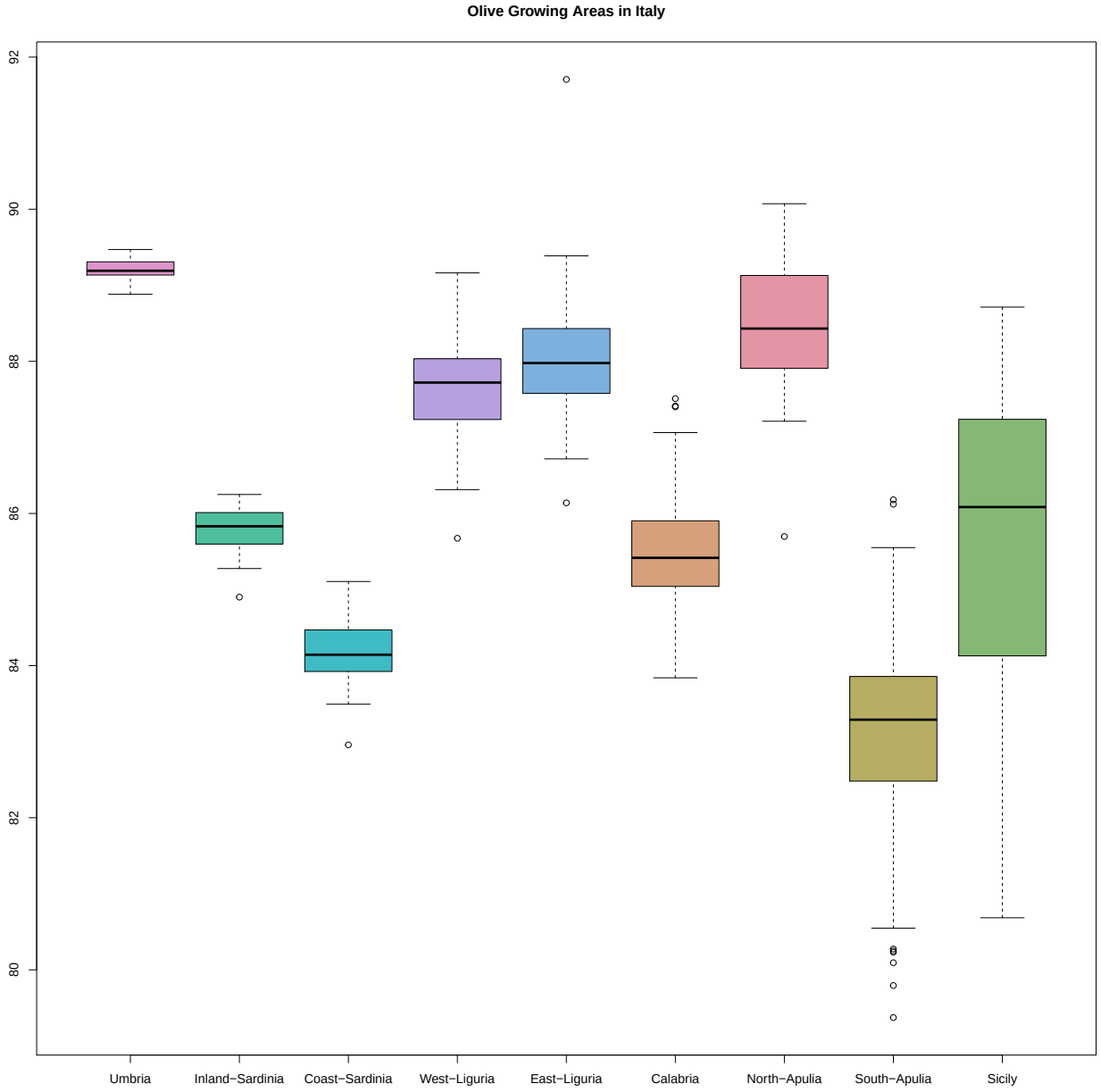
In this question you are going to focus on the fatty acid `oleic`.

(a) (2 marks) Separate the data on `oleic` into 9 different groups as defined by the olive growing `Area`, and draw side by side boxplots of all 9 groups. Colour the boxplots uniquely using

```
library(colorspace)
cols <- rainbow_hcl(9) # Use these colours.
```

Show your code together with your output.

```
groups = with(olive, split(sqrt(oleic), Area))
ord = order(sapply(groups, IQR))
boxplot(groups[ord], col=cols[ord],
        main="Olive Growing Areas in Italy")
```



(b) Load the `R` package `PairViz`. Use the variate `oleic` and the same colours for the olive growing areas as in part (a) throughout the following:

```
library(PairViz)
```

```
## Loading required package: TSP
## Loading required package: gtools
## Loading required package: graph
## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
```

```
##     parLapplyLB, parRapply, parSapply, parSapplyLB
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, cbind, colMeans,
##     colnames, colSums, do.call, duplicated, eval, evalq, Filter,
##     Find, get, grep, grepl, intersect, is.unsorted, lapply,
##     lengths, Map, mapply, match, mget, order, paste, pmax,
##     pmax.int, pmin, pmin.int, Position, rank, rbind, Reduce,
##     rowMeans, rownames, rowSums, sapply, setdiff, sort, table,
##     tapply, union, unique, unsplit, which, which.max, which.min
```

i. (3 marks) Suppose we wish every pair of boxplots to appear next to one another in the same plot.

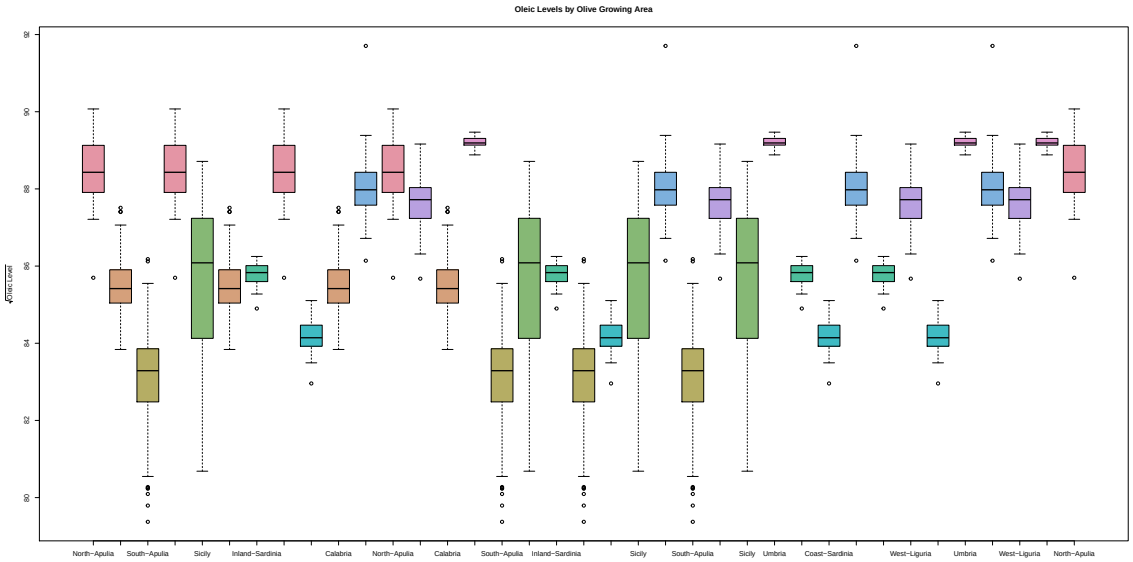
- How many such pairwise comparisons exist?

  Since we do not care about the order of the boxplots in the comparisons, the number of ways that we can choose **2** boxplots to compare out of the total **9** is $C(9,2) = \frac{9!}{7!2!} = \frac{9(8)}{2} = 36$. Hence, **36** pairwise comparisons exist.

