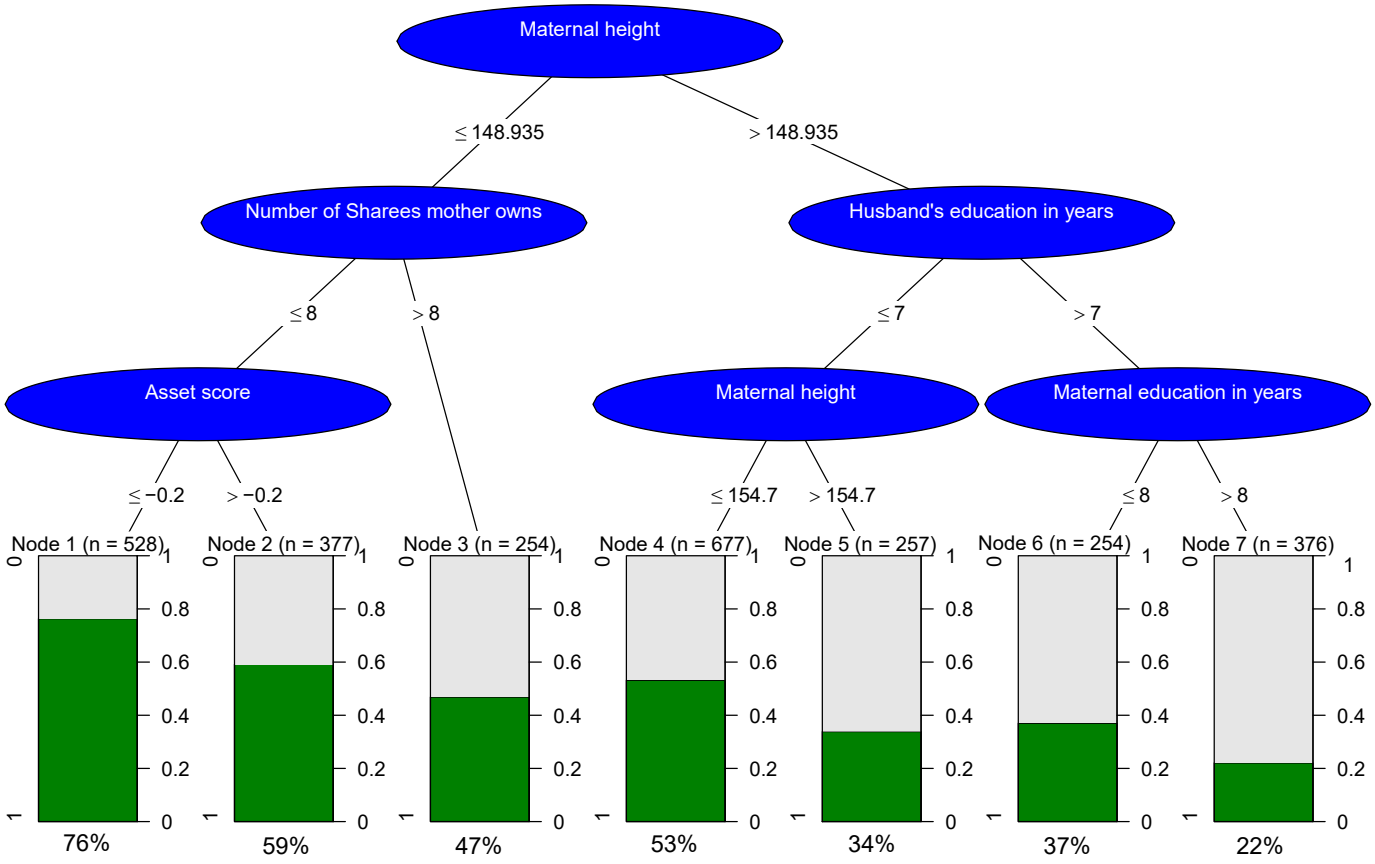
Visualization-Lab 1

Group 19 - fahha780 , vinbe289

Assignment 1



Tree

Imporving tree.pdf file using Inkscape resulted in the following tree which is much better to view and is of publication quality SVG file.

Assignment 2

1

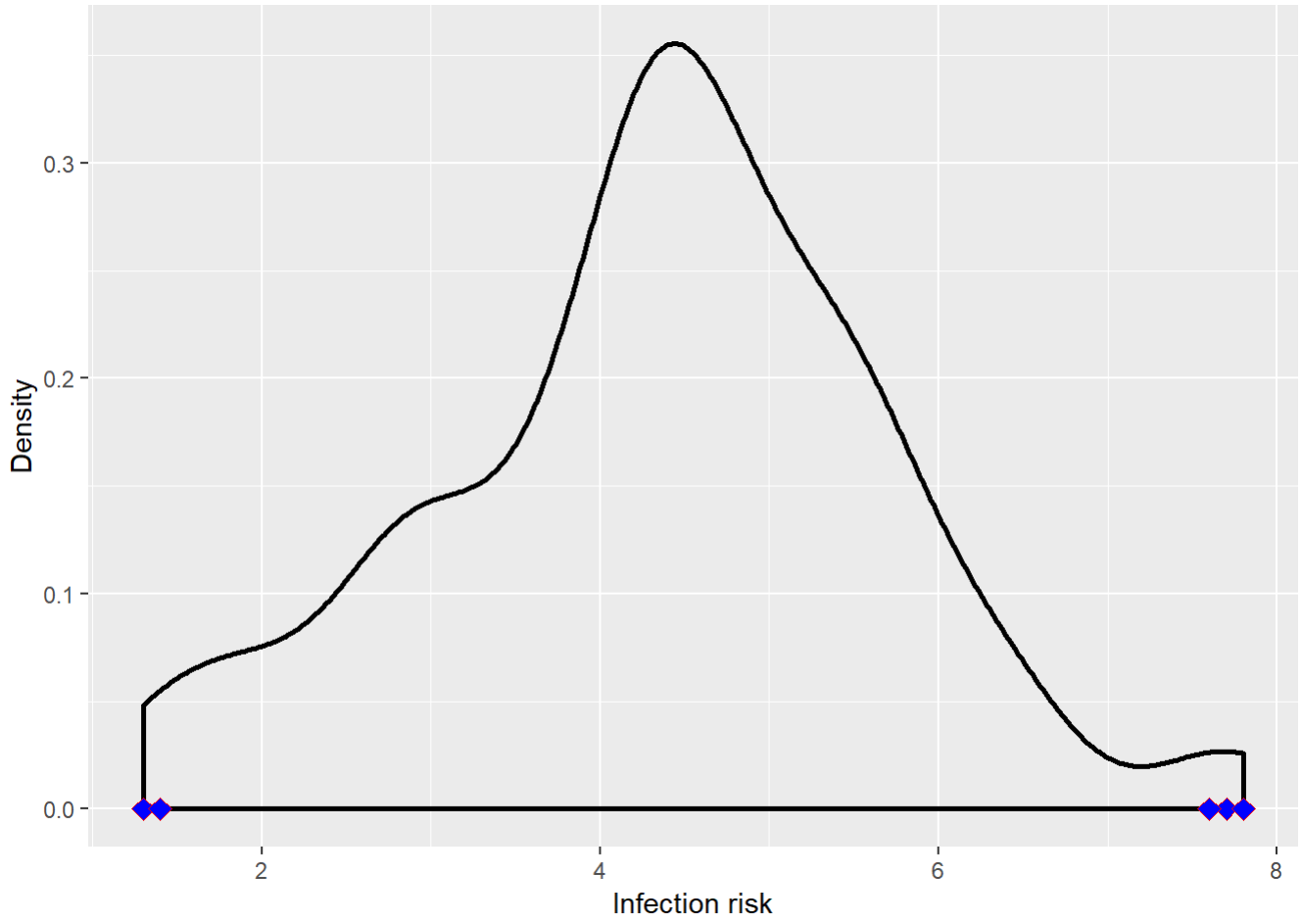
```
senic <- read.table(file="SENIC.txt")
```

2

```
col_Names <- c("ID", "Length of Stay", "Age", "Infection risk",
               "Routine Culturing Ratio", "Routine Chest X-ray Ratio", "Number of Beds",
               "Medical School Affiliation", "Region","Average Daily Census",
               "Number of Nurses", "Available Facilities & Services")

colnames(senic) <- col_Names

fun_quantile <- function(X){
  Q1 <- unname(quantile(X, 0.25))
  Q3 <- unname(quantile(X, 0.75))
  greater <- Q3 + 1.5*(Q3 - Q1)
  less <- Q1 - 1.5*(Q3 - Q1)
  index <- senic[(which(X > greater | X < less)),]$ID
  return(index)
}
```

3

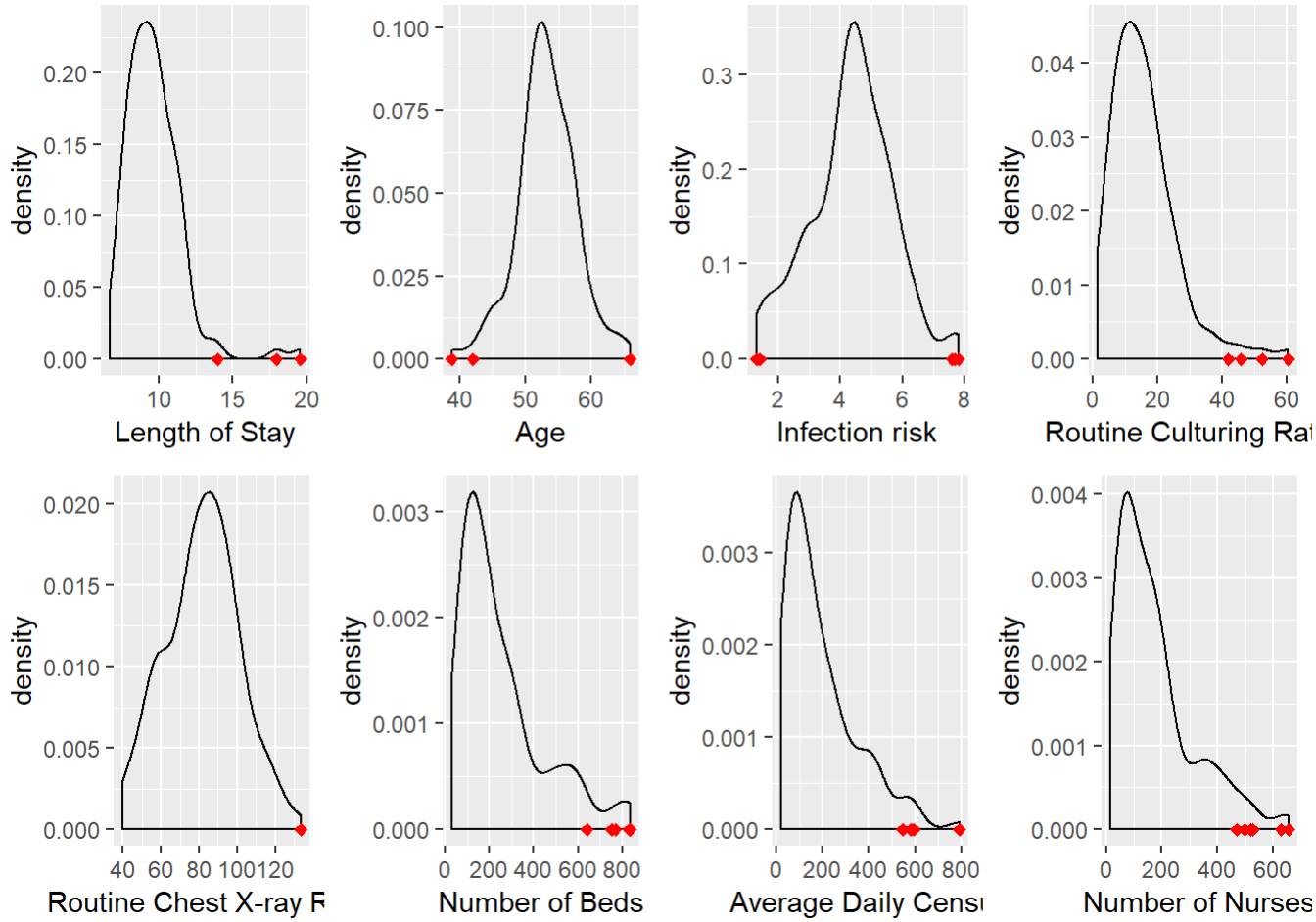
```
library(ggplot2)

Inf_risk <- fun_quantile(senic$`Infection risk`)
#extract Infection risk values corresponding to outliers index
value_Inf <- senic[Inf_risk,]$`Infection risk`
# 7.7 1.3 7.6 7.8 1.3 1.4
df_Infrisk <- as.data.frame(value_Inf)

Plot_1 <- ggplot() +
  geom_density(data = senic,aes(x=senic$`Infection risk`), size=1) +
  labs(x="Infection risk", y="Density") +
  geom_point(data = df_Infrisk, aes(x=value_Inf,y = 0, pch=3),size=3,shape=23,colour="red",fill=
"blue")
Plot_1
```



In the density graph of Infection risk the six outliers are plotted on the X-axis as diamond symbols. Outliers: 7.7 1.3 7.6 7.8 1.3 1.4 Majority of the values seem to lie in the region between 4 and 5.

4

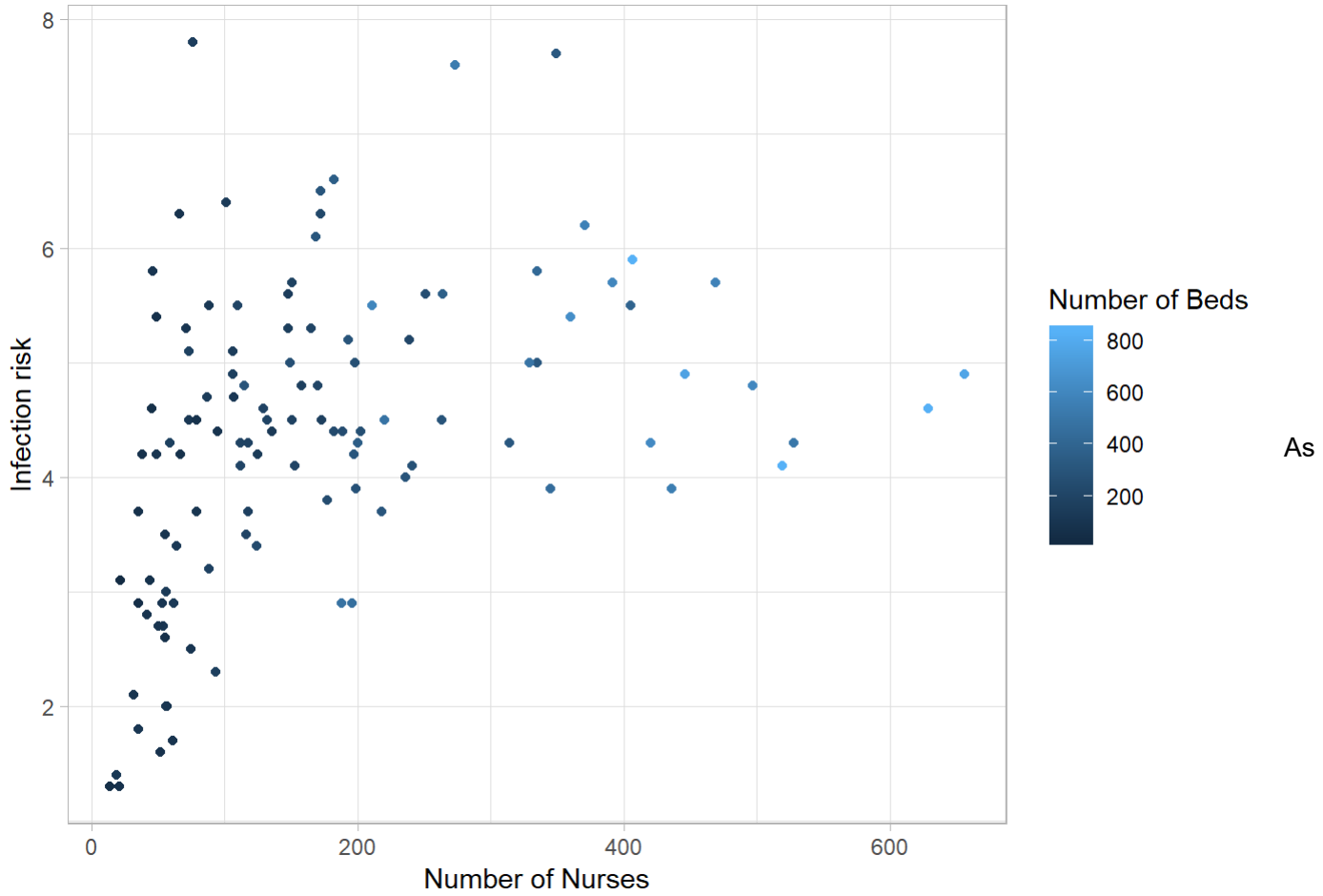
```
library(gridExtra)
combined_plots <- lapply(c(2:7,10,11), function(i){
  ggplot() +
    geom_density(aes(senic[,i])) +
    geom_point(aes(x = senic[fun_quantile(senic[,i]),i], y = 0), shape = 23, col = "red", fill =
 "red") +
    labs(x = col_Names[i])
})

Plot_Final <- arrangeGrob(grobs = combined_plots, nrow=2)
plot(Plot_Final)
```



The density plots for all quantitative variables are plotted in a similar manner. The outliers are again plotted on the X-axis in Diamond symbols. The outliers for Density graph of Length of stay, Routine cultring ratio, Routine chest x-ray ratio, Number of beds, Average daiy census and Number of nurses seem to lie only in the right portion of the graph. The outliers for Density graph of Infection risk, Age, seem to lie on either sides of the graph. There are no outliers in the variable Available Facilities and services. The Density graph for Length of stay, Routine Culturing ratio, Number of beds, Average daily census, Number of nurses is right scewed.

5

```
ggplot(data = senic, aes(x = senic$`Number of Nurses`, y = senic$`Infection risk`,
                         color = senic$`Number of Beds`))+
  labs(x="Number of Nurses", y="Infection risk", colour="Number of Beds")+
  geom_point()+
  theme_light()
```



the plot includes 3 variables Infection risk, Number of nurses and Number of beds we can see the dependence of these variables on each other which was not available in the density graph for each of these variables. The Infection risk seem to increase with the increase in the number of beds while the Number of Nurses stay almost constant.But when there is an increase in the Number of nurses along with the increase in Numer of beds the Infection risk reduces signifcantly. The Infetion risk is lowestwhen both Number of beds and nurses is low. Danger- The minute difference in colour saturation cannot be easily distinguished. If any ouliers exist, they cannot be differentiated easily compared to when distinct colours are used.