- Give the code that will construct this display (without any other constraint on the ordering).

- Show the display which resulted from your code.

```r
# Make this bigger and shorten names
n = length(groups)
ordEul = eulerian(n)
boxplot(groups[ordEul], col=cols[ordEul],
        ylab=expression(sqrt("Oleic Level")),
        main="Oleic Levels by Olive Growing Area")
```

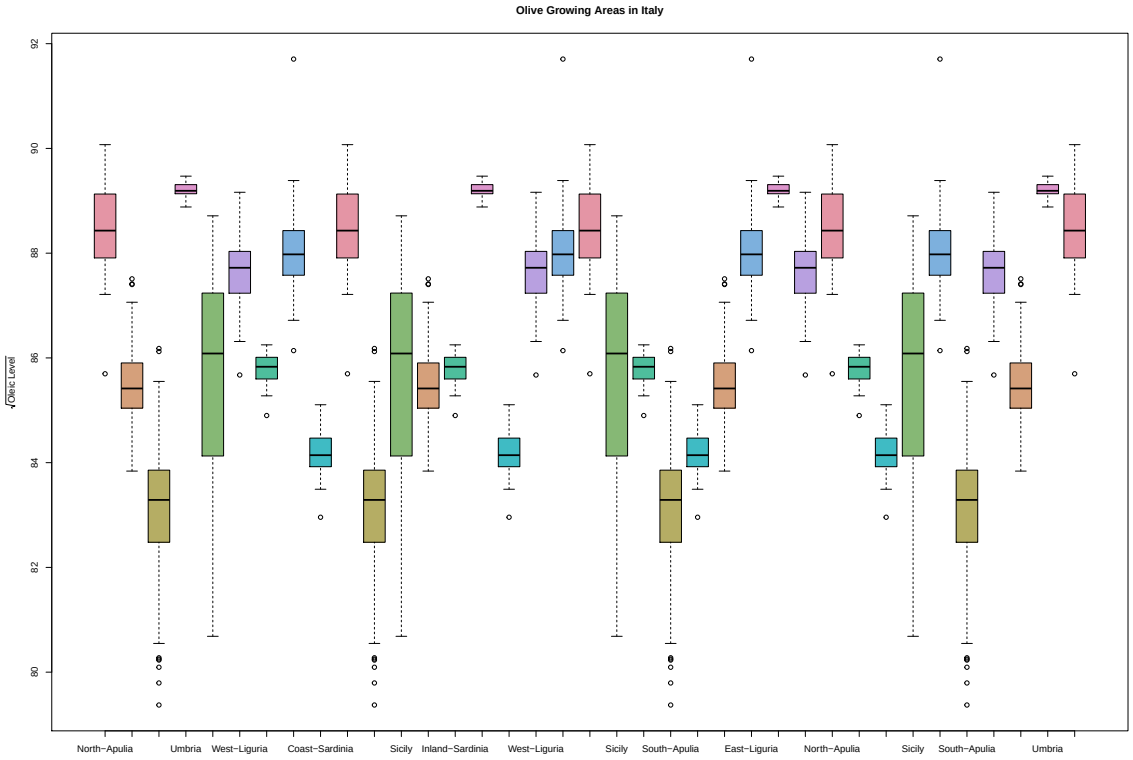


ii. (5 marks) Suppose we wish every pair of boxplots to appear next to one another in the same plot but that the boxplots should be grouped so that all areas appear only once in each group.

- Maintaining the same colours for the areas as before, give the code that will construct this display (without any other constraint on the ordering).

- Show the display which resulted from your code.

```
ordHam = hpaths(n, matrix=FALSE)
boxplot(groups[ordHam], col=cols[ordHam],
        ylab=expression(sqrt("Oleic Level")),
        main="Olive Growing Areas in Italy")
```

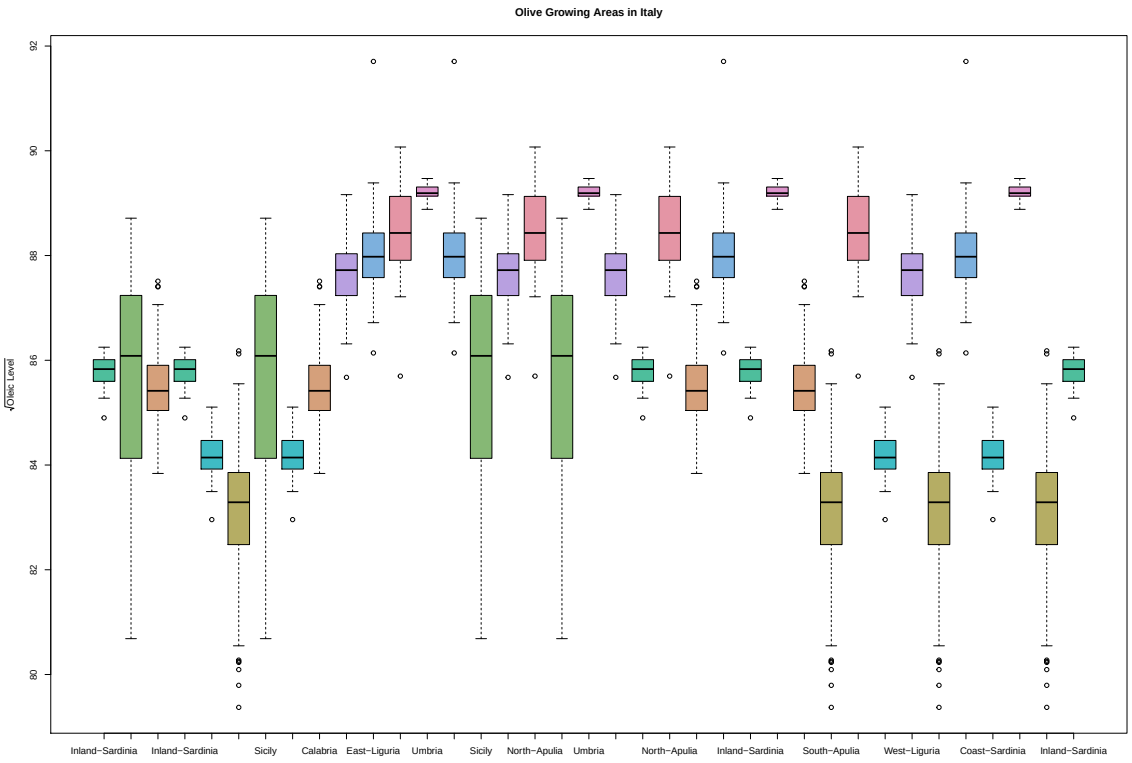


iii. (7 marks) Construct t tests for every pair of olive growing areas (recall `pairwise.t.test` from class). Use the significance levels from these tests to construct an ordering of the boxplot pairs, one which favours having the most significantly different pairs at the left of the display.

- Show your code.
- Show the resulting display.

```
# running the pairwise t-tests
t_tests = with(olive,
               pairwise.t.test(sqrt(oleic), Area))

# creating a matrix of weights set to the sig. levels of the t-tests
weights = matrix(0, nrow=n, ncol=n)
rownames(weights) = names(groups)
colnames(weights) = names(groups)
weights[2:n, 1:(n-1)] = t_tests$p.value

# getting rid of the NA values and fixing the weights matrix
diag(weights) = 0
for(i in 1:(n-1)) {
  for(j in (i+1):n) {
    weights[i,j] = weights[j,i]
  }
}

highlowEulOrd = eulerian(-weights)
boxplot(groups[highlowEulOrd], col=cols[highlowEulOrd],
        ylab=expression(sqrt("Oleic Level")),
```

```
        main="Olive Growing Areas in Italy")
```



- Does the ordering perfectly arrange the boxplots so that for any pairwise comparison, those to the left are more significant and those to the right are less significant?