6

```
library(plotly)
```

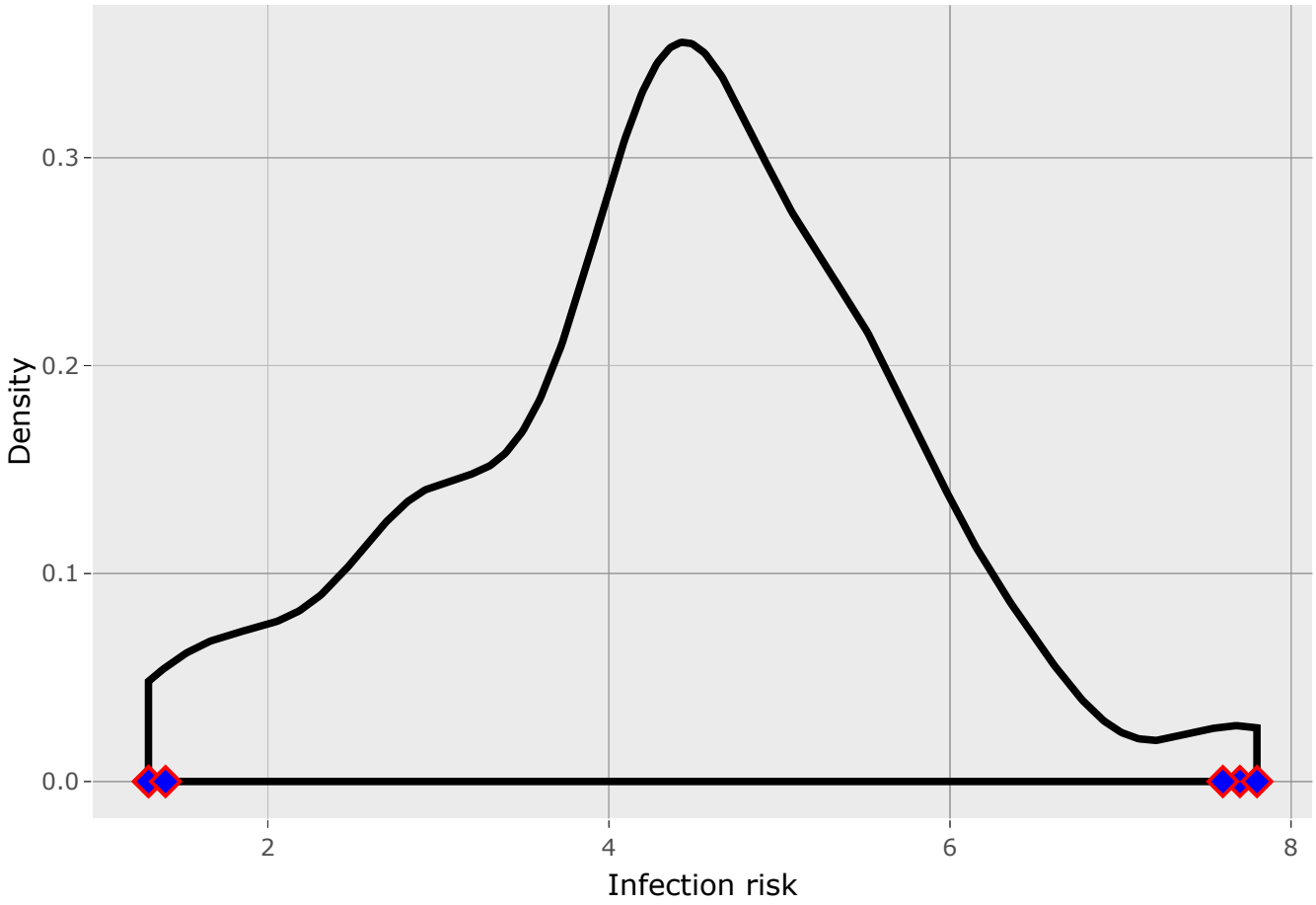
```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(later)
ggplotly(Plot_1)
```



We get an interactive graph wherein the cursor can be hovered on the plot to get info such as the x-axis and y-axis variable values at any particular point on the plot. The value of outliers can easily be found out by just hovering the cursor on the outliers. The approximate value of mean can be checked instantly.

7

```
library(dplyr)
```

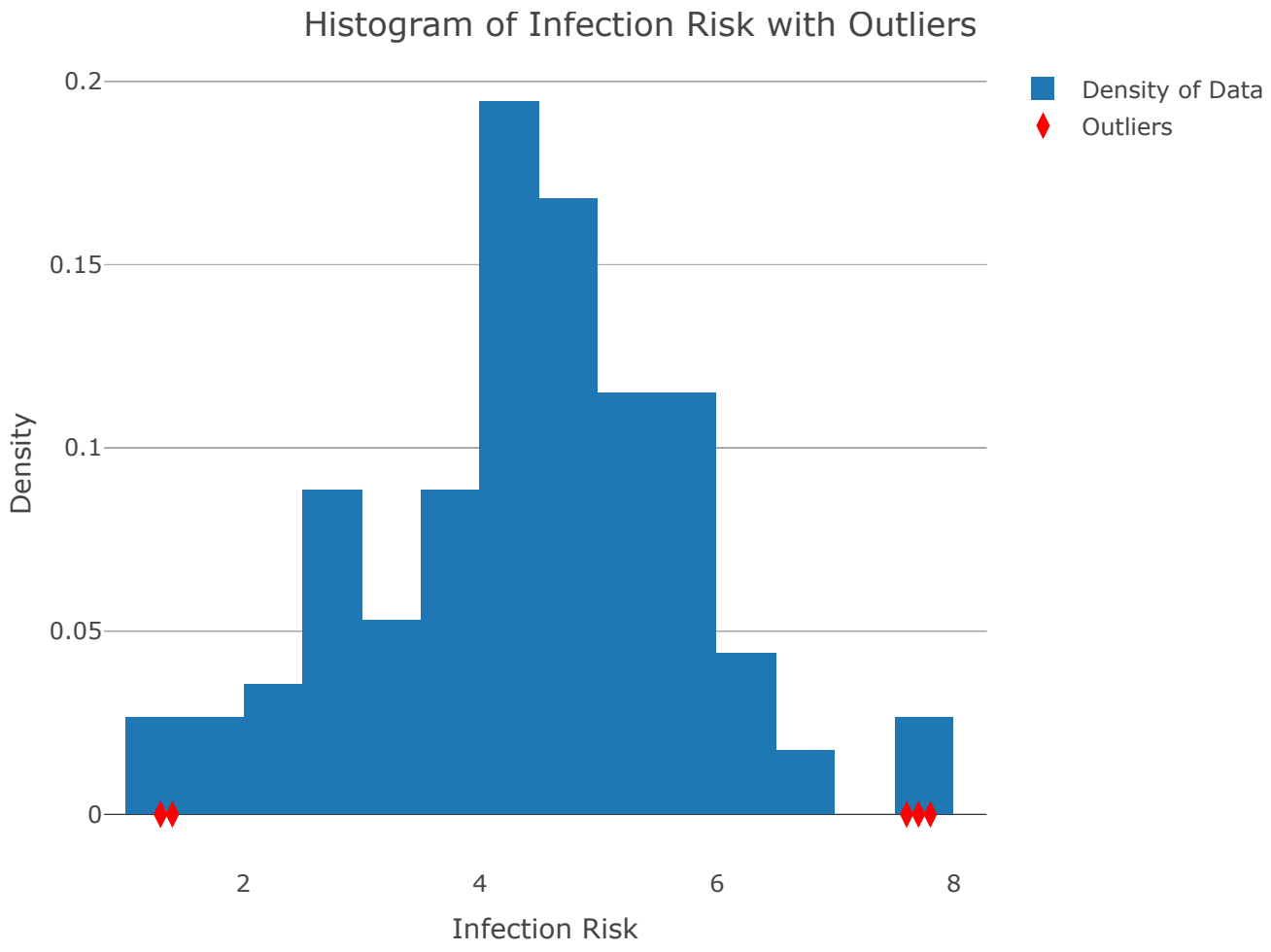
```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:gridExtra':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
riskIndex <- fun_quantile(senic$`Infection risk`)
outliersRisk <- senic[riskIndex,]

plt <- senic %>% select('Infection risk') %>%
  mutate(Rank=ntile(senic$`Infection risk`,3)) %>%
  plot_ly(x=senic$`Infection risk`, type = "histogram",
          histnorm = "probability",
                name = "Density of Data") %>%
  add_markers(x = outliersRisk$'Infection risk' , y = 0,
              marker = list(size =10 ,symbol = 23,color = "red"),
              name = "Outliers") %>%
  layout(title = "Histogram of Infection Risk with Outliers",
         xaxis = list(title = "Infection Risk"),
         yaxis = list(title = "Density"))
plt
```



8

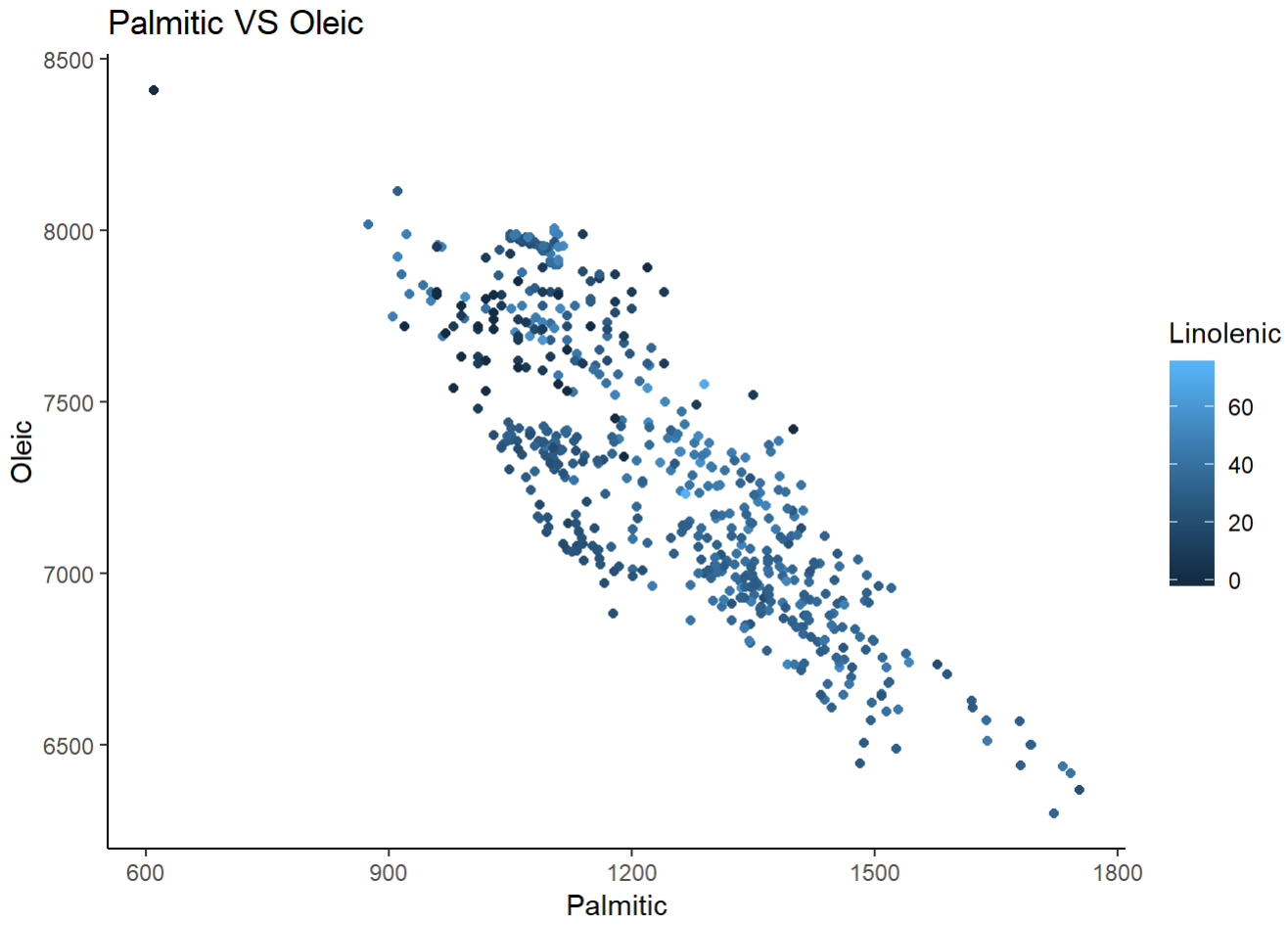
```
senic <- read.table("SENIC.txt")

colnames(senic) <- c("ID","STAY","AGE","RISK","CULTURE_RATIO","CHEST","BEDS","AFFILIATION","REGI
ON","CENSUS","NURSES","FS")

fun_quantile <- function(X){
  Q1 <- unname(quantile(X, 0.25))
  Q3 <- unname(quantile(X, 0.75))
  greater <- Q3 + 1.5*(Q3 - Q1)
  less <- Q1 - 1.5*(Q3 - Q1)
  index <- senic[(which(X > greater | X < less)),]$ID
  return(index)
}

Select_Variables <- c("STAY","AGE","RISK","CULTURE_RATIO","CHEST","BEDS","CENSUS","NURSES","FS")

ui <- fluidPage(
  sliderInput(inputId="bw_value", label="Choose bandwidth size", value=0.1, min=0.01, max=1),
  checkboxGroupInput("var", "Variables ", Select_Variables , inline=TRUE,selected = "STAY"),
  plotOutput("densPlot")
)

server <- function(input, output) {
  output$densPlot <- renderPlot({
    sele <- input$var
    lst <- vector("list" , length = length(sele))

    for (i  in 1:length(sele)) {
      vals <- fun_quantile(senic[,sele[i]])
      senic2 <- senic[vals,]
      lst[[i]] <- ggplot() + geom_density(data = senic, aes_string(x = sele[i] ), size=1,bw = in
put$bw_value) +
        geom_point(data=senic2, aes_string(x = sele[i],y=0, pch=3),size=3,shape=23,colour="red",
fill="blue") +
       scale_shape_identity()
    }
    final <- arrangeGrob(grobs = lst)
    grid.arrange(final)
  })
}

shinyApp(ui = ui, server = server)
```

The plot seems to get cluttered at low bandwidth and we can view every point in the graph which is not the proper way to see density of a variable. When we increase the bandwidth the graph becomes more smooth and shows us general view of the variable. The chosen bandwidth is 0.01 to 1 with increase of 0.01. By analyzing the graph we found at 0.38 bandwidth the graph shows the best view of data density.

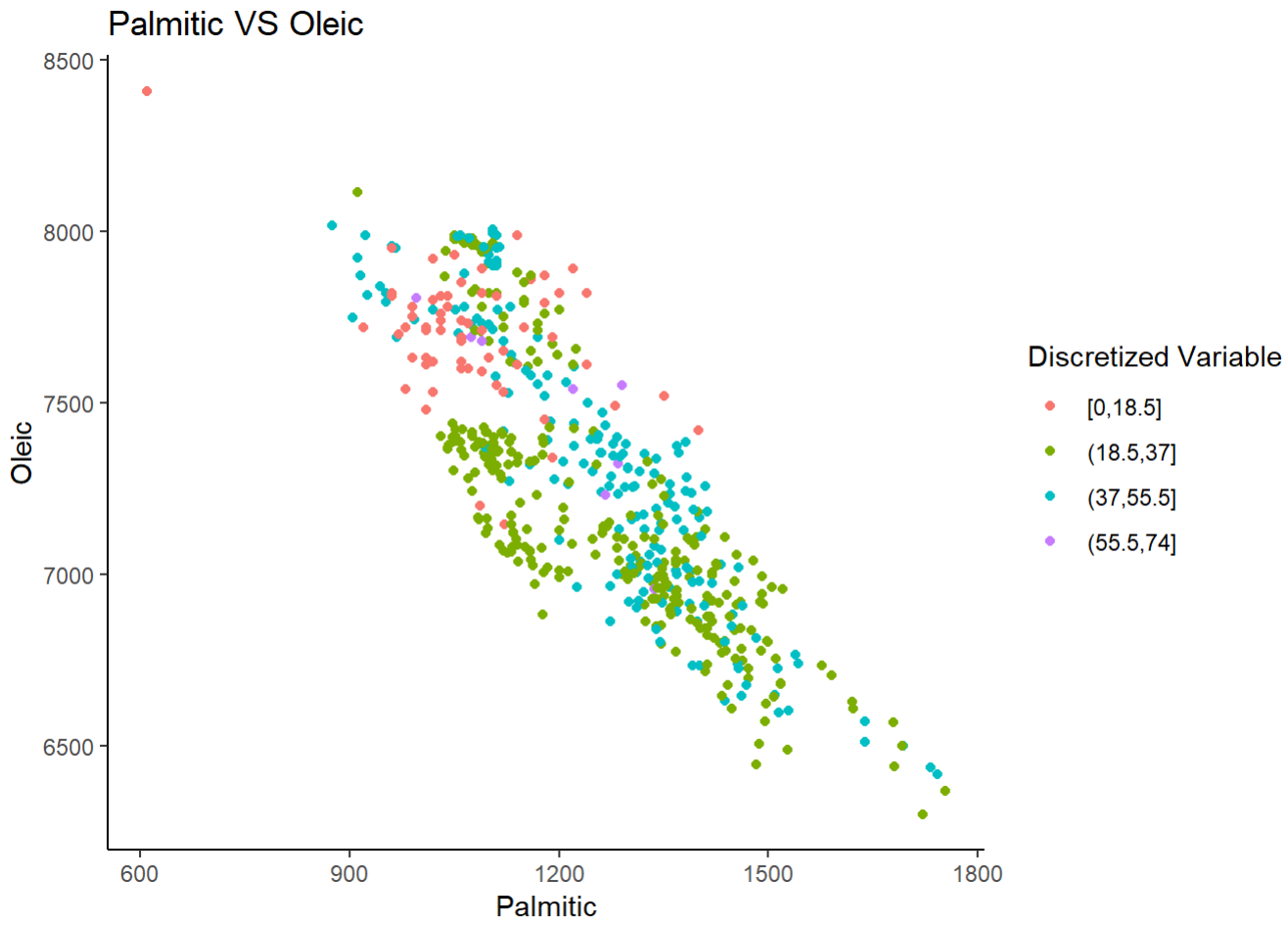
Lab 2

Group 19

```
library(ggplot2)
olive <- read.csv("olive.csv")

## 1 Palmitic on Oleic
ggplot(olive, aes(x=palmitic, y=oleic, col=linolenic)) + geom_point()  +
  labs(x="Palmitic", y="Oleic", colour="Linolenic") +
  theme_classic() + ggtitle("Palmitic VS Oleic")
```

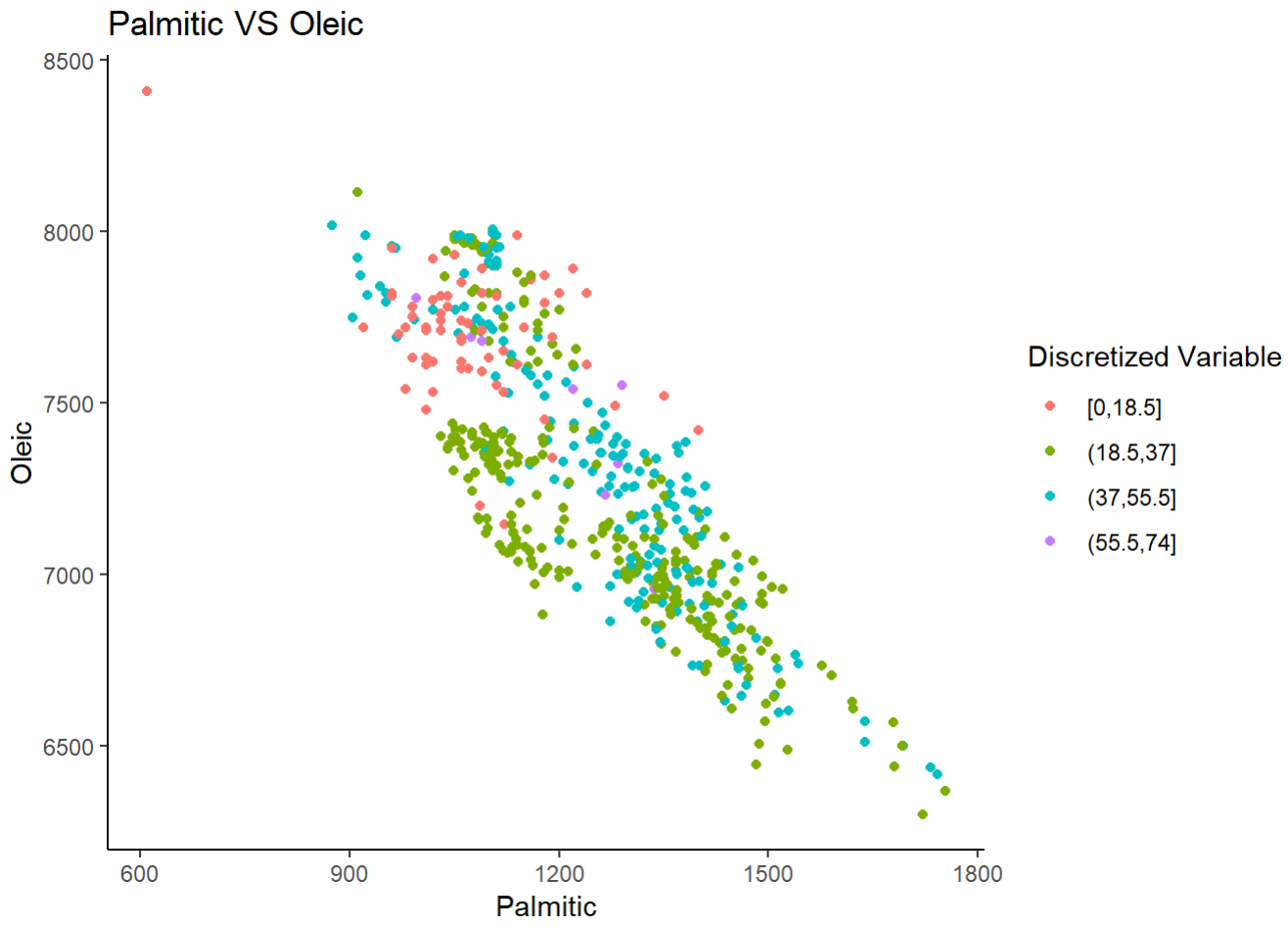



```
X_cut <- cut_interval(x=olive$linolenic, n = 4)

ggplot(olive, aes(x=palmitic, y=oleic, col=X_cut)) + geom_point()  +
  labs(x="Palmitic", y="Oleic", colour="Discretized Variable") +
  theme_classic() + ggtitle("Palmitic VS Oleic")
```



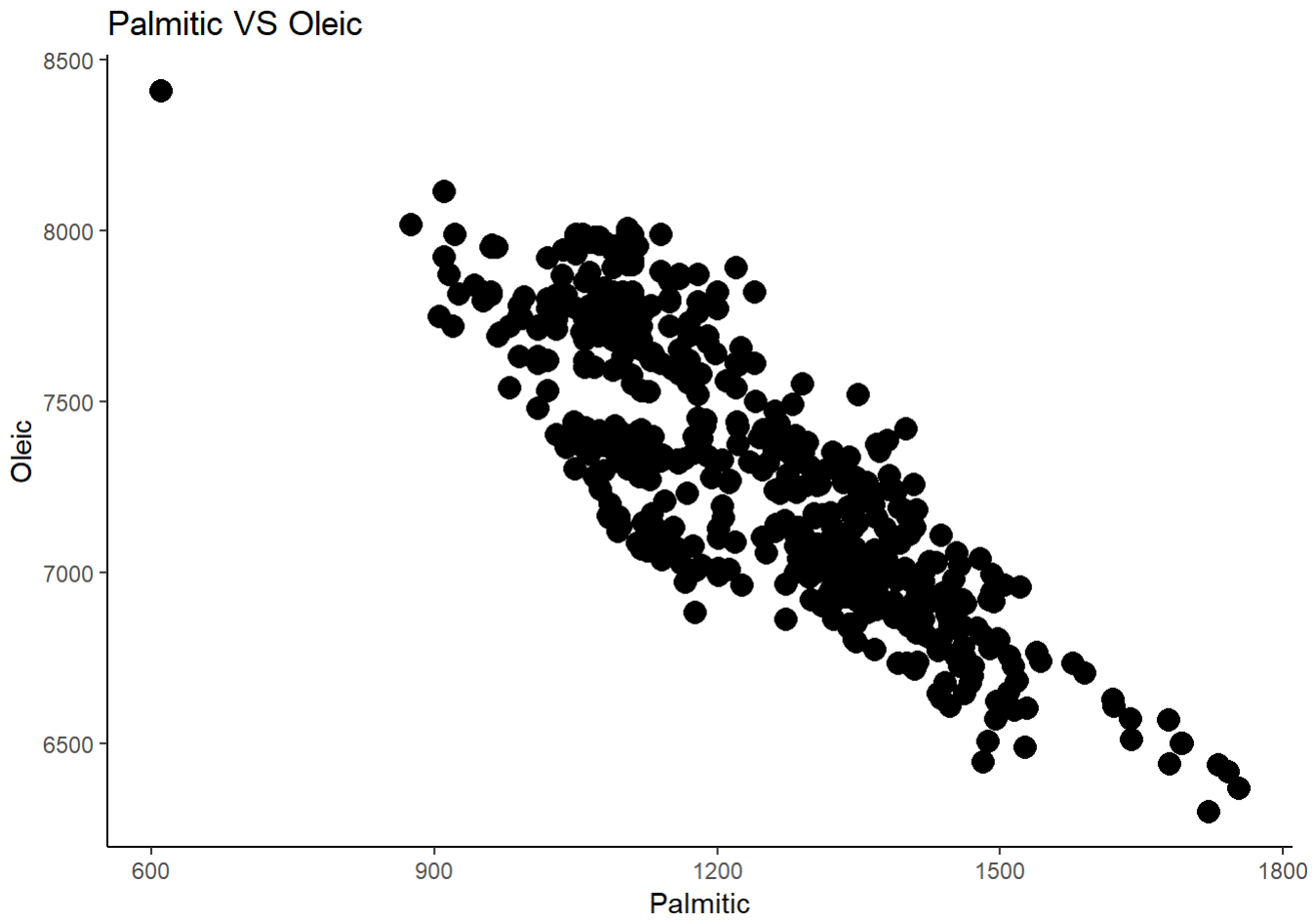
In the first graph, it is hard to catogorize the data points for Linolenic since it is hard to differntiate between data points whose values are colse to each other. The second plot is relatively easier to analyze but when the data is catogorized using distinct colours the data points with lighter coulour may not be noticable when a cluster of data points with dark colour overlaps it.

```
## 2 Palmitic vs Oleic
### a  Color
ggplot(olive, aes(x=palmitic, y=oleic,col=X_cut)) + geom_point()  +
  labs(x="Palmitic", y="Oleic", colour="Discretized Variable") +
  theme_classic() + ggtitle("Palmitic VS Oleic")
```

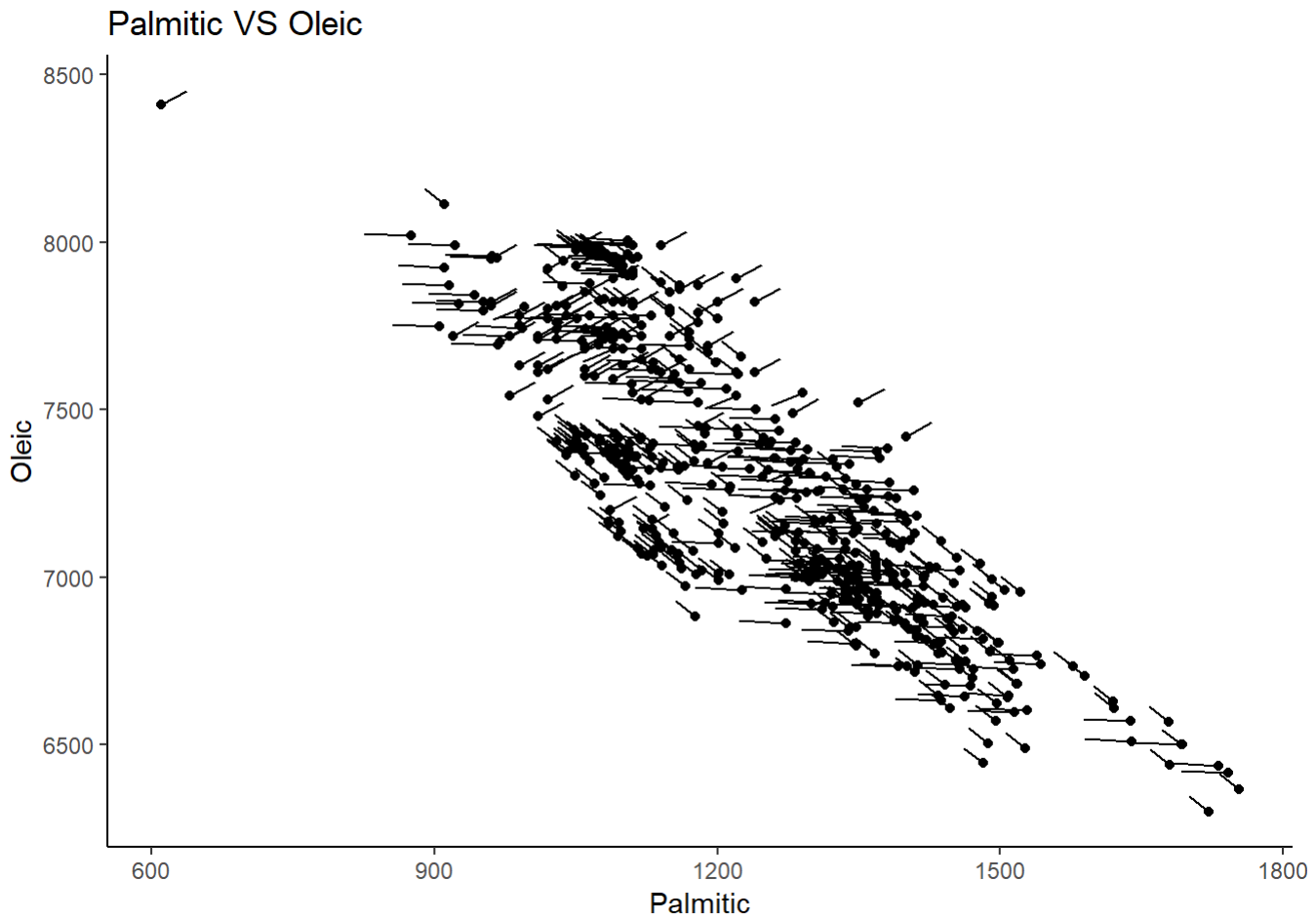



```
### b  Size
ggplot(olive, aes(x=palmitic, y=oleic)) + geom_point(size=X_cut)  +
  labs(x="Palmitic", y="Oleic", colour="Discretized Variable") +
  theme_classic() + ggtitle("Palmitic VS Oleic")
```

```
## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors
```

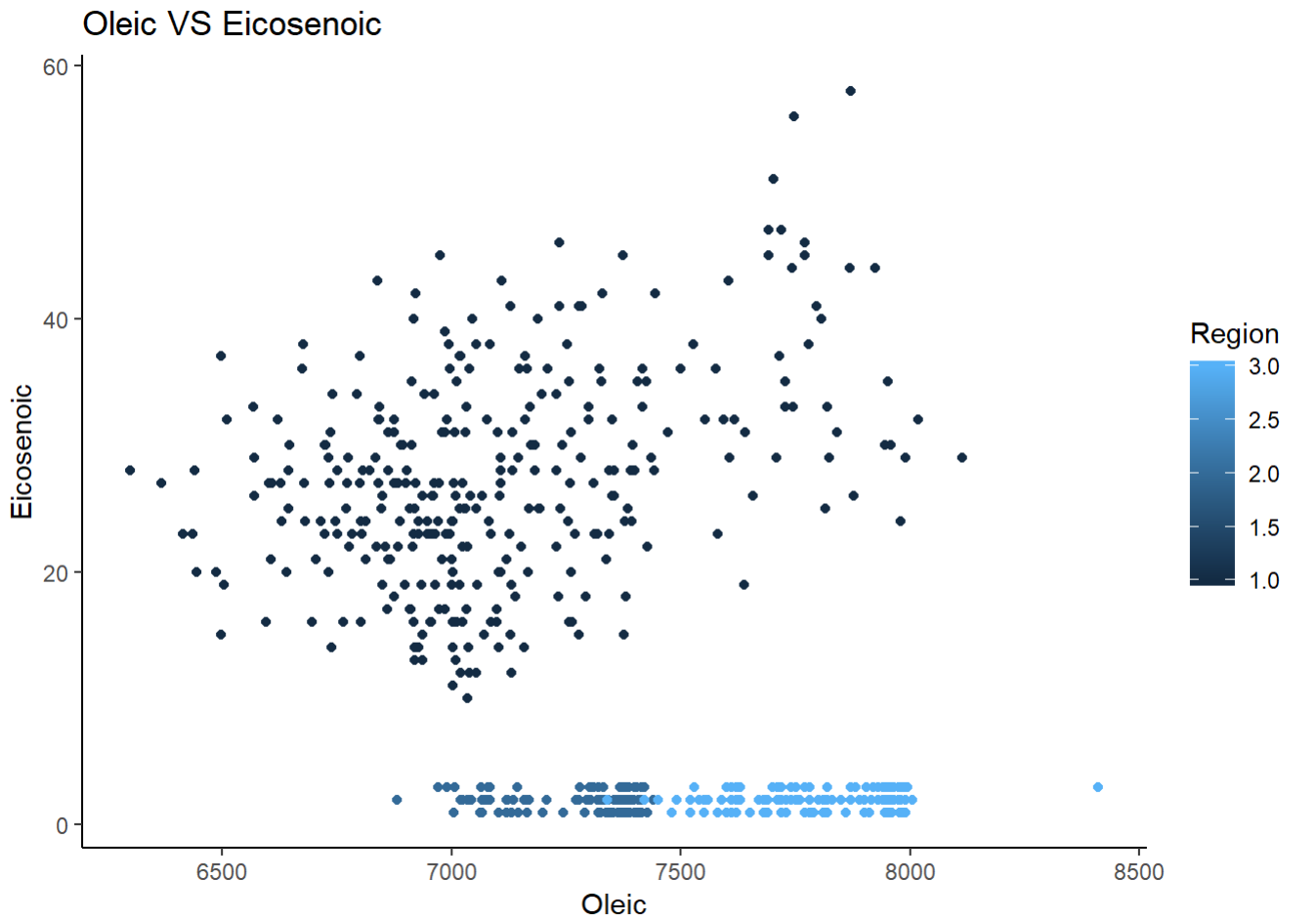



```
### c  Orientation Angle
ggplot(olive, aes(x=palmitic, y=oleic)) + geom_point()  +
  geom_spoke(aes(angle = as.numeric(X_cut)), radius = 50) +
  labs(x="Palmitic", y="Oleic", colour="Discretized Variable") +
  theme_classic() + ggtitle("Palmitic VS Oleic")
```

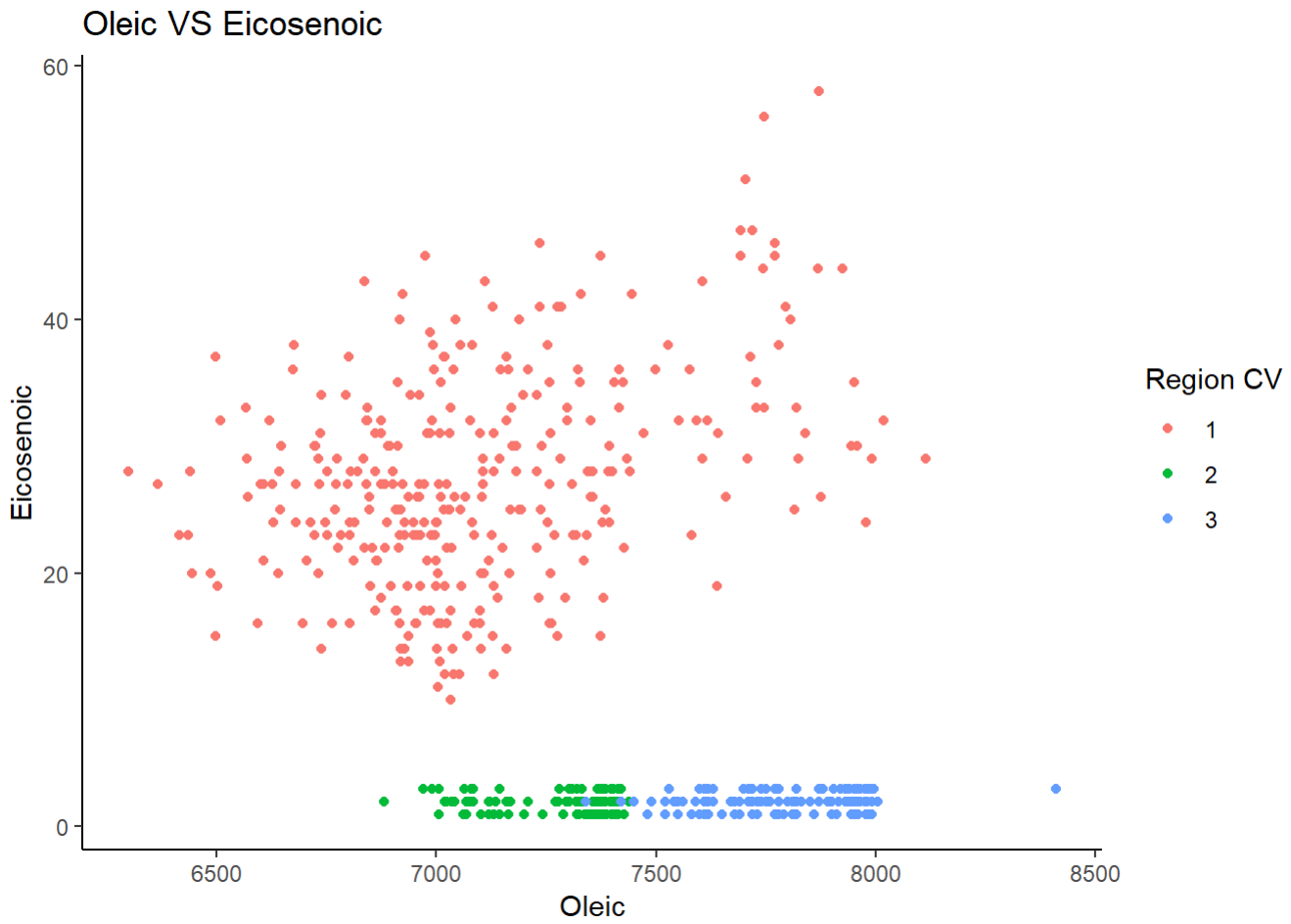


Second plot is the hardest plot to analyze since only 4-5 levels can be percieved(2.2 bits) which is less compared to that of colour hue(3.1 bits) and line orientation(3 bits). Also, the data points overlap each other heavily and the data points of the third variable cannot be catogorized visually. First plot is the easiest of all to analyze since we can percieve 8 levels in colour hue(3 bits) and the hue is fairly distinct.

```
## 3 Oleic vs Eicosenoic
ggplot(olive, aes(x=oleic, y=eicosenoic,col=Region)) + geom_point()  +
  labs(x="Oleic", y="Eicosenoic", colour="Region") +
  theme_classic() + ggtitle("Oleic VS Eicosenoic")
```