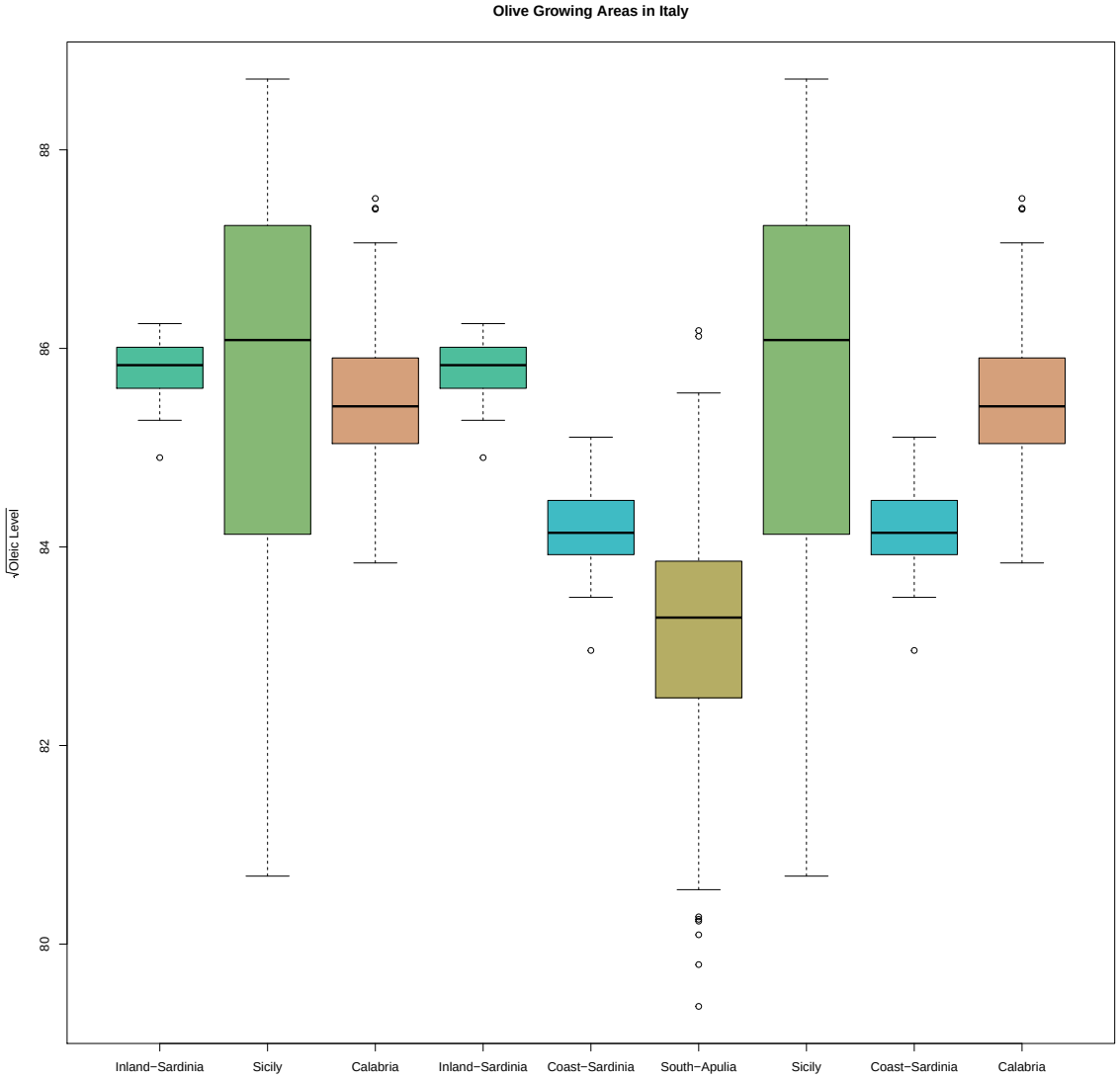
  No. The majority of the medians of the boxplots pairs get closer in value, i.e more identical, as the diagram moves from left to right, and the pairs become easier to compare as a result because the boxplots become more in line with each other on the vertical scale. So most of the pairwise comparisons are ordered correctly. However, some of them (e.g. the comparison between Sicily and Coast-Sardinia) are less significant than the comparisons to their right, and should be located further to the right of the plot.

- Explain why the ordering was successful (or unsuccessful) in this way.

  This is likely because a Eulerian path cannot have an edges walked through again. Hence, once the Eulerian has traversed an edge and arrived at a node/region, it will choose to take the next shortest path possible from that node and it cannot travel through edges it has already visited to get to the a different path. Hence, when creating the ordering for the boxplots, the next comparison is determined by the next most significant pairwise comparison for the node that is currently being visited, and this may not always which may not be necessarily the next most significant pairwise comparison overall.

- Show a display showing only the first 8 comparisons.

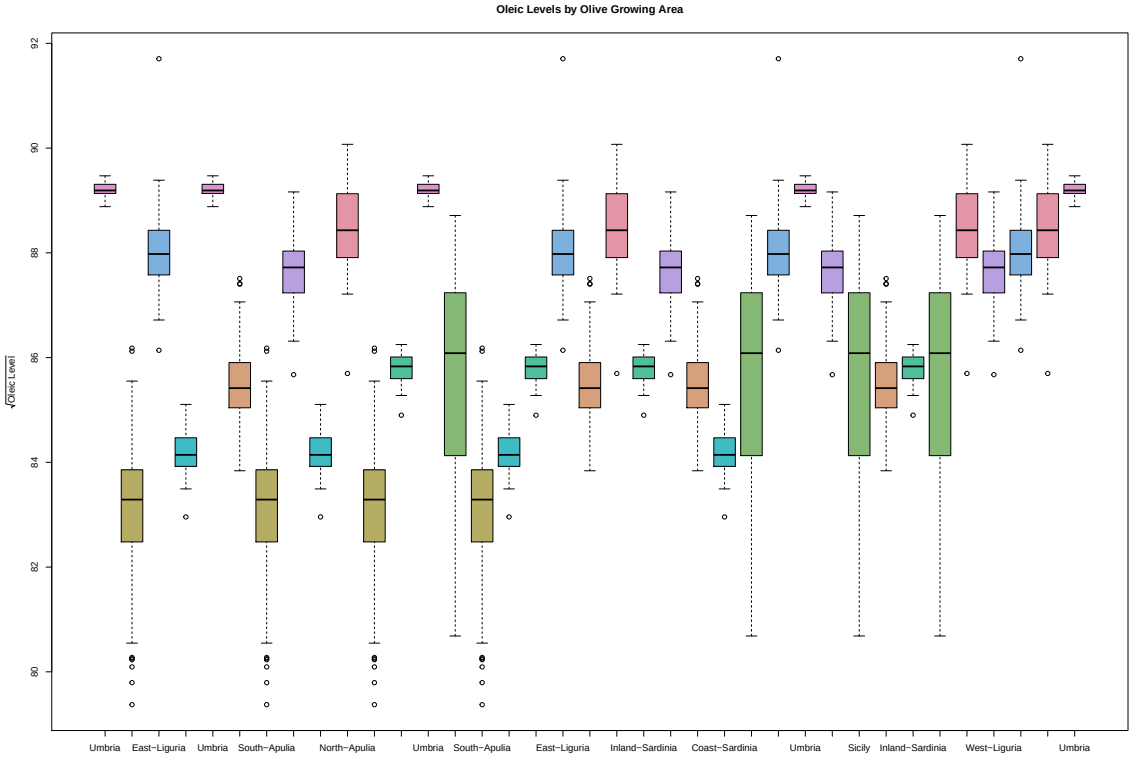
```
mostSig8 = highlowEulOrd[1:9]

boxplot(groups[mostSig8],
        col=cols[mostSig8],
        ylab=expression(sqrt("Oleic Level")),
        main="Olive Growing Areas in Italy")
```



iv. (7 marks) Use the significance levels from part (iii) but now order the boxplots so that the **least** significant differences appear earliest in the sequence from left to right.