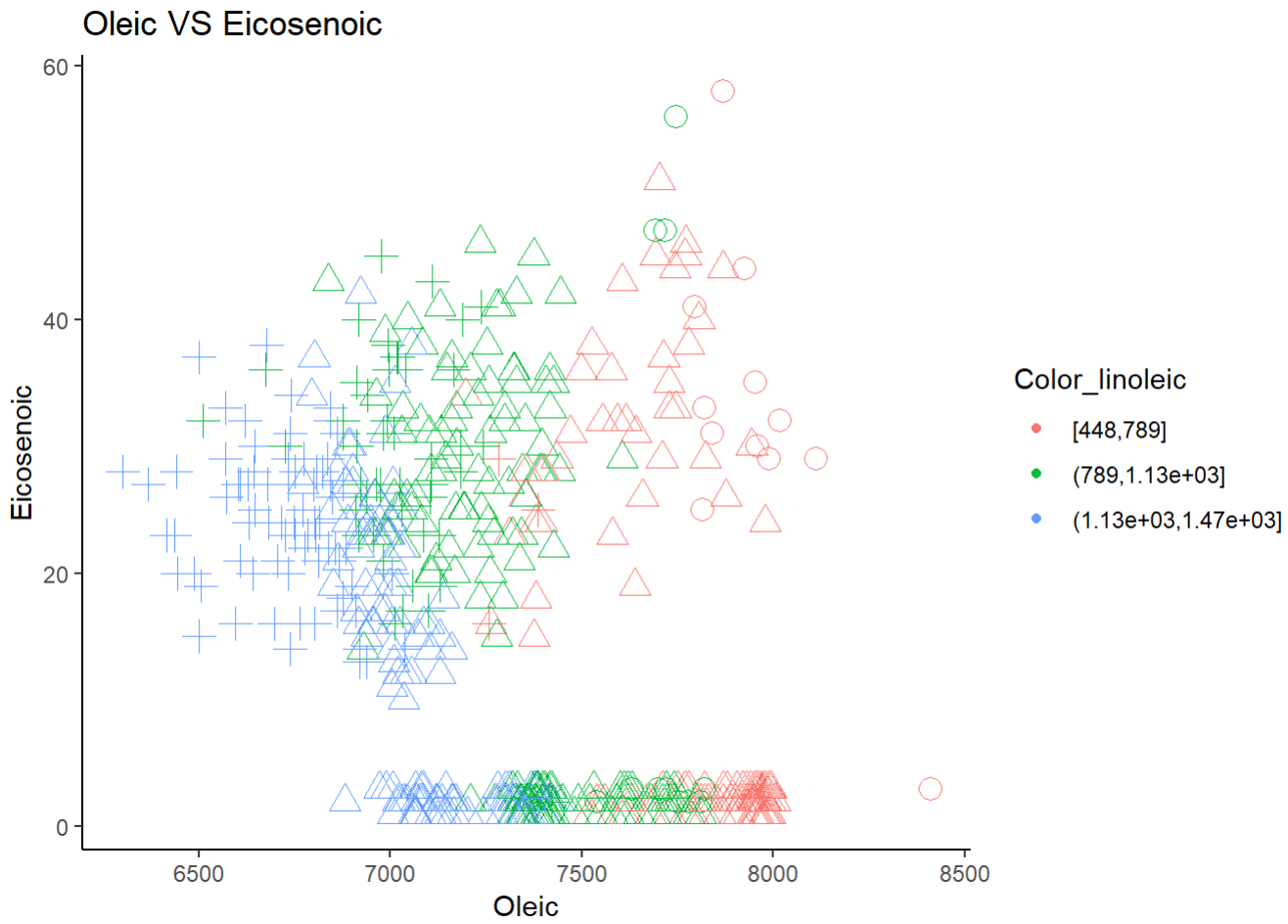

```
Region_categorical <- as.factor(olive$Region)
ggplot(olive, aes(x=oleic, y=eicosenoic,col=Region_categorical)) + geom_point()  +
  labs(x="Oleic", y="Eicosenoic", colour="Region CV") +
  theme_classic() + ggtitle("Oleic VS Eicosenoic")
```



In the first plot since region is a catogorical variable i.e. discrete, using this type of colouring is not advisable(differing brightness) because when there are larger number of regions it would make catogorizing regions visually with respect to region hard. This type of colouring can only be used for continuous variables. It is easier to catogorize visually with respect to region in the second plot due to preattentive mechanism.

```
## 4 Oleic vs Eicosenoic
Color_linoleic <- cut_interval(x=olive$linoleic, n = 3)
shape_palmitic <- cut_interval(x=olive$palmitic, n = 3)
size_palmitoleic <- cut_interval(x=olive$palmitoleic, n = 3)

ggplot(olive, aes(x=oleic, y=eicosenoic,col=Color_linoleic)) +
  geom_point(shape = shape_palmitic,size=size_palmitoleic)  +
  labs(x="Oleic", y="Eicosenoic", colour="Color_linoleic") +
  theme_classic() + ggtitle("Oleic VS Eicosenoic")
```

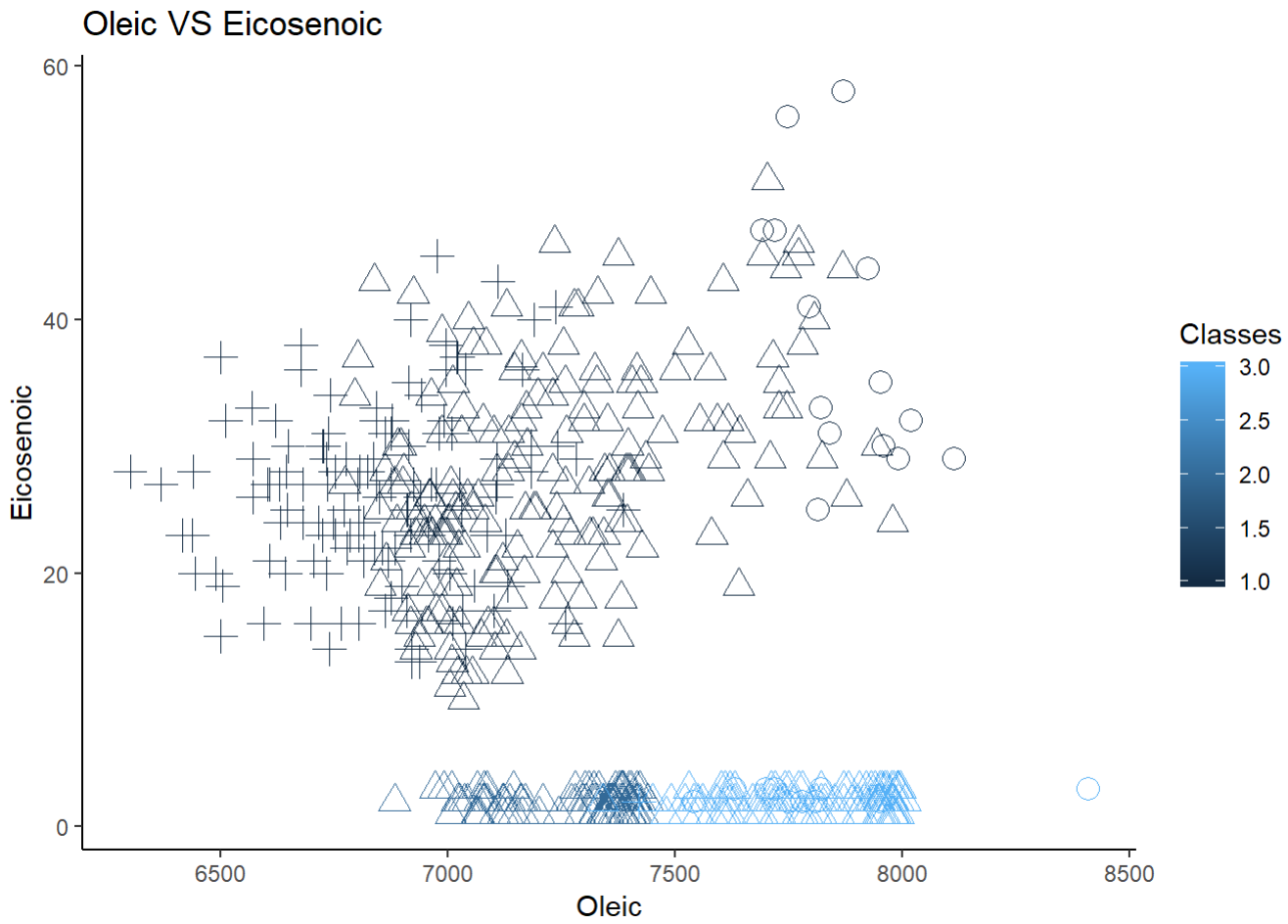
```
## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors
```



It is quite difficult to analize this type of plot since many variables are plotted using different metrics. It is also difficult to analyze since visual analysis does not become any easier by combining different metrics, in this case 3*3 i.e. 27 and preattentive mechanism does not make sense. The channel capacities does not sum up in this case.

```
## 5 Oleic vs Eicosenoic
ggplot(olive, aes(x=oleic, y=eicosenoic,col=Region)) +
  geom_point(shape = shape_palmitic,size=size_palmitoleic)  +
  labs(x="Oleic", y="Eicosenoic", colour="Classes") +
  theme_classic() + ggtitle("Oleic VS Eicosenoic")
```

```
## Warning in Ops.factor(coords$size, .pt): '*' not meaningful for factors
```



It is possible to clearly see a decision boundary between Regions despite using many aesthetics because only 3 regions exist and the colours are distinct due to this. The peattentive mechanism comes into picture as well. This figure is processed in parallel by checking colour, shape (and size to a little extent) individually which acts as individual feature maps. Specific preattentive task is performed on each of these feature maps which aids in quicker and easier visual analysis.

```
## 6
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(plotly)
```

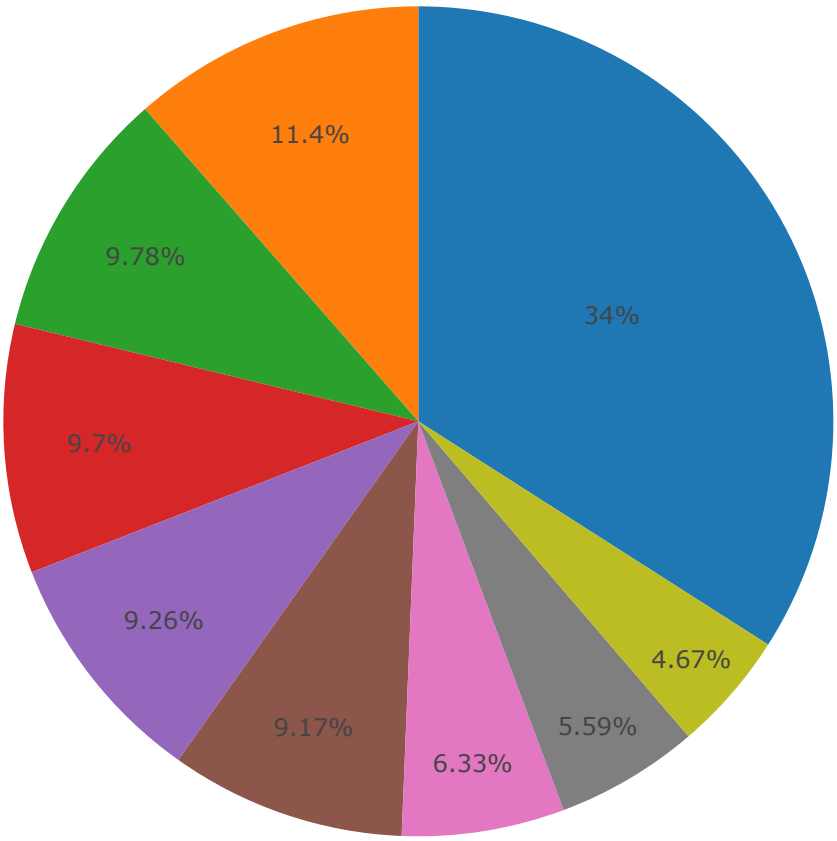
```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

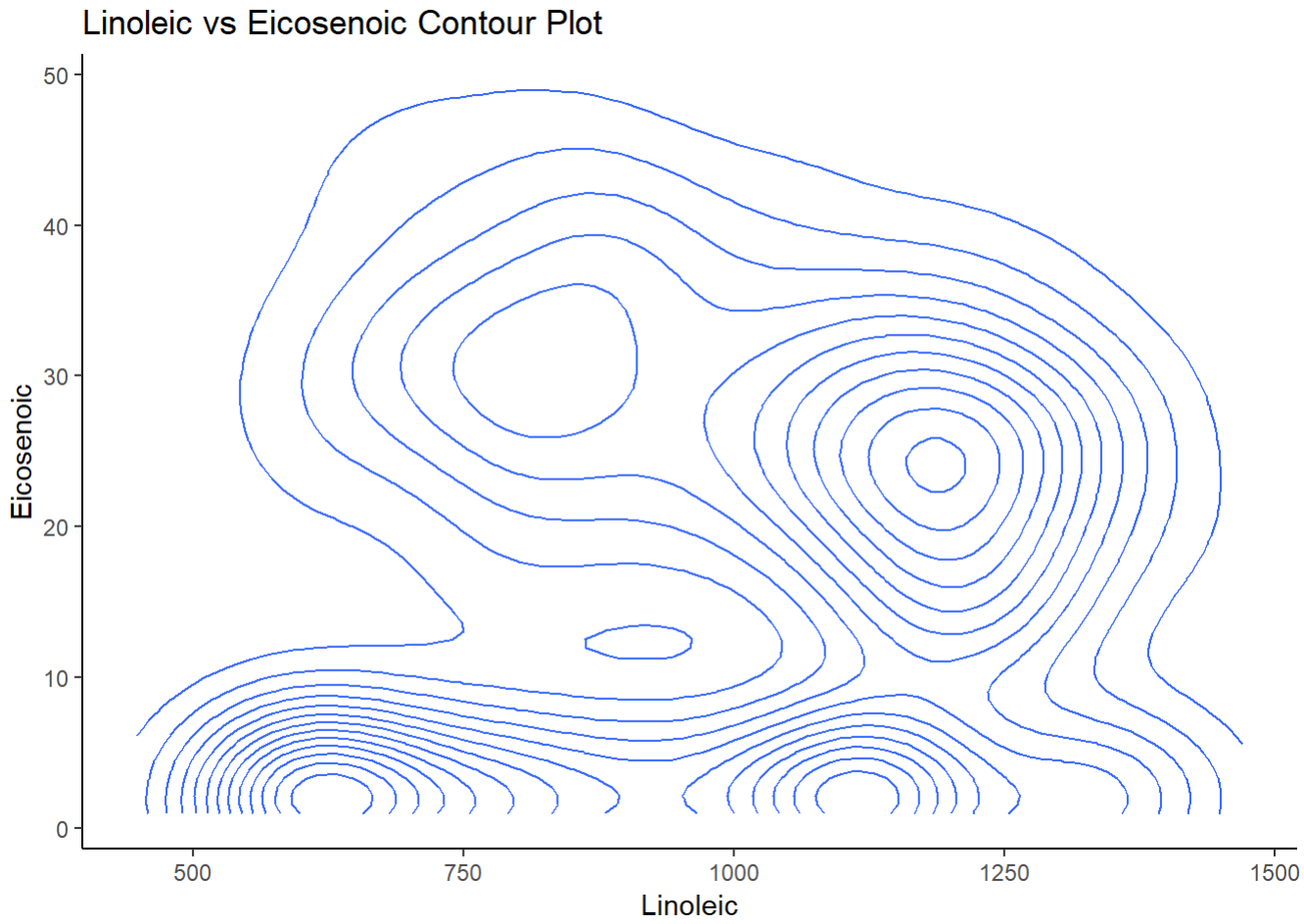
```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
PP<- plot_ly(olive, labels = ~Area, values = ~oleic, type = 'pie') %>%
                          layout(showlegend = FALSE)
PP
```

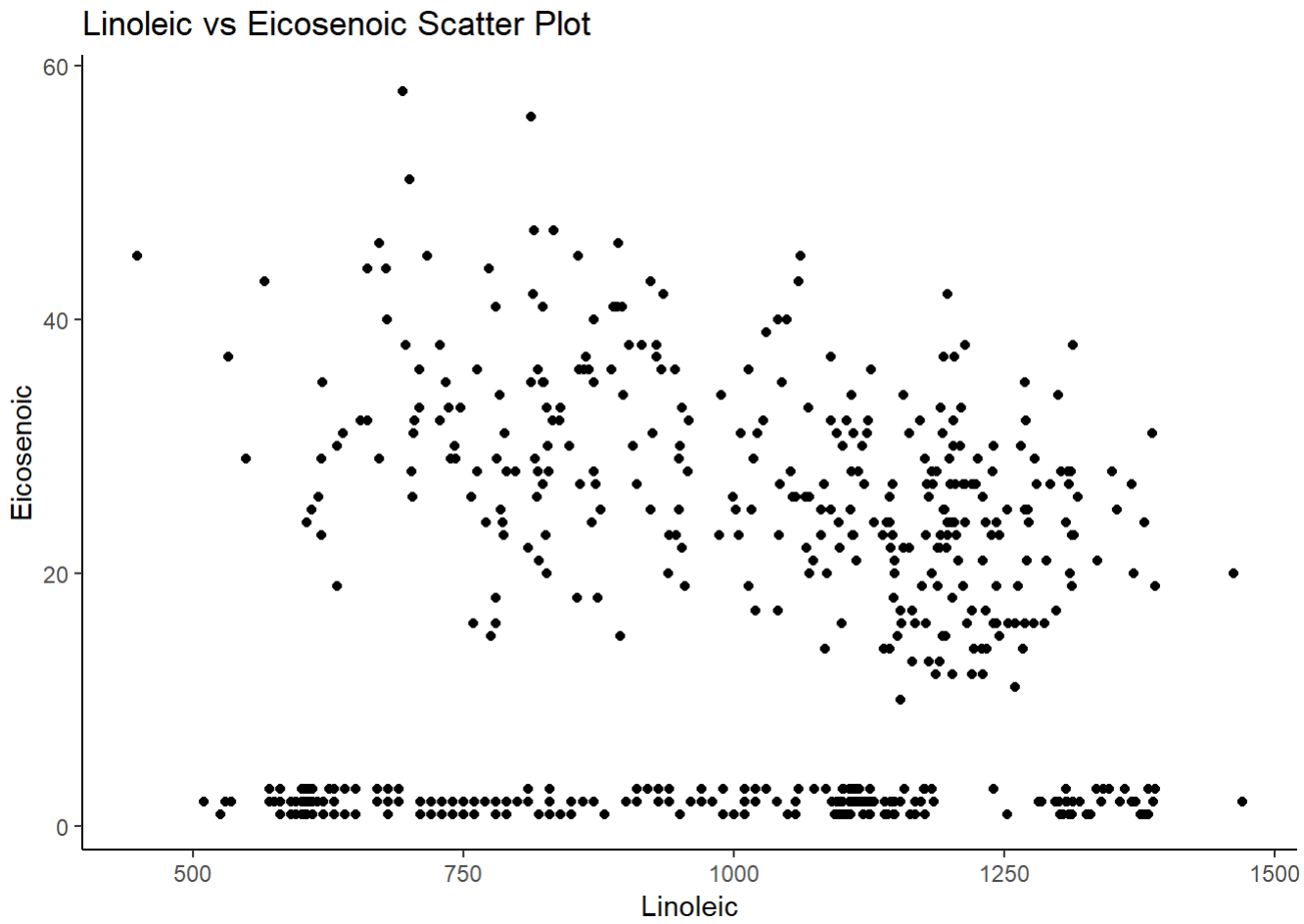


Since lables are removed, the name of a particular area will only be visible when the cursor is hovered on that particular area on the pie chart and other area names will not be visible.

```
## 7 Linoleic vs Eicosenoic
ggplot(olive, aes(x=linoleic, y=eicosenoic) ) +
  geom_density_2d() +
  labs(x="Linoleic", y="Eicosenoic") +
  theme_classic() + ggtitle("Linoleic vs Eicosenoic Contour Plot")
```




```
ggplot(olive, aes(x=linoleic, y=eicosenoic)) +
  geom_point()  +
  labs(x="Linoleic", y="Eicosenoic", colour="Classes") +
  theme_classic() + ggtitle("Linoleic vs Eicosenoic Scatter Plot")
```



This contour plot can be misleading since the data points are not discrete or distinct. In addition the colour scale does not make the visual analysis any easier.

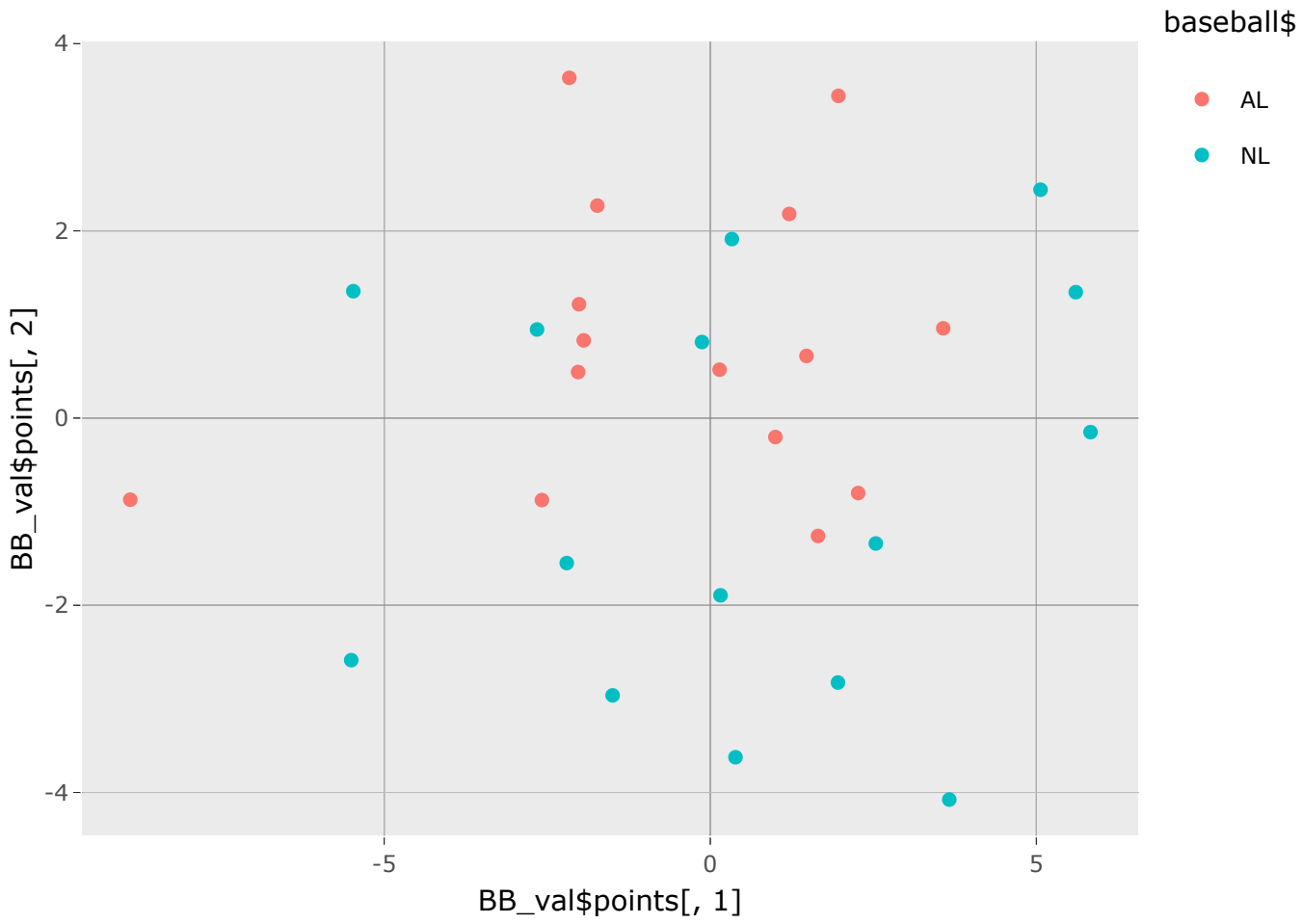
```
library(xlsx)
library(MASS)
library(plotly)
library(ggpubr)

## 1
baseball <- read.xlsx("baseball-2016.xlsx", sheetIndex = 1)
baseball.df <- as.data.frame(baseball)

baseball.numeric <- scale(baseball[,sapply(baseball,is.numeric)])
dist_baseball <- dist(baseball.numeric,"minkowski")
```

The numeric variables in the dataset baseball have different scales. When we analyze the numeric data in terms of their mean and variance they differ too much so it is necessary to scale the numeric varibles to perform Multi dimensional scalling to get proper plot.

```
## 2 Non MDS
BB_val <- isoMDS(dist_baseball, k=2, p=2)
```

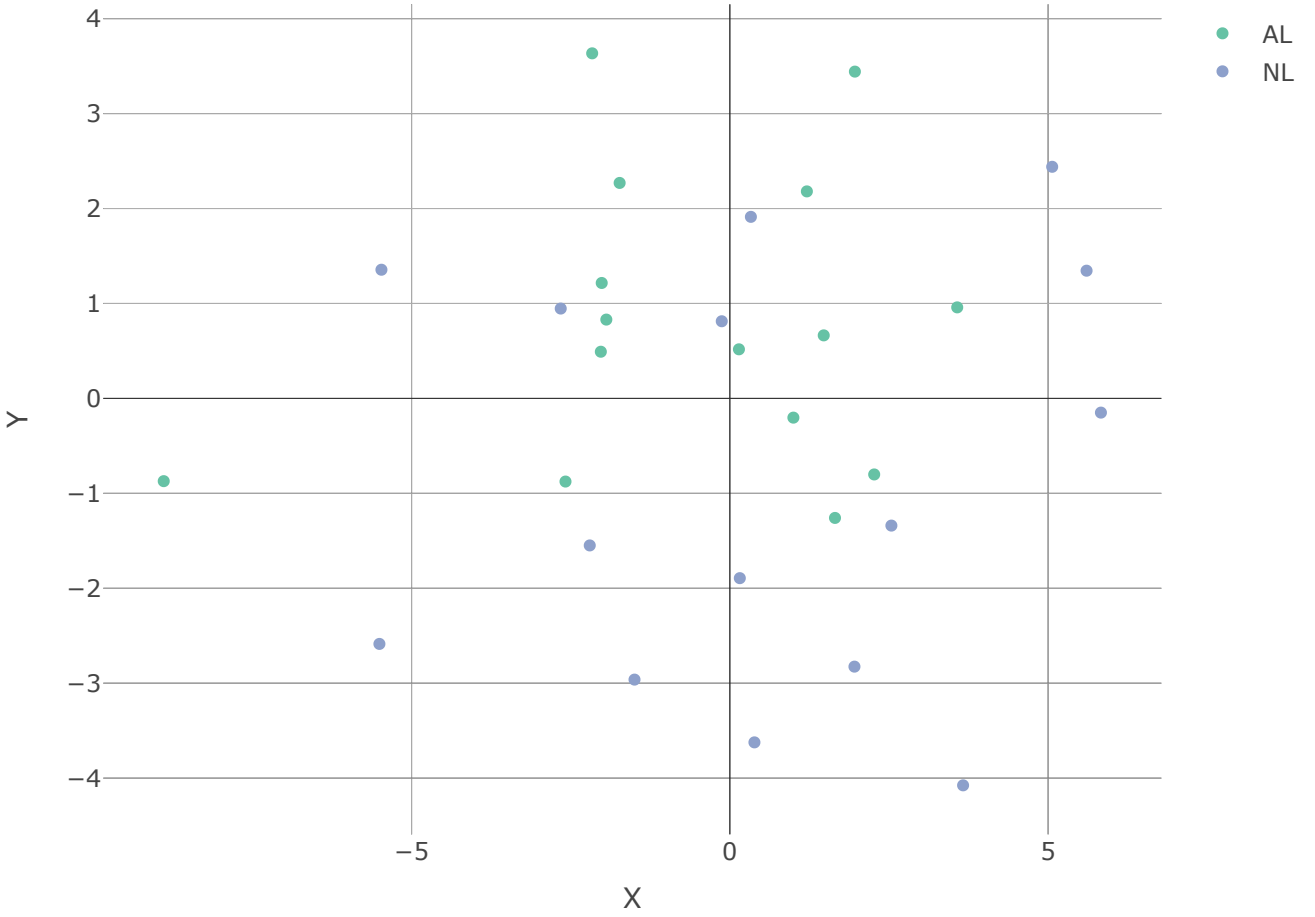
```
## initial  value 19.856833
## iter   5 value 16.319153
## iter  10 value 16.046215
## final  value 15.935476
## converged
```

```
BB_val_df <- data.frame(BB_val)
coords <- BB_val$points

NOMSD_plot <- ggplot(BB_val_df, aes(x=BB_val$points[,1], y=BB_val$points[,2])) + geom_point(aes
(colour = baseball$League))
ggplotly(NOMSD_plot)
```



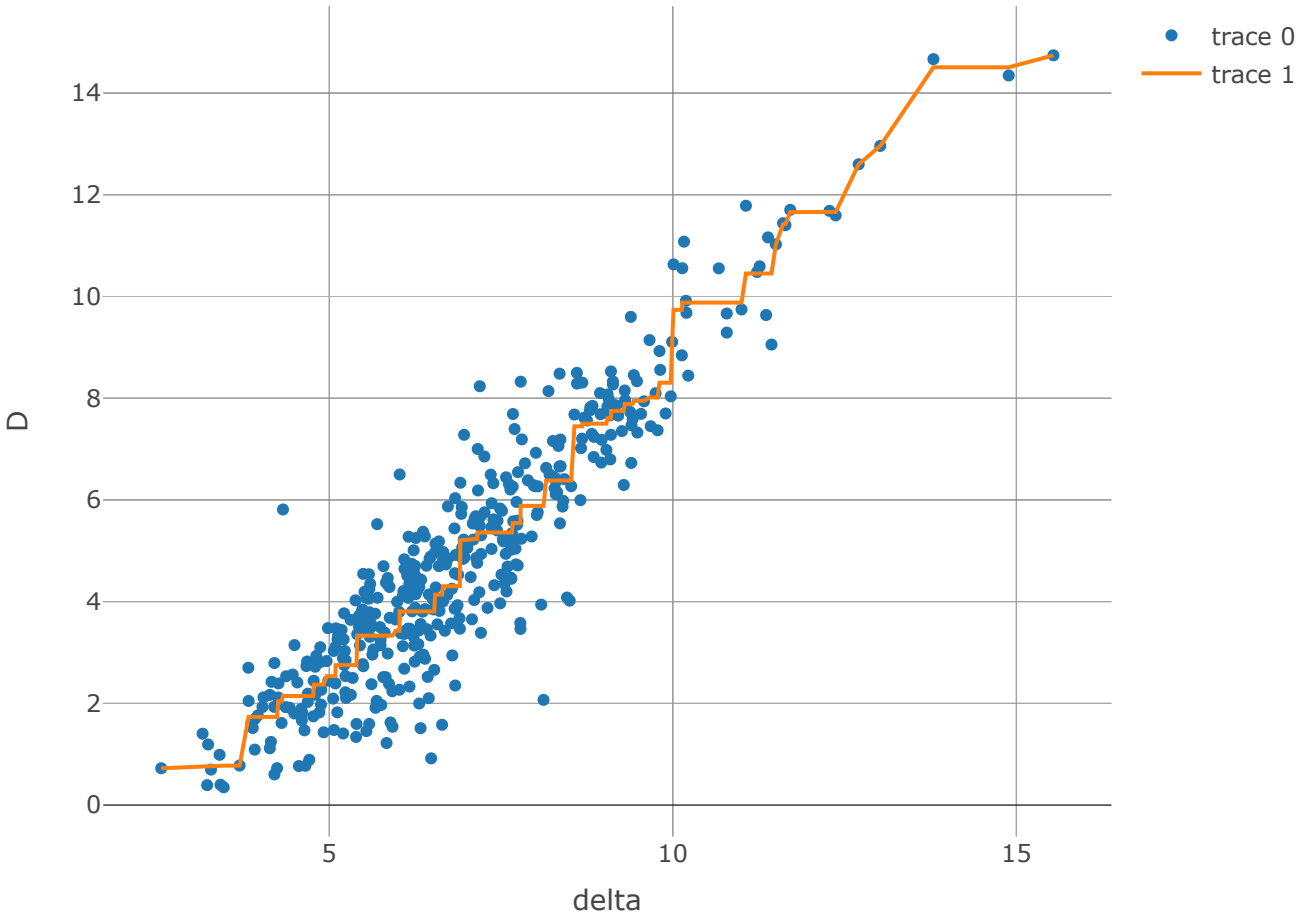

```
X <- BB_val$points[,1]
Y <- BB_val$points[,2]

plot_ly(type = "scatter",data = BB_val_df,x=~X, y=~Y ,text=baseball$Team,
        mode = "markers",color = ~baseball$League)
```

```
## Warning in RColorBrewer::brewer.pal(N, "Set2"): minimal value for n is 3, returning requested
palette with 3 different levels

## Warning in RColorBrewer::brewer.pal(N, "Set2"): minimal value for n is 3, returning requested
palette with 3 different levels
```



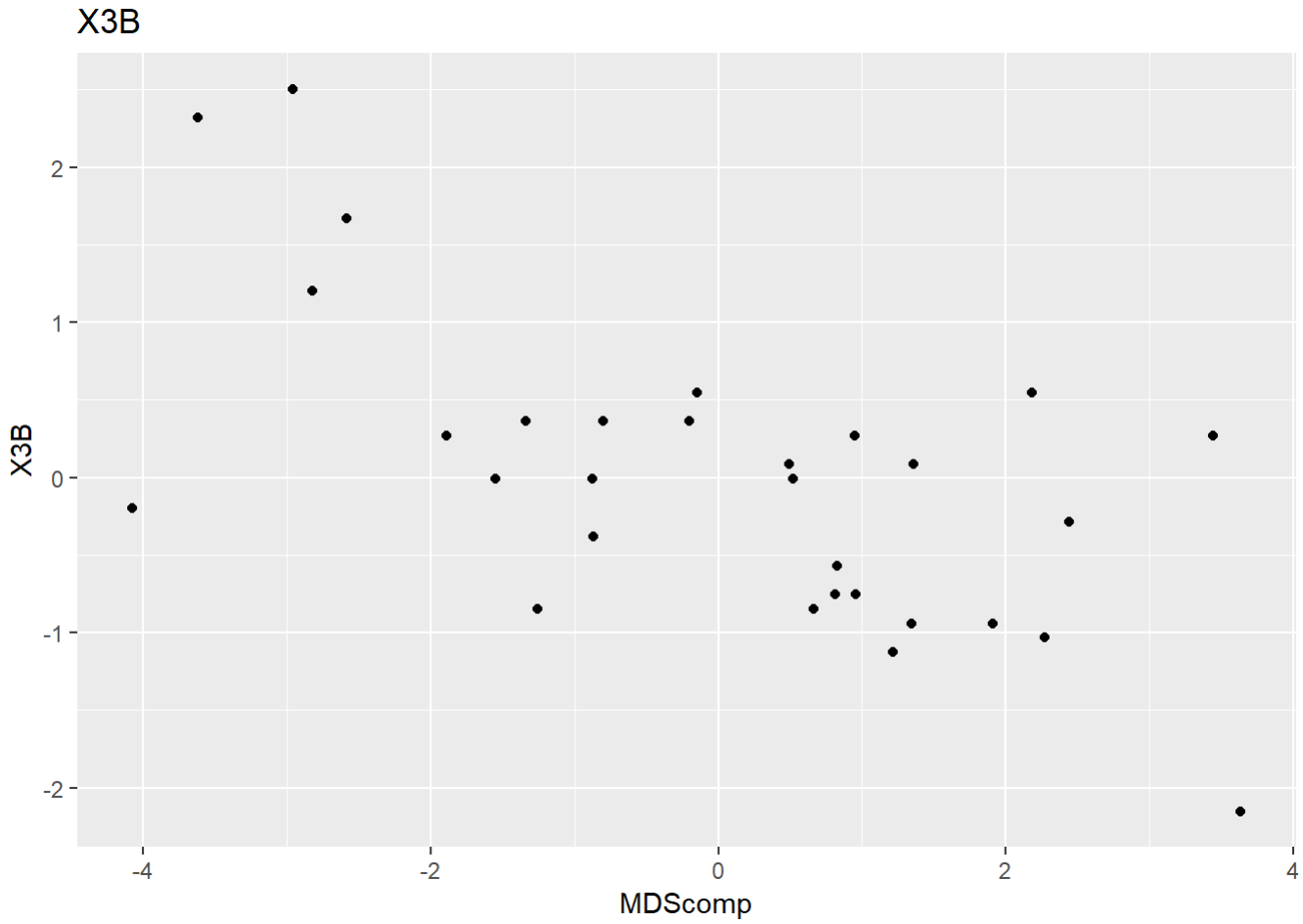
The AL league is more concentrated in the center whereas NL is more scattered away from the axis. The Y component seems to provide best differentiation between leagues as it can be seen from the graph that more NL points lie on negative Y than AL points.BOSTON RED SOX and ATLANTA BRAVES seems to be the outliers.

```
sh <- Shepard(dist_baseball, coords)
delta <-as.numeric(dist_baseball)
D <- as.numeric(dist(coords))

n = nrow(coords)
index = matrix(1:n, nrow=n, ncol=n)
index1 = as.numeric(index[lower.tri(index)])

n = nrow(coords)
index = matrix(1:n, nrow=n, ncol=n, byrow = TRUE)
index2 = as.numeric(index[lower.tri(index)])

row.names(baseball) <- baseball[,1]

sh_p <- plot_ly()%>%
  add_markers(x=~delta, y=~D, hoverinfo = 'text',
              text = ~paste('Obj1: ', rownames(baseball)[index1],
                            '<br> Obj 2: ', rownames(baseball)[index2]))%>%
  add_lines(x=~sh$x, y=~sh$yf)
sh_p
```

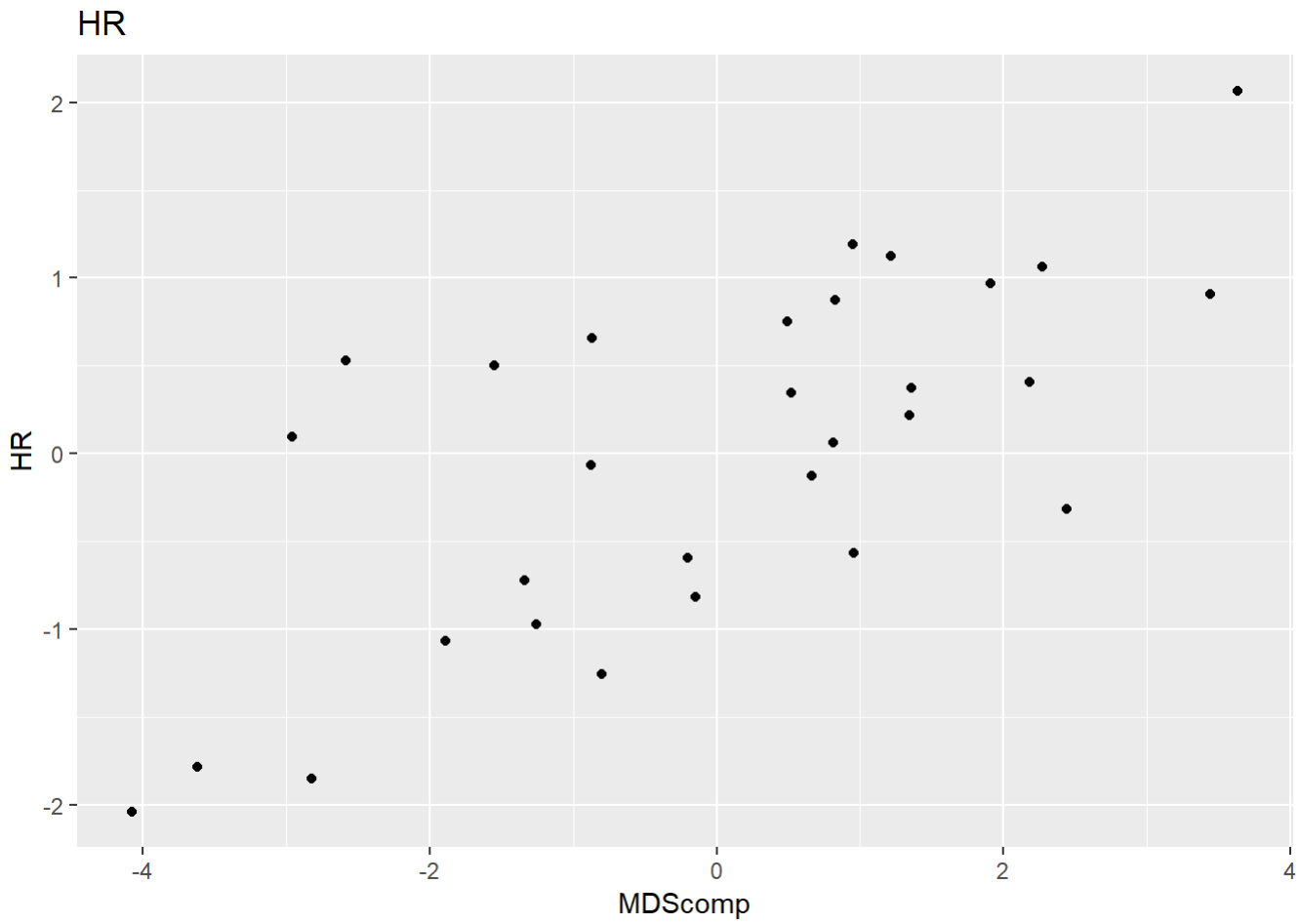


We think that it is a decent fit, plot converges with slight fluctuations there is a monotonic increase in the values but there lies some values that are away from others. Observation pairs Minnesota Twins vs Arizona Diamondbacks, Oakland Athletics VS Milwaukee Brewers were hard for MDS to map successfully.