- Show your code.
- Show the resulting display.

```
lowhighEulOrd = eulerian(weights)
boxplot(groups[lowhighEulOrd], col=cols[lowhighEulOrd],
        ylab=expression(sqrt("Oleic Level")),
        main="Oleic Levels by Olive Growing Area")
```



- Does the ordering perfectly arrange the boxplots so that for any pairwise comparison, those to the left are less significant and those to the right are more significant?

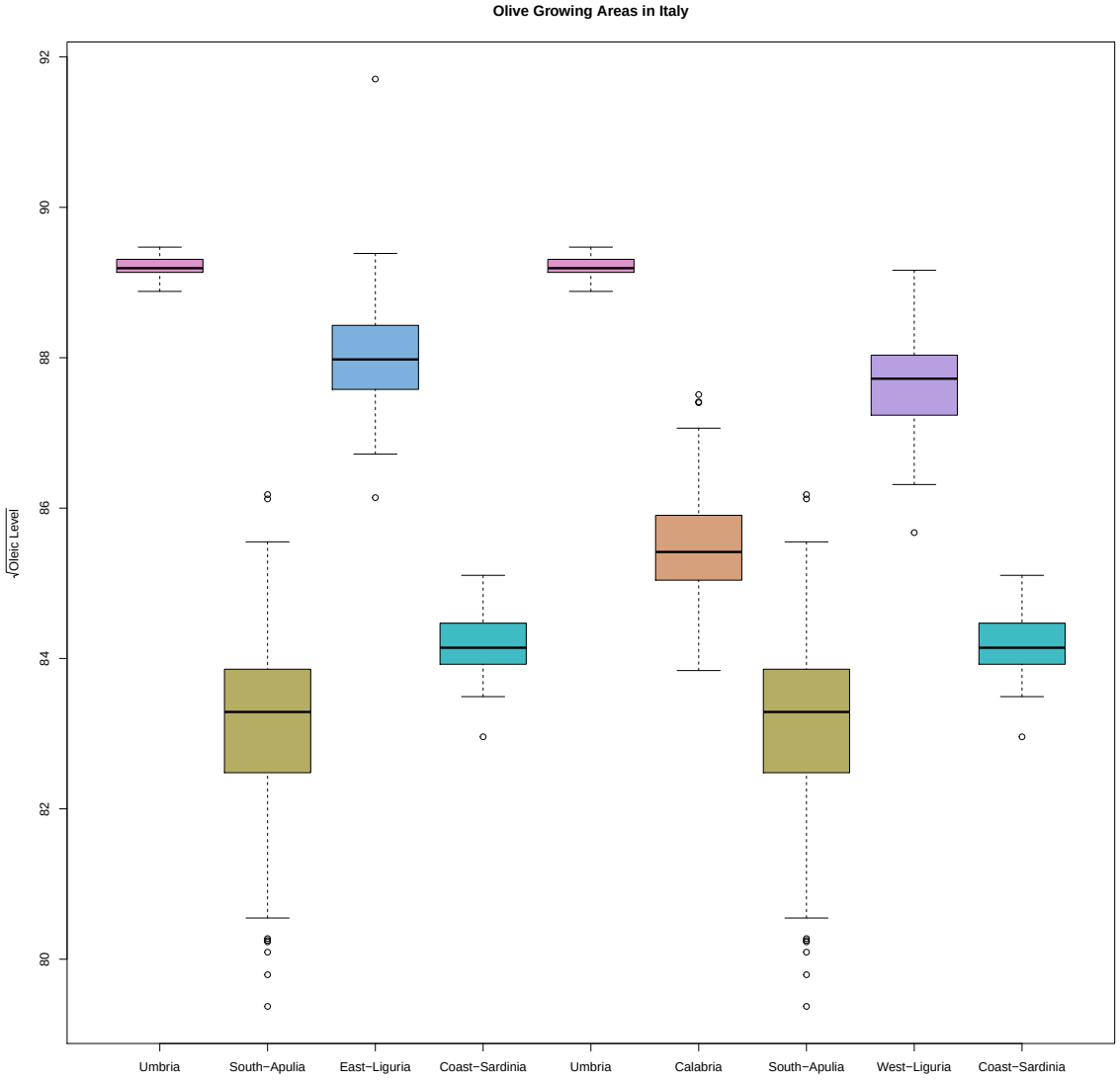
  No. For the most part, the medians of the boxplots pairs get further away in value, i.e less identical, as the diagram moves from left to right, and the pairs become more to compare as a result because the boxplots become more out of line with each other on the vertical scale. So most of the pairwise comparisons are ordered correctly. However, some of them (e.g. the comparison between South-Apulia and Coast-Sardinia) are more significant than the comparisons to their right, and should be located further to the right of the plot.

- Explain why the ordering was successful (or unsuccessful) in this way.

  This was unsuccessful for the same reasons as mentioned in the previous question: since the algorithm for `eulerian` is greedy and an edge that has already been traversed cannot be visited again, the next comparison is determined by the next least significant pairwise comparison for the node/region that is currently being visited, and this may not always which may not be necessarily the next least significant pairwise comparison overall.

- Show a display showing only the first 8 comparisons.

```
leastSig8 = lowhighEulOrd[1:9]

boxplot(groups[leastSig8],
        col=cols[leastSig8],
        ylab=expression(sqrt("Oleic Level")),
        main="Olive Growing Areas in Italy")
```



v. (2 marks) Is the sequence used in part (iii) the reverse of that in part (iv)?

No, it is not.

- If it is, must it be the reverse?

  No, it doesn't because there are 36 unique pairwise comparisons that can be made, so the the set of the 8 most significant ones and the set of the 8 least signifcant ones (but in reverse order) will not be the same, i.e. will not fully overlap, as this could only happen if the Eulerian traversed the same edges again, which by definition does not occur.

- If it is not, is it impossible for it to be the same?

  For this type of comparison, it is possible.

- Either way, explain your reasoning.

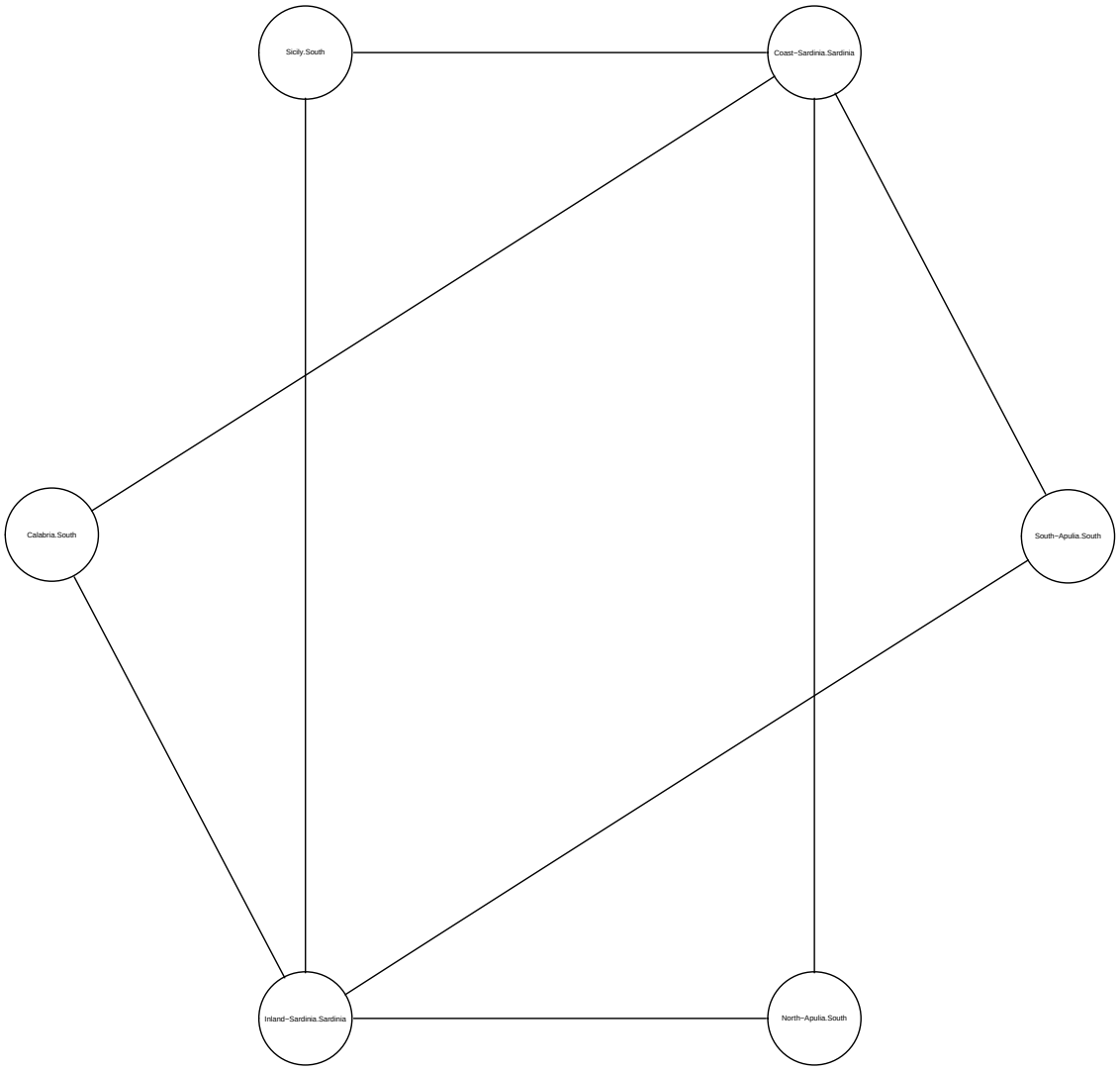
  Note that the plots of all the comparisons from most significant to least is the reverse of the plots from least to most. Hence, the only way that it would be possible for part (iii) to be the reverse of part (iv) is if there were only 8 pairwise comparisons in total to consider, instead of 36.

(c) The olive growing areas are divided into three different regions: North, South, and Sardinia. In this part of the question, interest lies only in comparisons between each growing area in the south and each area in Sardinia. That is, each southern area (4 areas) is to be compared to each Sardinian area (2 areas) yielding a total of 8 comparisons of interest.

i. (4 marks) Having loaded `PairViz`, create a graph having all six areas in the South and Sardinia as nodes and with edges between every pair whose comparison is of interest.

- plot this graph
- show the code used to create the graph and to plot it.

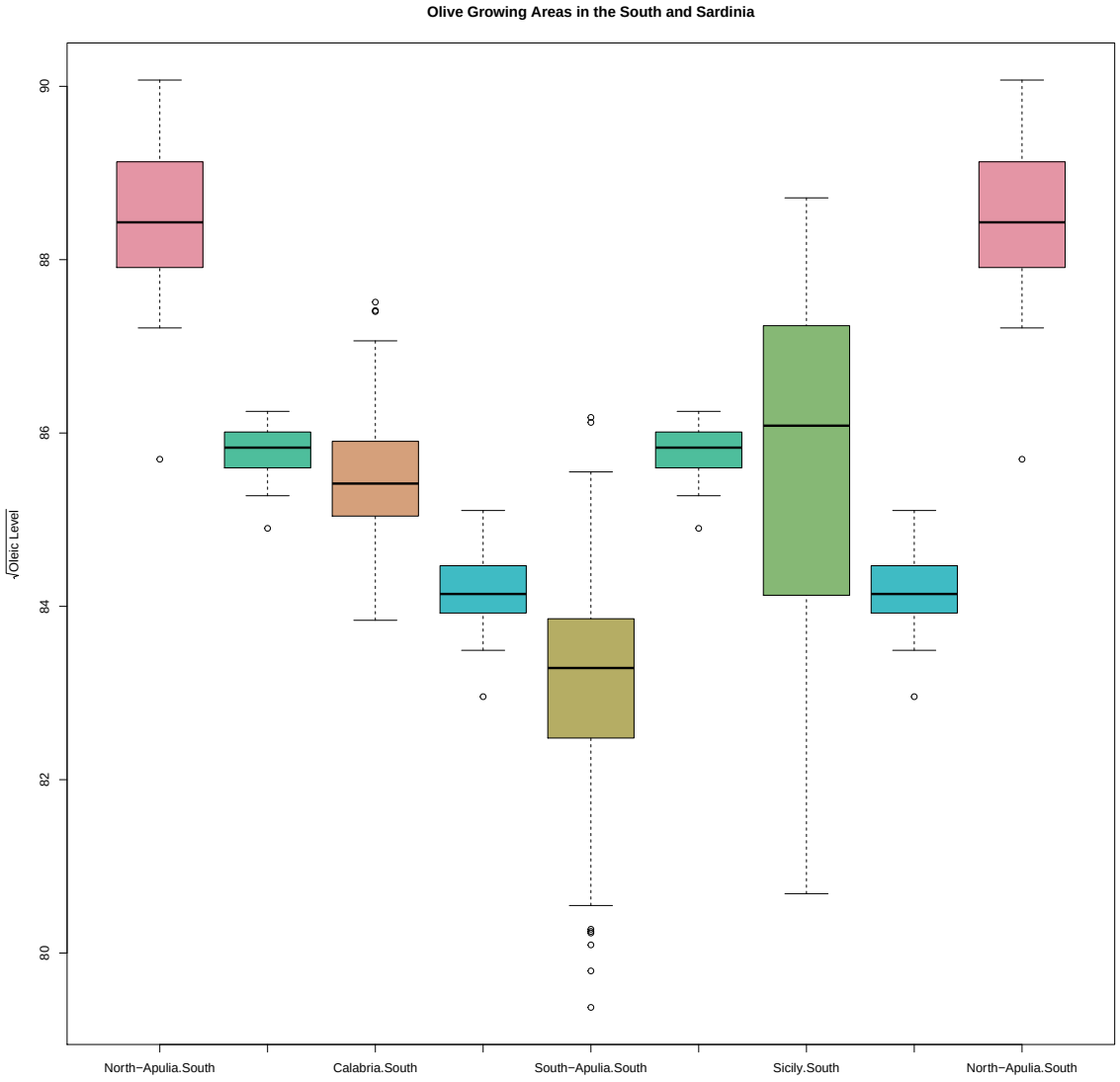
```
regional = olive[olive$Region %in% c('South','Sardinia'),]
regions = with(regional, split(sqrt(oleic), list(Area,Region), drop=TRUE))
nodes = names(regions)
SouthRegions = nodes[endsWith(nodes, "South")]
SardiniaRegions = nodes[!endsWith(nodes, "South")]
RegionGraph = new("graphNEL", nodes=nodes, edgemode="undirected")
for (i in 1:4) {
  stregion = SouthRegions[i]
  for (j in 1:2) {
    sdregion = SardiniaRegions[j]
    RegionGraph = addEdge(stregion, sdregion, RegionGraph)
  }
}

plot(RegionGraph, "circo")
```



ii. (4 marks) Using the graph from part (i), construct an Eulerian and use that Eulerian to produce a sequence of boxplots that show the comparisons of interest.