```
## 2.4

sp_data <- cbind(baseball.numeric, coords[,2])
colnames(sp_data)[27] <- "MDScomp"
sp_data <- as.data.frame(sp_data)

varMDS_Fun <- function(given_col){
  ggplot(data=sp_data, aes(x=MDScomp, y=sp_data[,given_col])) + geom_point() +
    labs(y=colnames(sp_data)[given_col], title=paste(colnames(sp_data)[given_col]))
}

varMDS_Fun(9)
```




```
varMDS_Fun(10)
```



Home Run variable showed the strongest positive connection with MDS component Y and the Triples variable showed the strongest negetive connection with MDS component. By searching google we came to know that both variables Home Run and Triples are very important in baseball for scoring. The variable Y is highly influenced by some variables related to the process of making runs.

Lab-3_Group-19

Assignment 1

1

```
library(ggplot2)
library(plotly)
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

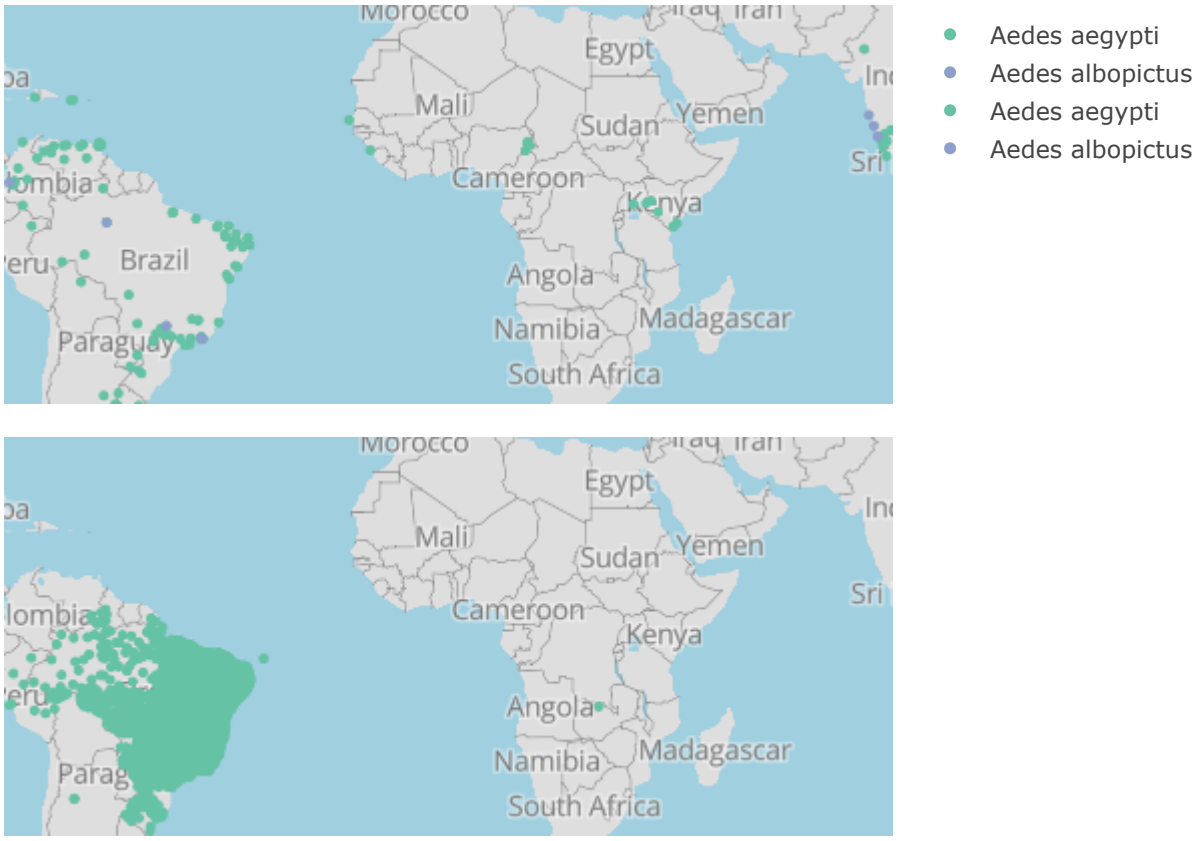
```
library(maps)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
mosquito <- read.csv("aegypti_albopictus.csv")
mosquito_df <- as.data.frame(mosquito)

# MailBox Token
Sys.setenv('MAPBOX_TOKEN' = 'pk.eyJ1IjoiZmFoaGE3ODAiLCJhIjoiY2ptOXFuM2Y1NGd4aTNwcXVoZmN2a2NzMiJ
9.5VOs9LfMU_jeSj6taxZ_IA')

mosquito_2004 <- mosquito[which(mosquito$YEAR == 2004),]
mosquito_2013 <- mosquito[which(mosquito$YEAR == 2013),]

mosquito_plot_2004 <- mosquito_2004 %>%
  plot_mapbox(lat = ~Y, lon = ~X, color = ~factor(VECTOR),
              mode = 'scattermapbox')

mosquito_plot_2013 <- mosquito_2013 %>%
  plot_mapbox(lat = ~Y, lon = ~X, color = ~factor(VECTOR),
              mode = 'scattermapbox')

subplot(mosquito_plot_2004, mosquito_plot_2013, nrows = 2)
```



Year 2004 South America continent seems to have more amount of Aedes aegypti type of mosquitoes than any other continent. Brazil has the most amount of it and it is concentrated along the costline. USA has more amount of Aedes albopictus than any other country. Year 2013 The highest density of Aedes aegypti is in Brazil while the highest density of Aedes albopictus is in Taiwan. There is a decrese in Aedes aegypti all over the world from the year 2004 to 2013 whereas there is an increse in the population of Aedes albopictus.

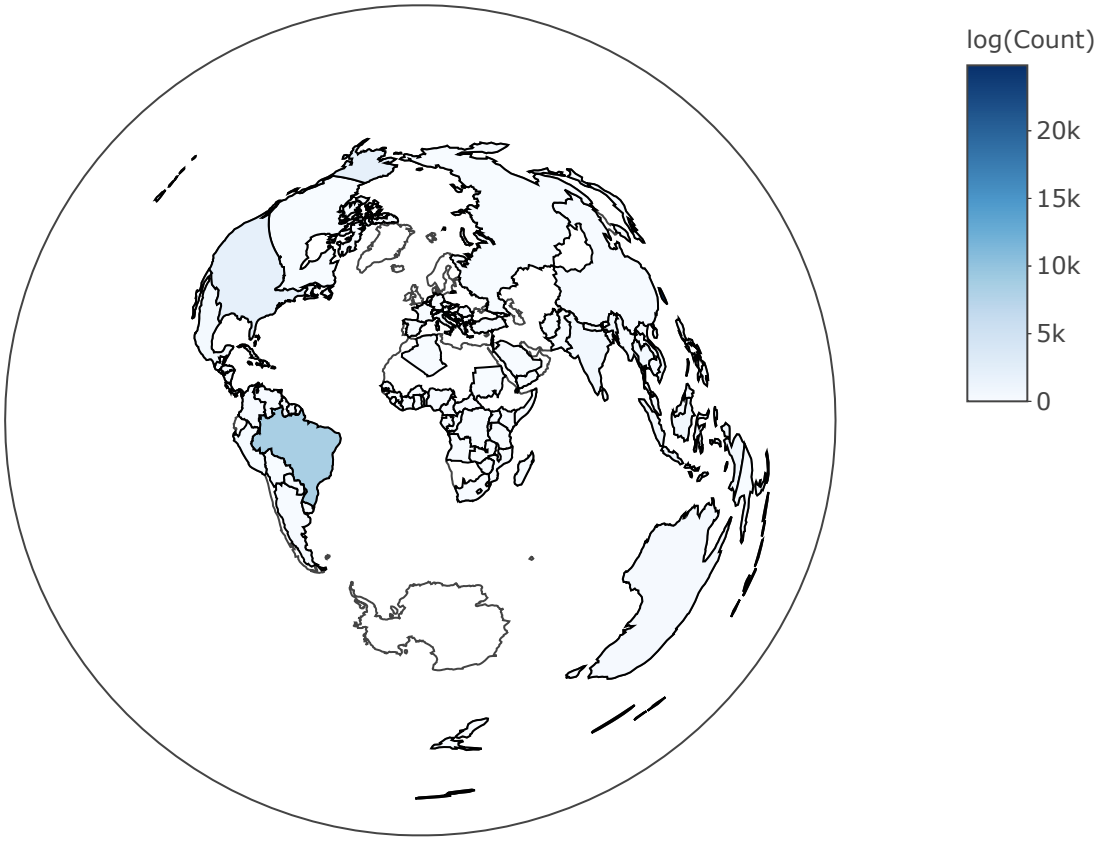
2

```
Z <- mosquito %>%
   count(mosquito$COUNTRY)

Z_df <- as.data.frame(Z)

col_Names <- c("Country","Count")
colnames(Z_df) <- col_Names

g <- list(
  projection = list(type = 'Equirectangular')
)
p <- plot_geo(Z_df) %>%
  add_trace(
    z = ~Count ,color=~Count, type="choropleth", colors = 'Blues',
    locations = ~Country,locationmode ='country names'
  ) %>%
  layout(
    geo = g
  )
p
```

We think that there is little information available from the graph since it shows only the total population of mosquitoes but not the population of mosquitoes of individual types.

3

```
g <- list(
  projection = list(type = 'azimuthal equidistant')
)
p2 <- plot_geo(Z_df) %>%
  add_trace(
    z = ~Count ,color=~log(Count), type="choropleth", colors = 'Blues',
    locations = ~Country,locationmode ='country names'
  ) %>%
  layout(
    geo = g
  )
p2
```

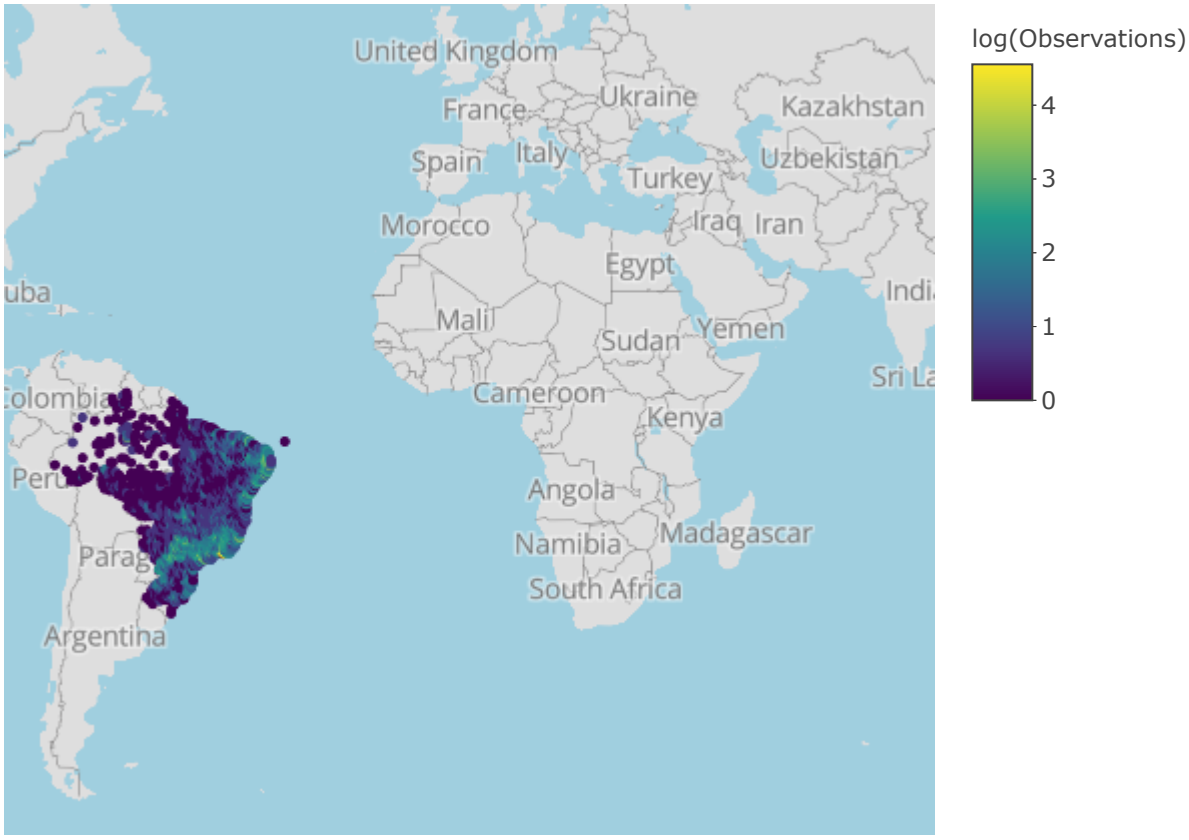


```
g <- list(
  projection = list(type = 'conic equal area')
)
p3 <- plot_geo(Z_df) %>%
  add_trace(
    z = ~Count ,color=~log(Count), type="choropleth", colors = 'Blues',
    locations = ~Country,locationmode ='country names'
  ) %>%
  layout(
    geo = g
  )
p3
```

Eqidistant projection: The country size shrinks to true size when bought to the centre which gives the actual idea about the plot, but the mosquito population cannot be cannot be compared realtive to any other country since the shape and size of other countries are distorted. Conic equal area projection: The actual shape of countries towars the south pole is distorted while the counties towards the north pole has the actual size and shape, so comparing the mosquito population relatively is not feasible.

4

```
mosquito_2013 <- mosquito[which(mosquito$YEAR == 2013),]
mosquito_2013_Brazil <- mosquito[which(mosquito$COUNTRY == "Brazil"),]

### a
mosquito_2013_Brazil$X1 <- cut_interval(mosquito_2013_Brazil$X, 100)
### b
mosquito_2013_Brazil$Y1 <- cut_interval(mosquito_2013_Brazil$Y, 100)

### c
meanCal <- mosquito_2013_Brazil %>%
  group_by(X1,Y1) %>%
  summarise(meanX = mean(X), meanY = mean(Y), Observations = n())

### d
BrazilPlot <- plot_mapbox(data= meanCal,x= ~meanX, y= ~meanY, color = ~log(Observations) ,mode=
"scattermapbox")
BrazilPlot
```



We can see from the map that in Brazil most of the parts are affected, by concentrating on specific area using cut interval it is more easy to analyze. Areas like Rio de Janerio, Volta Redonda are most affected areas and after these some parts of Recife are also affected more than any other place, Whereas we are not able to clearly see other parts because they have two colors of dark and light blue.

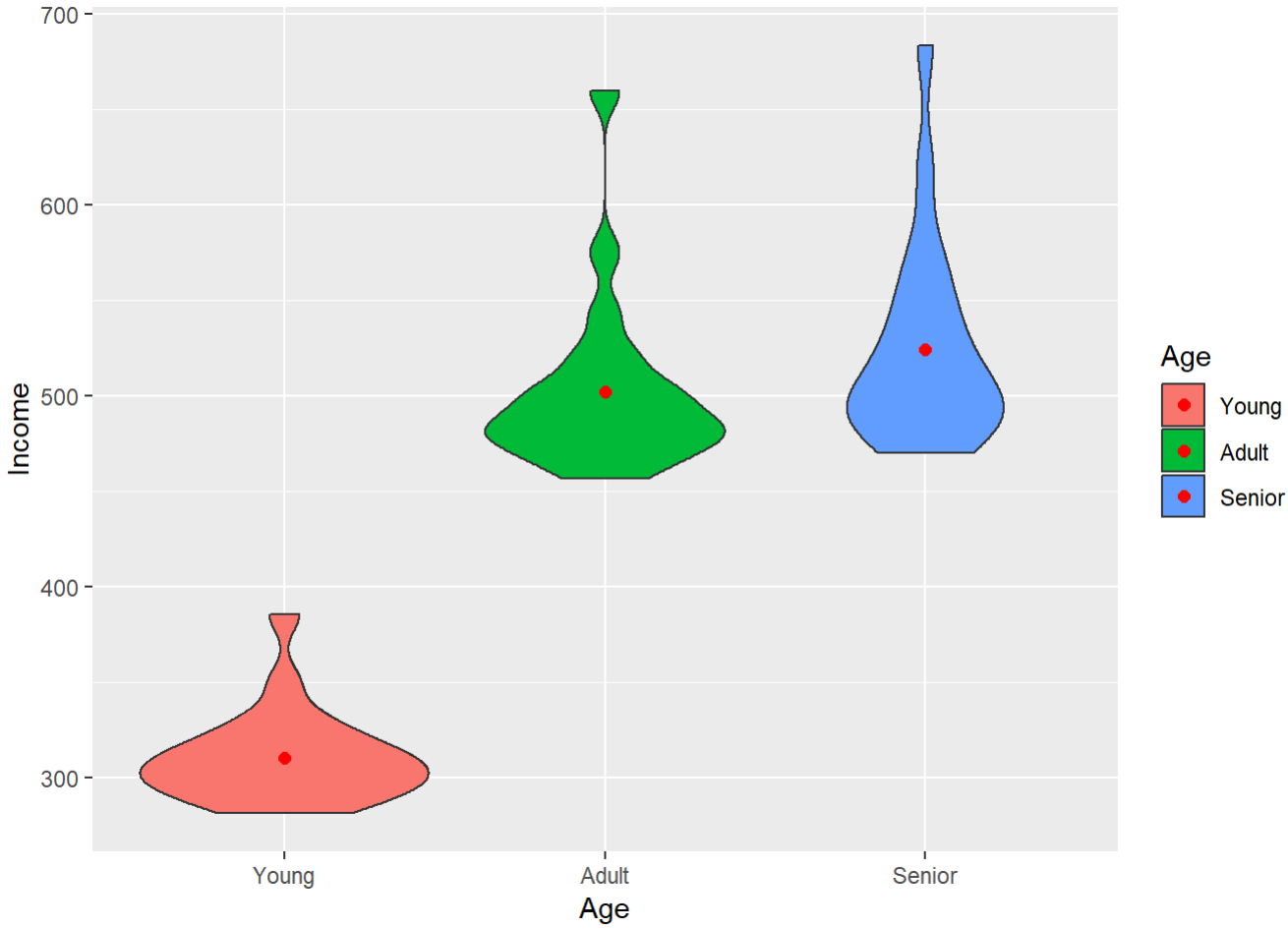
Assignment 2

1

```
data <- read.csv("SwedData.csv")

levels(data$age) <- c("Young","Adult" ,"Senior")

data <- data[order(data$age),]
data1 <- filter(data , age == "Senior")
data2 <-  filter(data , age == "Adult")
data3 <-  filter(data , age == "Young")

transform_Data <- cbind(data1,data2,data3)

transform_Data <- transform_Data[,-4]
transform_Data <- transform_Data[,-6]

transform_Data <- transform_Data[,c(-2,-4,-6)]

colnames(transform_Data) <- c("Region","Senior","Adult","Young")

columnsplit <- strsplit(levels(transform_Data$Region)," ")
levels(transform_Data$Region) <- sapply(columnsplit,function(x) x[[2]])
```