- show the boxplot display
- show the code used to construct the Eulerian and the display.

```
ssOrd = eulerian(RegionGraph)

colEulOrd = rep(0, length(ssOrd))
for (i in 1:length(ssOrd)) {
  colEulOrd[i] =  which(nodes==ssOrd[i])
}

boxplot(regions[ssOrd], col=cols[colEulOrd],
        ylab=expression(sqrt("Oleic Level")),
        main="Olive Growing Areas in the South and Sardinia")
```



3. **Antarctic sea ice** On the website for this assignment, in the directory called `R` you will find a file called `seaice.csv`. Download this file.

   We will need to do some manipulations to this data before it can be easily used in our analysis.

```
seaice <- read.csv("./R/seaice.csv", header=TRUE)
```

The last line above shows that `seaice` is a data frame having four variates. The first three identify the year, month, and day that the last measurement was taken. The last measure is a determination of the extent of Antarctic sea ice in millions of square kilometres as determined by satellite imagery.

(a) *Irregular time series* First, we begin by putting the year, month, and day together into a single date. Do this as follows:

```
date <- paste(seaice$Year,seaice$Month,seaice$Day, sep=" ")
head(date)
```

```
## [1] "1979 1 2"  "1979 1 4"  "1979 1 6"  "1979 1 8"  "1979 1 10" "1979 1 12"
```

```
# The following function turns that into a standard date format
# You will need the package "lubridate"
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

```
date <- ymd(date)
head(date)
```

```
## [1] "1979-01-02" "1979-01-04" "1979-01-06" "1979-01-08" "1979-01-10"
## [6] "1979-01-12"
```

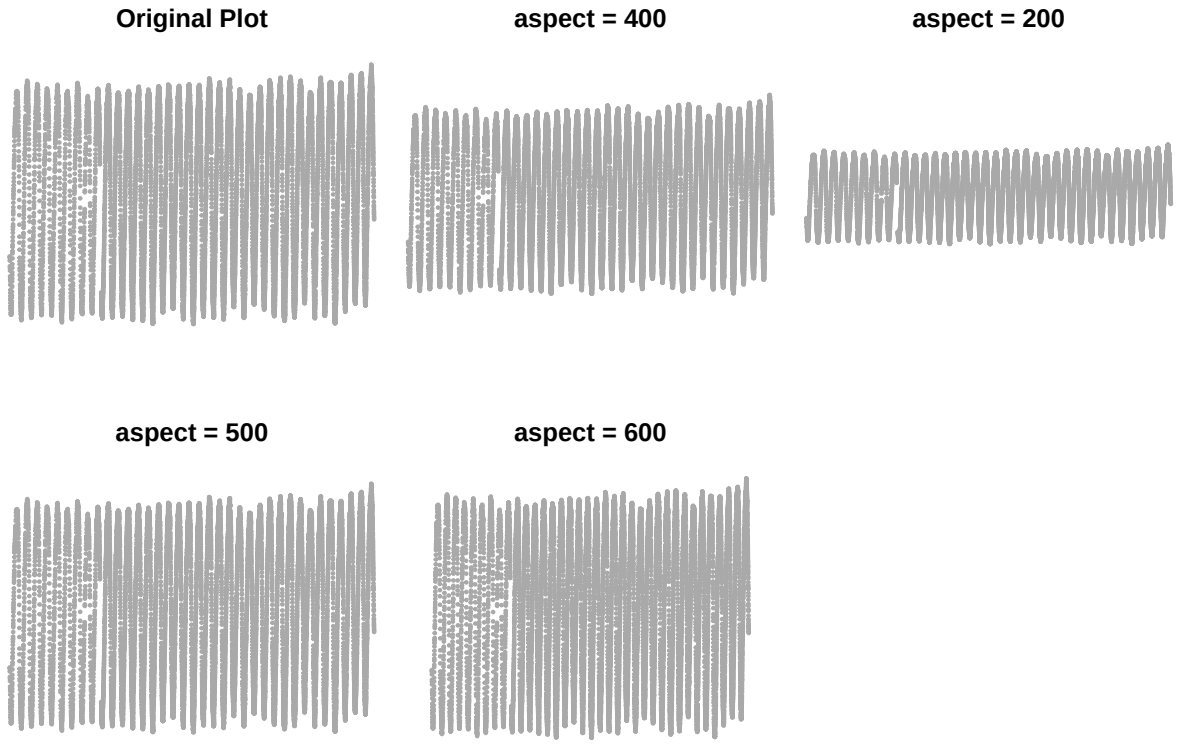
```
# And we create a new data structure called sea.ice
sea.ice <- data.frame(date=date, extent = seaice$Extent)
head(sea.ice)
```

```
##         date extent
## 1 1979-01-02  6.945
## 2 1979-01-04  6.841
## 3 1979-01-06  6.648
## 4 1979-01-08  6.270
## 5 1979-01-10  6.138
## 6 1979-01-12  5.957
```

We will be using the data frame `sea.ice` for this part of the question.

i. **(4 marks)** Call `plot(...)` directly on the data frame `sea.ice` with `cex=0.3, pch=19, col="darkgrey"`. Zoom in on this plot and experiment with different aspect ratios by reshaping the plot's window. Describe whatever interesting characteristics you see in the time series.

```
# Aspect ratio
savePar = par(mfrow=c(2,3), mar=c(2.5,0.1,3,0.1))
plot(sea.ice, xlab="", ylab="", axes=FALSE,
     main="Original Plot", cex=0.3, pch=19, col="darkgrey")
for (aspect in c(400, 200, 500, 600)) {
  plot(sea.ice, asp=aspect,
       main=paste("aspect =",aspect),
       xlab="", ylab="", axes=FALSE,
       cex=0.3, pch=19, col="darkgrey")
}
par(savePar)
```




```
# Zooming with loon
require(loon)
```

```
## Loading required package: loon
```

```
## Loading required package: tcltk
```

```
##
## Attaching package: 'loon'
```

```
## The following object is masked from 'package:graph':
##
##     complement
```

```
l_plot(sea.ice, showScales=TRUE,
     size=0.3, glyph="ccircle",
     color="darkgrey")
```