2

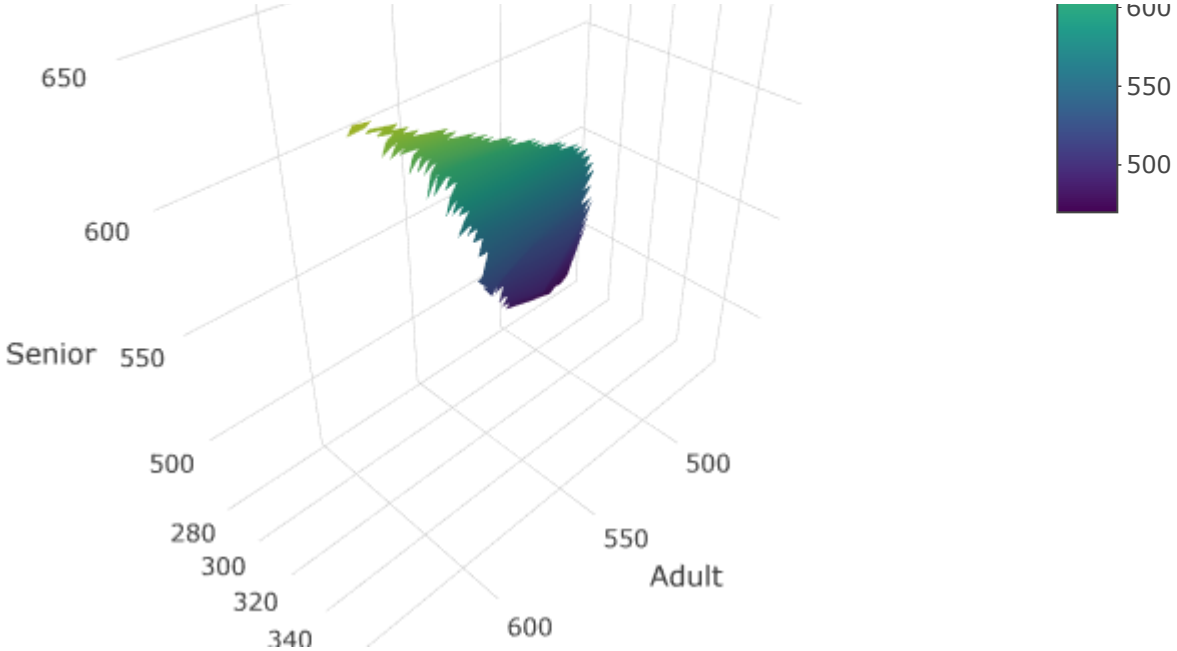
```
Salary <- data$X2016
Age <- data$age

mean_plot <- ggplot(data, aes(y=as.numeric(Salary), x=factor(Age), fill = Age)) +
  xlab("Age") + ylab("Income")+
  geom_violin()

mean_plot <- mean_plot + stat_summary(fun.y=mean, geom="point", size=2, color="red")
mean_plot
```



According to the plot the mean income of young is the lowest and for the senior it is the highest. The income for young people lies around the mean whereas for adult the salary varies a lot from the mean and for seniors majority of the people's income seem to lie below the mean income.

3

```
library(plotly)
library(akima)

intpolt <- interp(x=transform_Data$Adult,y=transform_Data$Young,z=transform_Data$Senior, duplica
te = "mean")
plot_ly(x=~intpolt$x, y=~intpolt$y, z=~intpolt$z, type = "surface") %>%
  layout(
    title = "Income Dependence of Seniors on Adult and Young Incomes",
    scene = list(
      xaxis = list(title = "Adult"),
      yaxis = list(title = "Young"),
      zaxis = list(title = "Senior")
    ))
```



All the three age groups follow the same trend and are linearly dependent on each other following linear path. Linear regression would be suitable to model this dependence. It is quite difficult to analyze all the three factors at once.

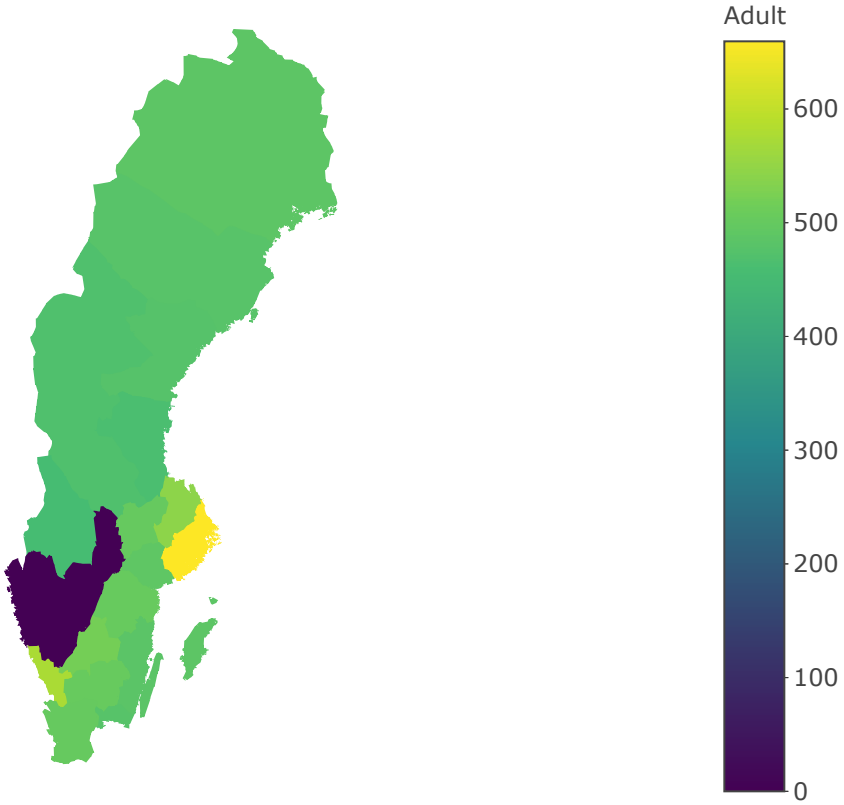
4

```
rds <- readRDS("gadm36_SWE_1_sf.RDS")

rownames(transform_Data) = transform_Data$Region

rds$Young = transform_Data[rds$NAME_1,"Young"]
rds$Young[is.na(rds$Young)] = 0

ch_plot1 <- rds %>% plot_ly() %>%
    add_sf(split=~NAME_1,color=~Young, showlegend=F, alpha=1) #%>%
ch_plot1
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```


```
rds$Adult = transform_Data[rds$NAME_1,"Adult"]
rds$Adult[is.na(rds$Adult)] = 0

ch_plot2 <- rds %>% plot_ly() %>%
    add_sf(split=~NAME_1,color=~Adult, showlegend=F, alpha=1) #%>%
ch_plot2
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

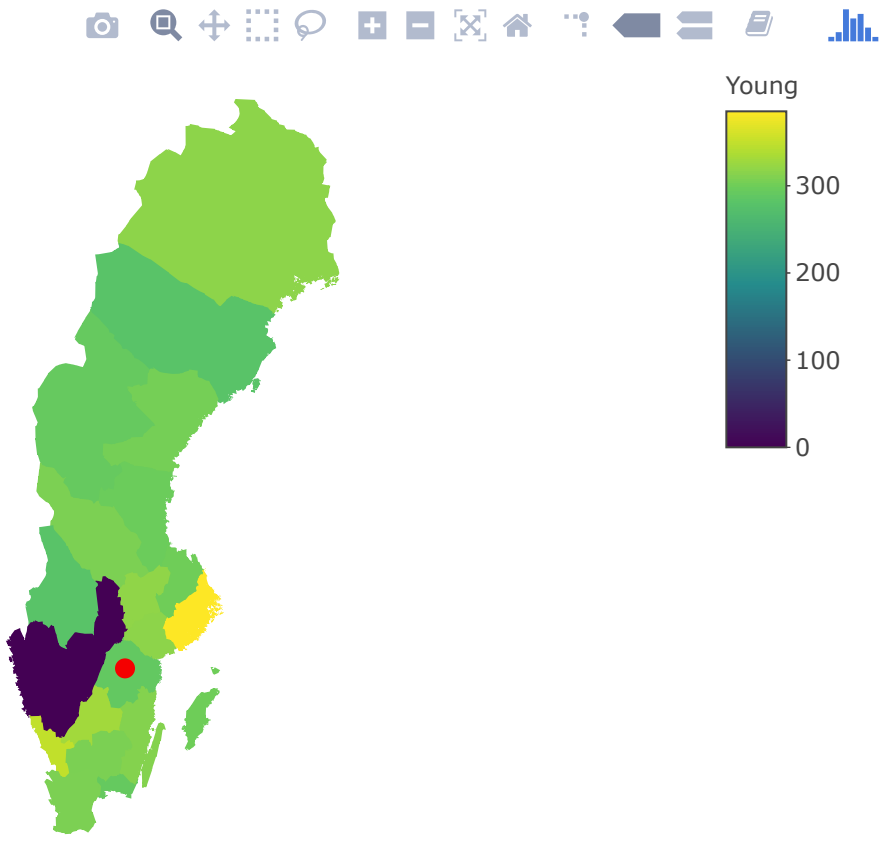


The income for Adult is found to be more all over the country than that compared to that of young, by looking at the scale. The income for young seem to vary in different places all over Sweden while for adult it remains fairly the same for most part of the country. Both Young and adult and earn highest and the lowest income in the same areas i.e Stockholm and vastra Gotaland respectively.

5

```
Linkoping_ch_plot <- rds %>% plot_ly() %>%
    add_sf(split=~NAME_1,color=~Young, showlegend=F, alpha=1) %>%
    add_markers(y=~58.409814,x=~15.624525,hoverinfo="text",text="Linköping",
                marker=list(color="#f40000" , size=10))

Linkoping_ch_plot
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```



In the Last part we placed a red mark at Linkoping with vertices y=58.409814,x=15.624525. Which tells us where we are.

Lab4 Group19

fahha780, vinbe289

Assignment 1

```
### 1

df_data <- as.data.frame(read.delim("prices-and-earnings.txt"))
data <- df_data[,c(1,2,5,6,7,9,10,16,17,18,19)]
row.names(data) <- data[,1]
scaleData <- scale(data[,2:11])
```

```
### 2
library(ggplot2)
library(plotly)
```

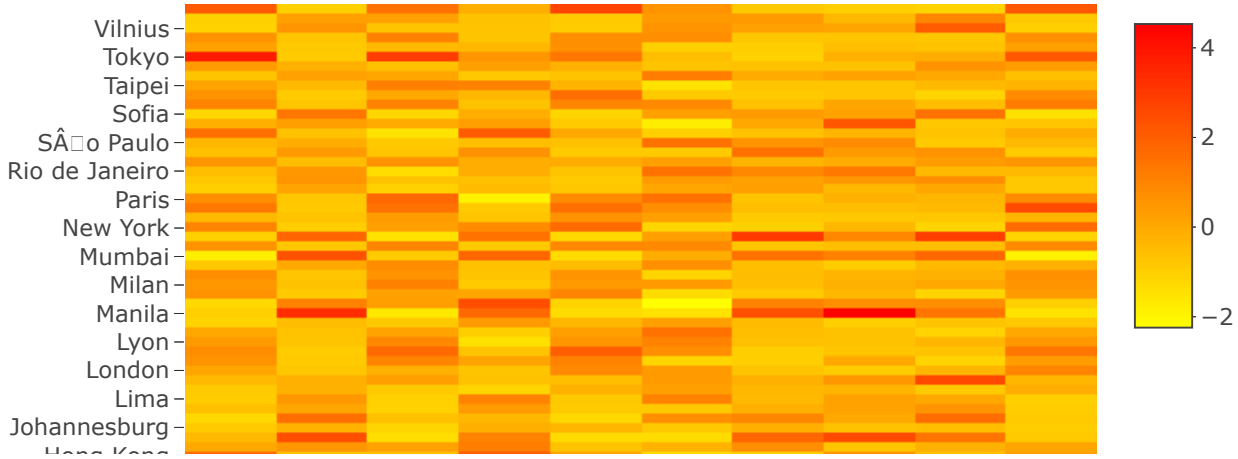
```
##
## Attaching package: 'plotly'
```

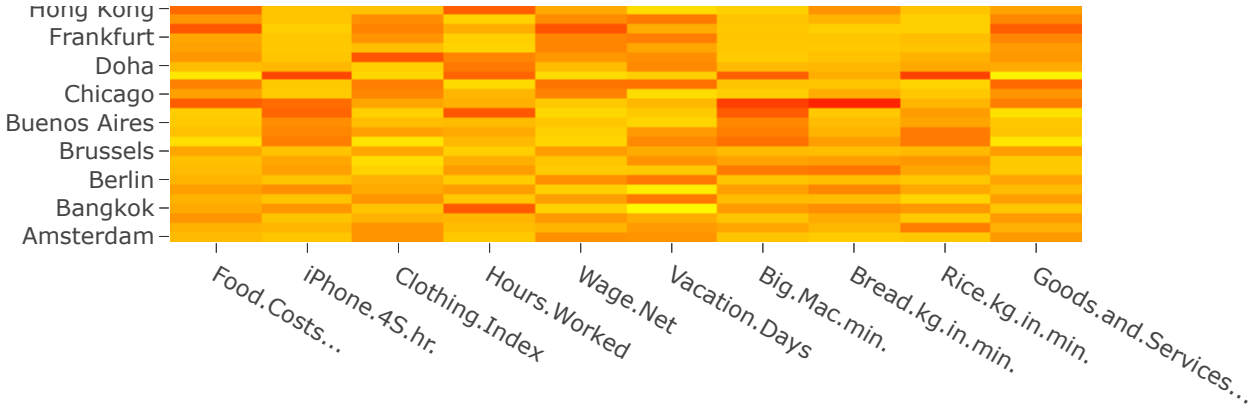
```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

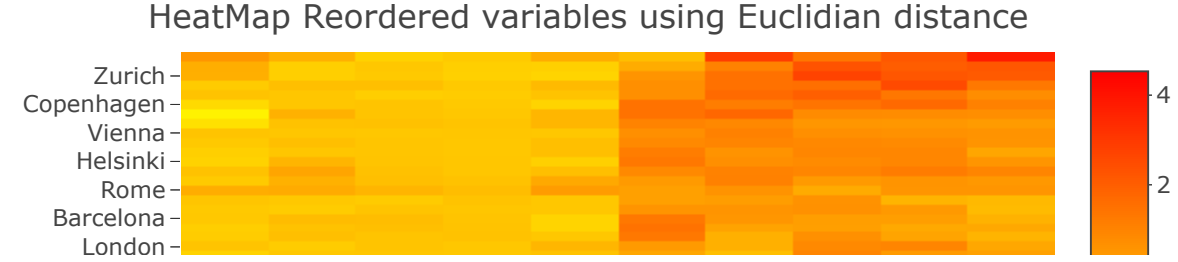
```
plot_ly(x=colnames(scaleData), y=rownames(scaleData),
    z=scaleData, type="heatmap", colors = colorRamp(c("yellow","red")))
```