```
## [1] ".l0.plot"
## attr(,"class")
## [1] "loon"
```

The value of the extent of the Antartic sea ice stays within the same range, but it continuously sharply decreasing and increasing over short time periods.

ii. Decomposition into a long term trend and a short term trend. First we need to transform the data again.

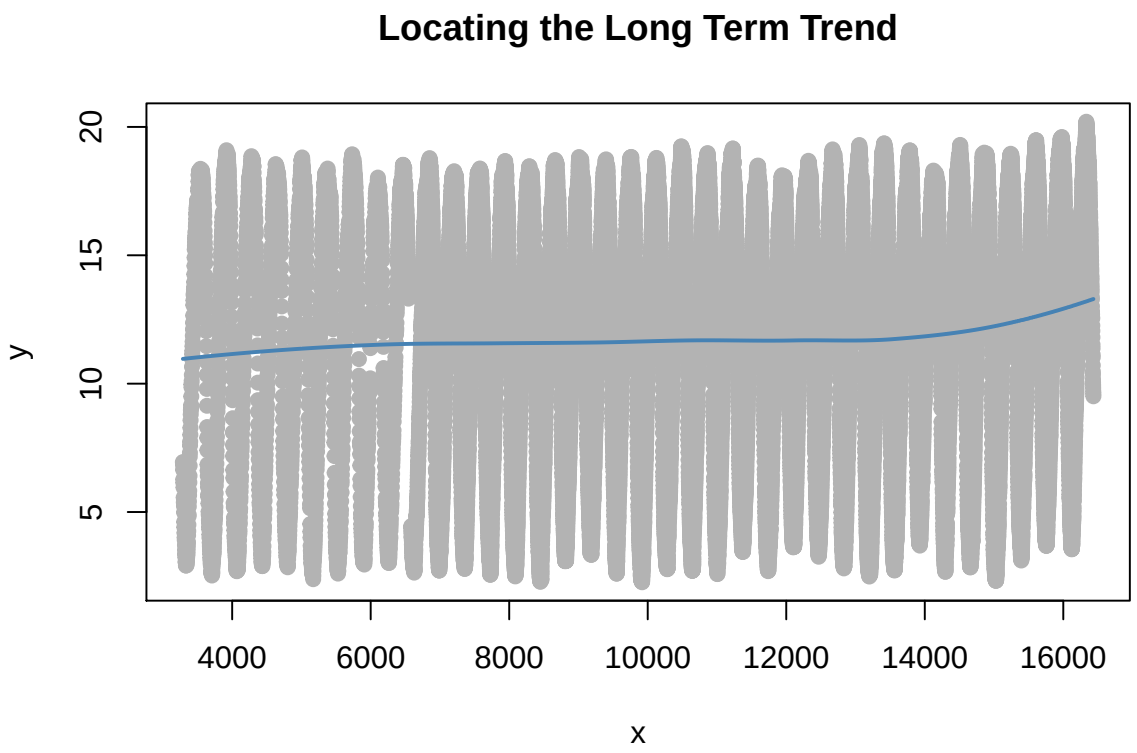
```
sea.ice.xy <- xy.coords(sea.ice)
y <- sea.ice.xy$y
x <- sea.ice.xy$x
# plotting x and y as before will yield the same plot, but
# with the dates lost.
#plot(x, y, cex=0.3, pch=19, col="darkgrey")
```

The data range from the beginning of 1979 to the end of 2014. The number of years covered then is (2014-1979 + 1)=36. So each year is approximately 1/36 of the series. A month is about 1/12 of this

again.

- **(2 marks)** Using `loess(..)` as in class, find a long term trend using `span=2/3`. Add this trend as a curve in a different colour on top the points (as in the above plot). Describe the fit.

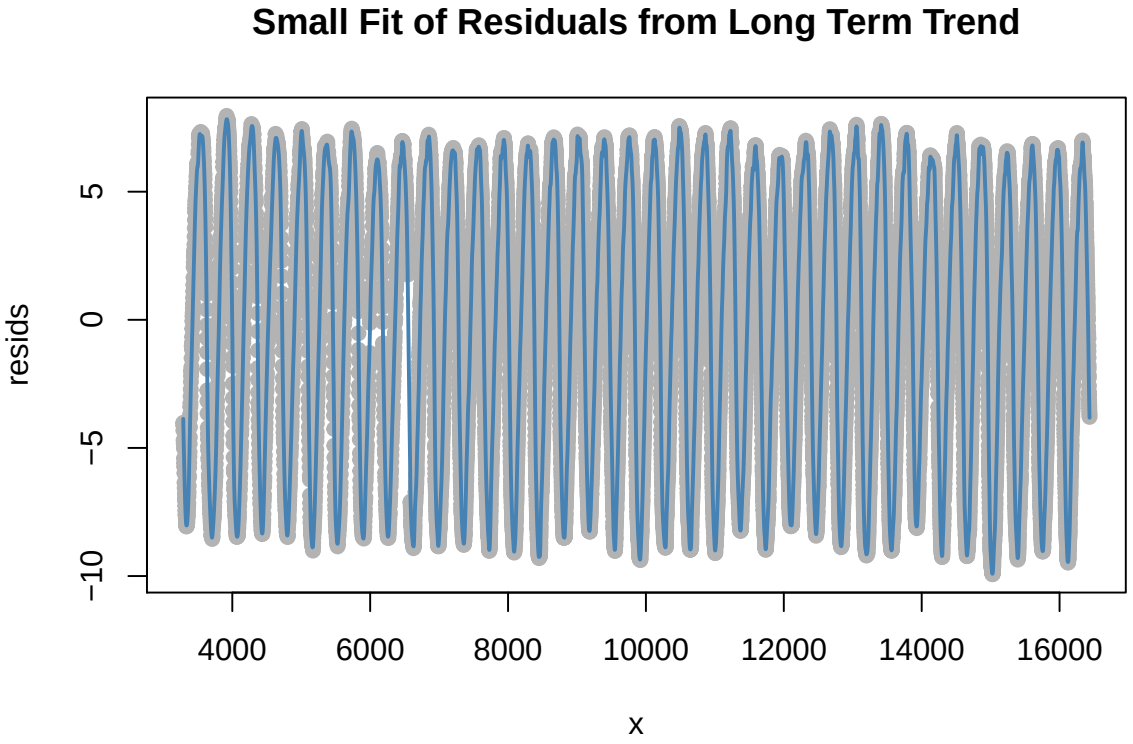
```
ltfit = loess(y~x, span=2/3)
ltpred = predict(ltfit)
xord = order(x)
plot(x, y, col="grey70", pch=19,
     main="Locating the Long Term Trend")
lines(x[xord], ltpred[xord], lwd=2, col="steelblue")
```



The extent of Antarctic sea ice stays fairly consistent and stationary, then there is a small jump upwards and the extent begins to trend upwards (i.e. increase) slowly.

- **(2 marks)** Now fit a `loess` short term smooth with `span=1/(36*12)` to the residuals from the long term trend. Plot the residuals from the long-term trend and add the fitted short term trend. Describe the fit.

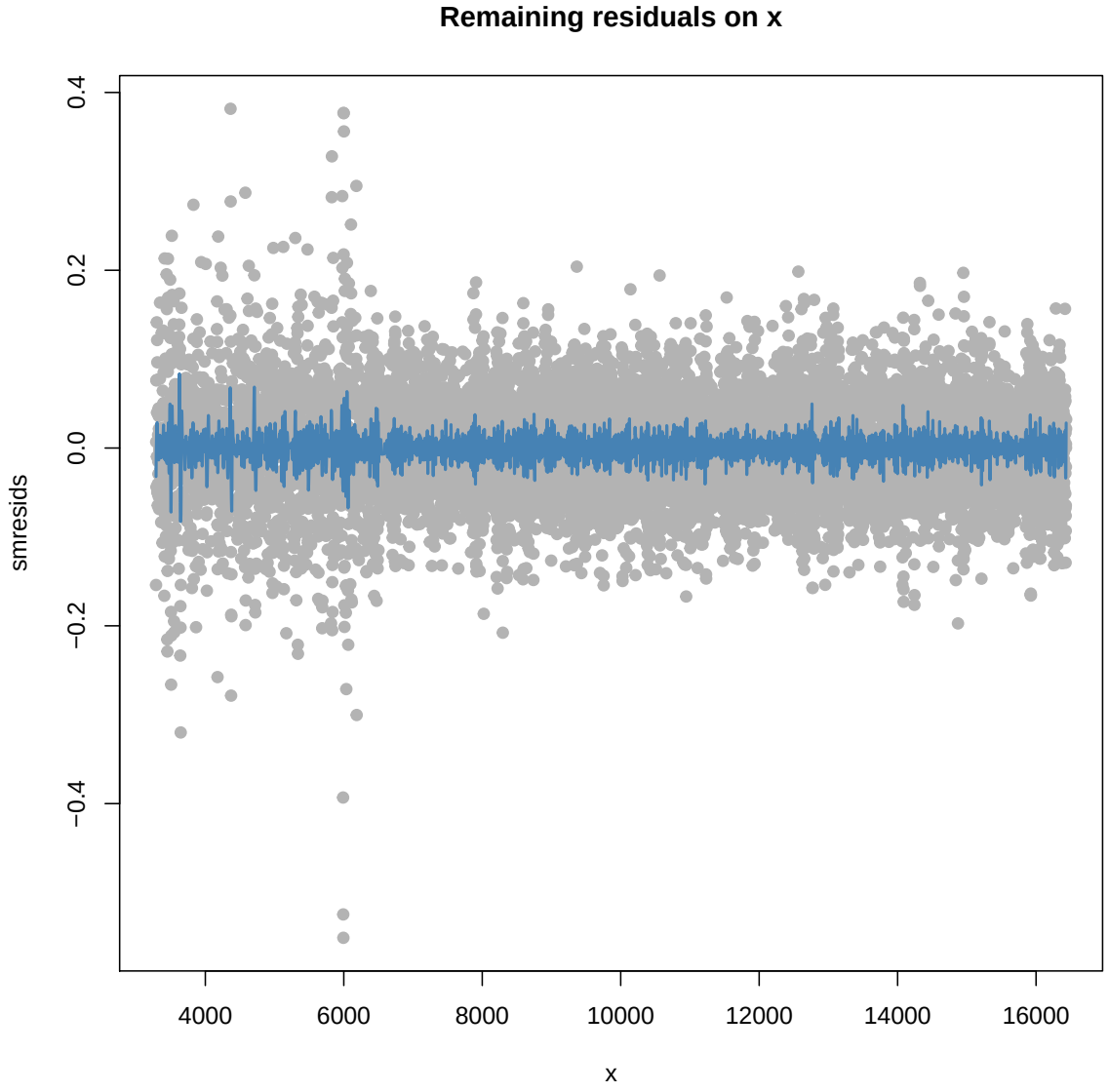
```
resids = ltfit$residuals
plot(x, resids, col="grey70", pch=19,
     main="Small Fit of Residuals from Long Term Trend")
smfit = loess(resids~x, span=1/(36*12))
smpred = predict(smfit)
lines(x[xord], smpred[xord], lwd=2, col="steelblue")
```



The residuals exhibit a consistently seasonal trend - it trends upward and the downwards very quickly for about every $333\frac{1}{3}$ units of x, staying with the range of about $[-10, 10]$.

- **(3 marks)** Plot the residuals from the short-term trend. Describe any patterns you find.

```
smresids = smfit$residuals
plot(x, smresids, col="grey70", pch=19,
     main="Remaining residuals on x")
rfit = loess(smresids~x, span=1/(36*12))
rpred = predict(rfit)
lines(x[xord], rpred[xord], lwd=2, col="steelblue")
```



There is no consistent trend in the plot - the remaining residuals are too random to locate such a distinct pattern.

- **(2 marks)** Explain your findings.

  The graphs represent the different parts of the decomposition of the time series model of the Antartic sea ice extent, with the first long term trend of the extent being the main trend in the model, the short term trend of the residuals representing the seasonality of the model, and the remaining residuals representing the remaining randomness in the model.

(b) **Monthly averages** The original series was very irregular. Here we replace it with a complete series using the average extent for each month. This time series is created as follows.

```
months <- unique(seaice$Month)
nmonths <- length(months)
years <- unique(seaice$Year)
nyears <- length(years)
# check the beginning and the end months.
seaice[1,]
```

```
##   Year Month Day Extent
## 1 1979     1   2  6.945
```

```
seaice[nrow(seaice),]
```

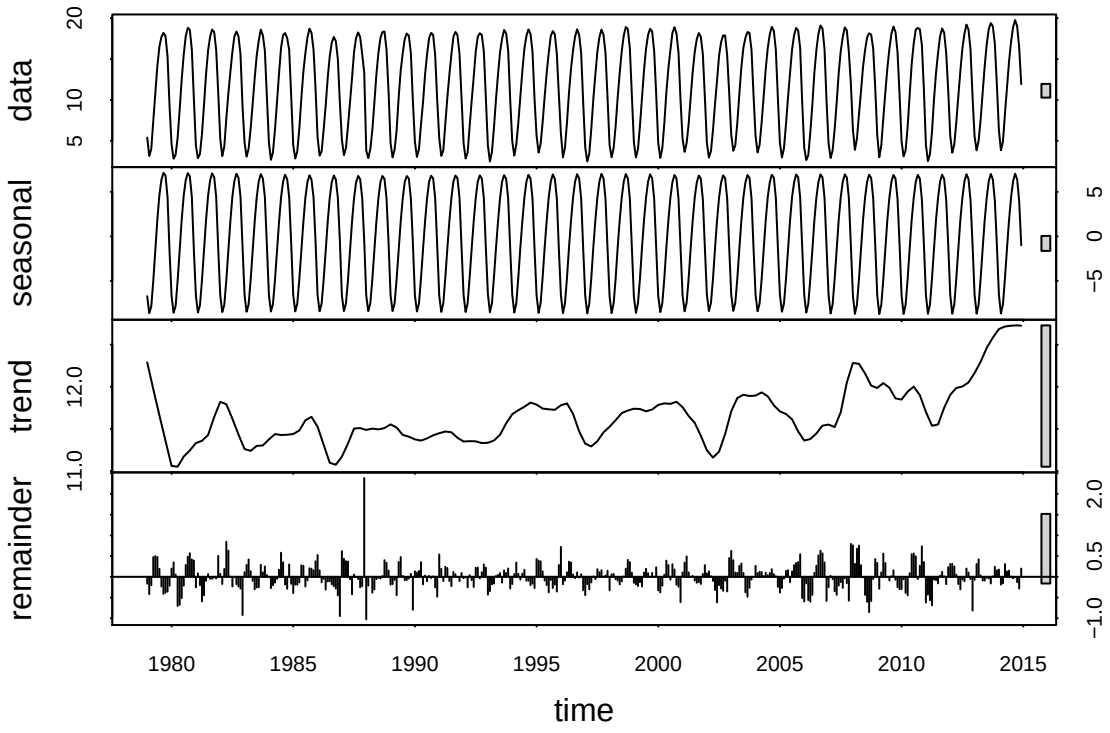
```
##       Year Month Day Extent
## 11552 2014    12  31  9.508
```

```
maxN <- nmonths * nyears
# place holder
iceMonthly <- vector("numeric", length=maxN)
for (i  in 1:nyears) {
  year <- seaice[,"Year"] == years[i]
  iceyear <- seaice[year,c("Month","Extent")]
  for (j in 1:nmonths) {
    index <- (i-1) * nmonths + j
    selection <-  iceyear[,"Month"] == months[j]
    iceMonthly[index] <- mean(iceyear[selection,"Extent"])
  }
}
#  Now we create a time series object
ice <- ts(iceMonthly, start=c(1979,1), frequency=12)
```

Conduct a seasonal trend analysis on `ice` using the built-in `stl(...)` function with parameters `s.window=11`, `s.degree=1`, and `t.degree=1`.

i. **(3 marks)** Plot the output of `stl` and describe your findings.

```
plot(stl(ice, s.window=11,
         s.degree=1, t.degree=1))
```



The long term trend of the data is cyclical - the short-term trends/cycles where the extent decreases and the increases again don't repeat regularly.

ii. **(2 marks)** Compare the output above with the decomposition of the irregular time series in part (a).

The shape of the seasonal portion of the diagram and randomness of the remainder is consistent with what was plotted in part (a) for the short term smooth of residuals and the plot of the remaining residuals respectively.

However, the main trend of the two questions are different, with the trend in part (a) having exhibited a stationary trend and the trend in part (b) being more cyclical. This is due to the irregularity of the data used in part (a), with part (b) showing the true trend of the data as the ice extents for the missing dates in part (a) have been accounted for.