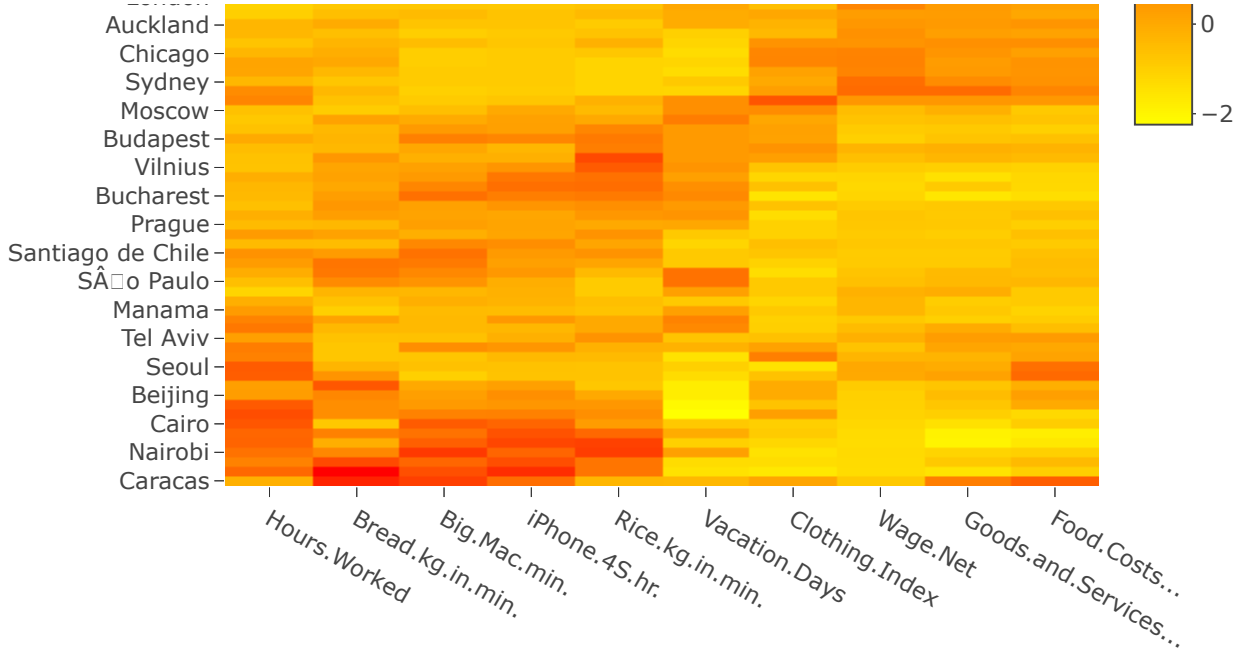




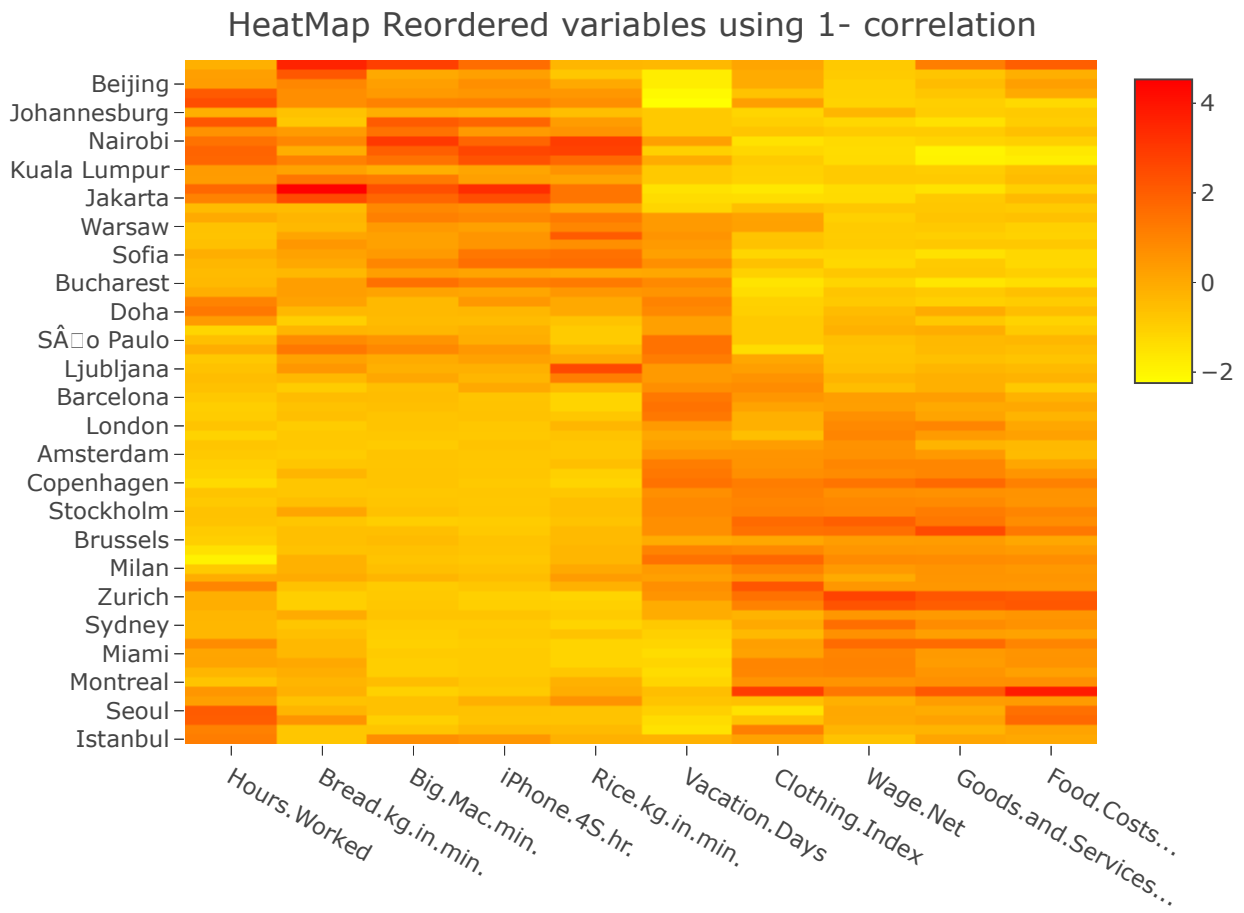
We are not able to see any clusters or outliers(clearly) here.

```
### 3

library(seriation)

rowdist_euc <- dist(scaleData)
rowdist_ser_euc <- seriate(rowdist_euc, method = "OLO")
roworder_euc <-get_order(rowdist_ser_euc)

coldist_euc <- dist(t(scaleData))
coldist_ser_euc <- seriate(coldist_euc, method = "OLO")
colorder_euc <- get_order(coldist_ser_euc)

reordered_euc <- scaleData[rev(roworder_euc),colorder_euc]

rowdist_cor <- 1 - as.dist(cor(t(scaleData)))
rowdist_ser_cor <- seriate(rowdist_cor, method = "OLO")
roworder_cor <- get_order(rowdist_ser_cor)

coldist_cor <- 1 - as.dist(cor(scaleData))
coldist_ser_cor <- seriate(coldist_cor, method = "OLO")
colorder_cor <- get_order(coldist_ser_cor)

reordered_cor <- scaleData[rev(roworder_cor),colorder_cor]

plot_3.1 <- plot_ly(x=colnames(reordered_euc), y=rownames(reordered_euc),
    z=reordered_euc, type="heatmap", colors = colorRamp(c("yellow","red"))) %>%
      layout(title= "HeatMap Reordered variables using Euclidian distance")

plot_3.2 <- plot_ly(x=colnames(reordered_cor), y=rownames(reordered_cor),
    z=reordered_cor, type="heatmap", colors = colorRamp(c("yellow","red"))) %>%
  layout(title= "HeatMap Reordered variables using 1- correlation")

plot_3.1
```





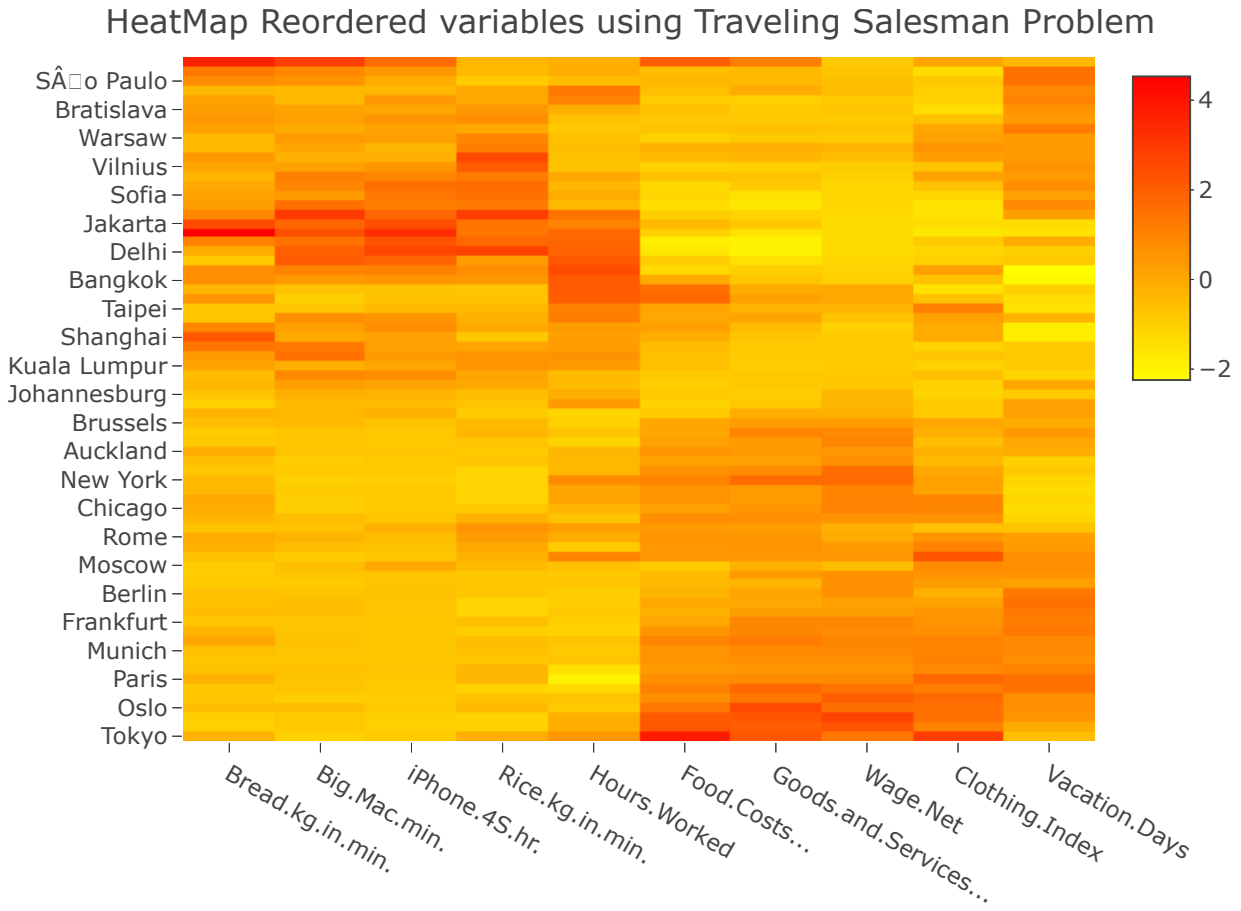

```
plot_3.2
```



The first plot seems to be easier to analyse since Clusters and outliers can be identified easily. In the first plot the Y axis variables i.e. cities are reordered to optimize Hamiltonion Path Length based on Euclidian distance. We can see clusters in ‘iPhone.4S.hr’ variable column for first 30 cities which means the hours of work (at the average wage) needed to buy an iPhone in these cities are very close to each other. The net wage seems to be the same and the least in last 10 cities

i.e. Cities from Shanghai to Caracas. We can easily see a few outliers as well such as the minimum price for a kg of bread is highest in Manila followed by Caracus and Jakarta. Paris is an oulier if the variable Hours.worked is considered. Tokyo is an outlier for the variable clothing index and Food cost seems to be the highest too.

```
### 4

roworder_tsp <- get_order(seriate(rowdist_euc, method = "TSP"))
colorder_tsp <- get_order(seriate(coldist_euc, method = "TSP"))
reordered_tsp <- scaleData[rev(roworder_tsp),colorder_tsp]
plot_4 <- plot_ly(x=colnames(reordered_tsp), y=rownames(reordered_tsp),
    z=reordered_tsp, type="heatmap", colors = colorRamp(c("yellow","red"))) %>%
      layout(title= "HeatMap Reordered variables using Traveling Salesman Problem")
plot_4
```



This heatmap produced by reordering variables to optimize Hamiltonian Path Length by using TSP as solver seems to be a better plot for analysis then heatmap produced by using HC as solver. By comparing the objective function values such as Hamiltonian Path Length and Gradient measure achieved by row permutations of TSP and HC solvers it can be seen that TSP was the best solver since Hamiltonian Path Length and Gradient measure was lesser than that of HC solver.

```
HC <- criterion(rowdist_euc,order=seriate(rowdist_euc, method = "GW"),method = c("Gradient_raw","Path
_length"))

TSP <- criterion(rowdist_euc,order=seriate(rowdist_euc, method = "TSP"),method = c("Gradient_raw","Pa
th_length"))
```

```
cat("\nHC-Solver\n")
```

```
##
## HC-Solver
```

```
HC
```

```
## Gradient_raw  Path_length
##   59506.0000     128.5114
```

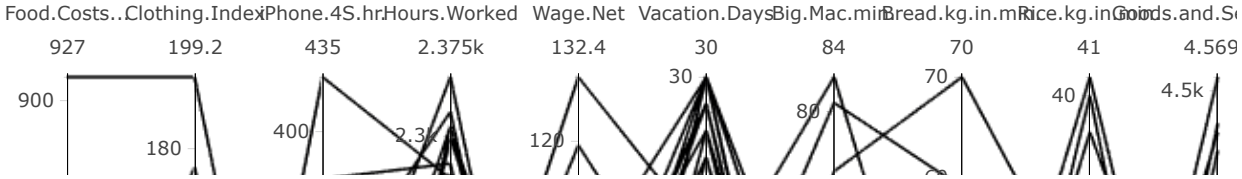
```
cat("\nTSP\n")
```

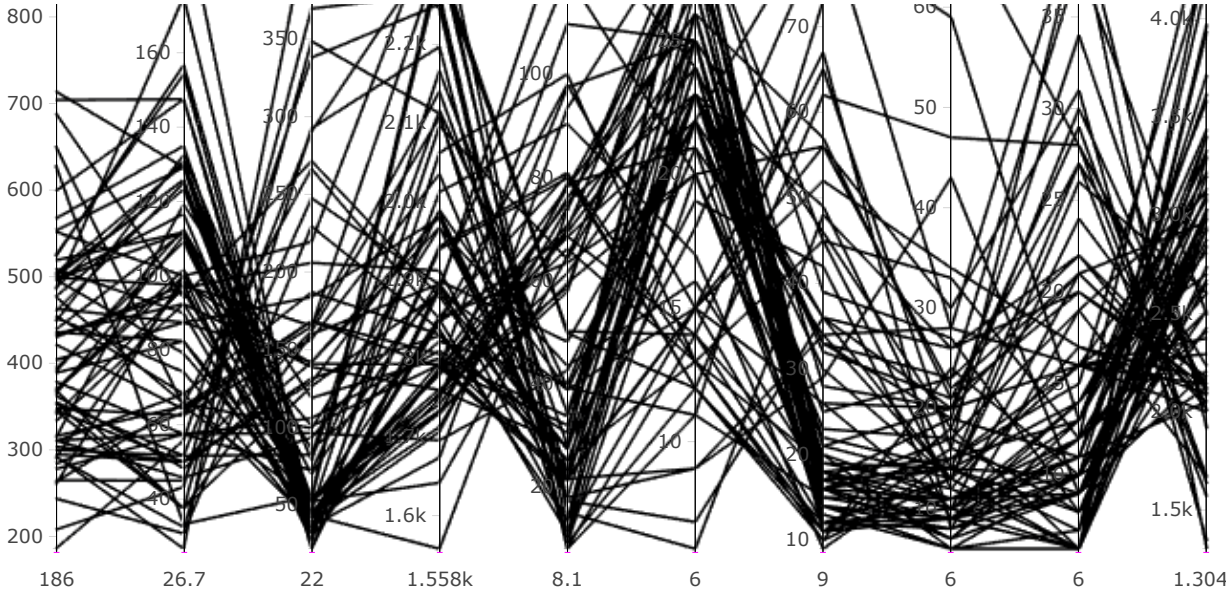
```
##
## TSP
```

```
TSP
```

```
## Gradient_raw  Path_length
##    36290.000      120.898
```

```
### 5

library(ggplot2)
library(plotly)

p1 <- as.data.frame(data) %>%
  plot_ly(type = 'parcoords',
          dimensions = list(
            list(label = "Food.Costs...", values =~Food.Costs...),
            list(label = "Clothing.Index", values =~Clothing.Index),
            list(label = "iPhone.4S.hr.", values =~iPhone.4S.hr.),
            list(label = "Hours.Worked", values =~Hours.Worked),
            list(label = "Wage.Net", values =~Wage.Net),
            list(label = "Vacation.Days", values =~Vacation.Days),
            list(label = "Big.Mac.min.", values =~Big.Mac.min.),
            list(label = "Bread.kg.in.min.", values =~Bread.kg.in.min.),
            list(label = "Rice.kg.in.min.", values =~Rice.kg.in.min.),
            list(label = "Goods.and.Services...", values =~Goods.and.Services...)
          )
  )
p1
```



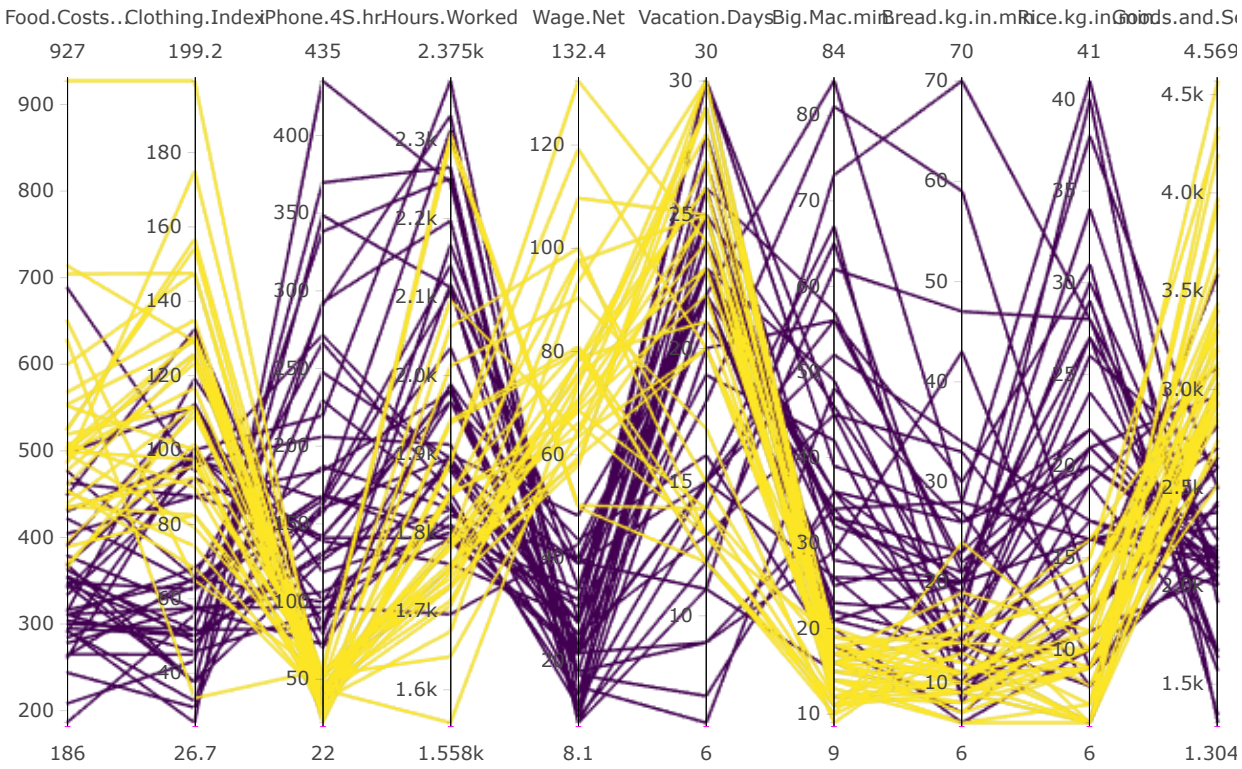


Quite a few number of outliers exist in the dataset according to the plot. When the coulumns variables were rearranged we could find clusters in Variable

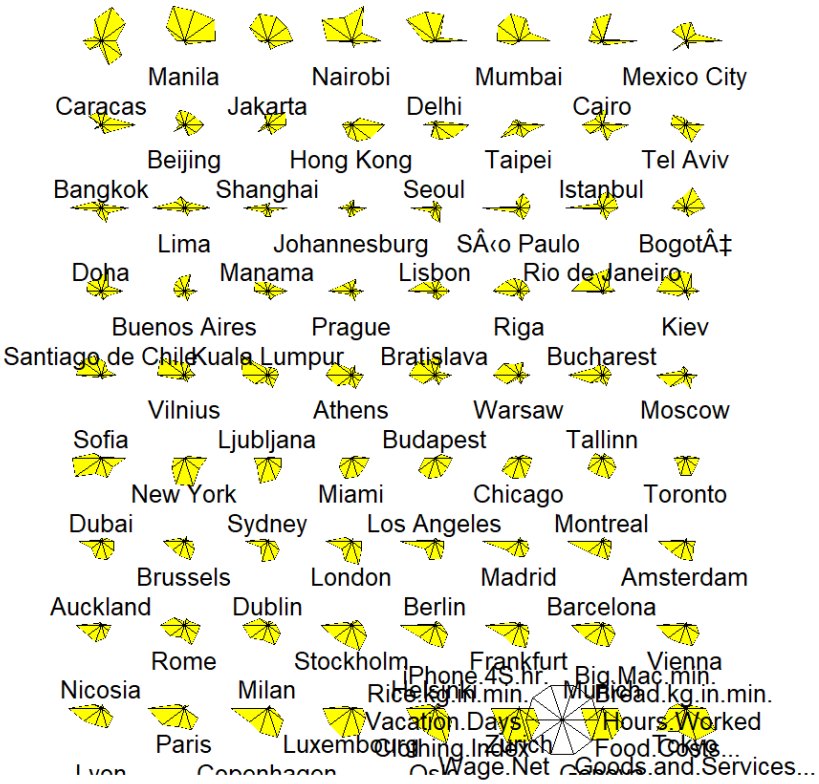
1. iphone.4s.hr. around the variable value 50
2. Big.mac.min around the value 20
3. Bread.kg.in.min at the value around 20 to 50
4. Rice.Kg.in.min around 10 to 15

By arranging and analyzing the plot we colored our plot on the basic of Wage.Net which we think is the most important factor in deciding the other variables by visualizing different variables dependence on Wage.Net. We found as that as the Wage.Net is high hourly work, Rice and bread consumption is low also Big Mac and Iphone variables are low whereas Vacation, clothing index and Good and services is high. Similarly if the wage is low this effects thesse variables in the opposite way.

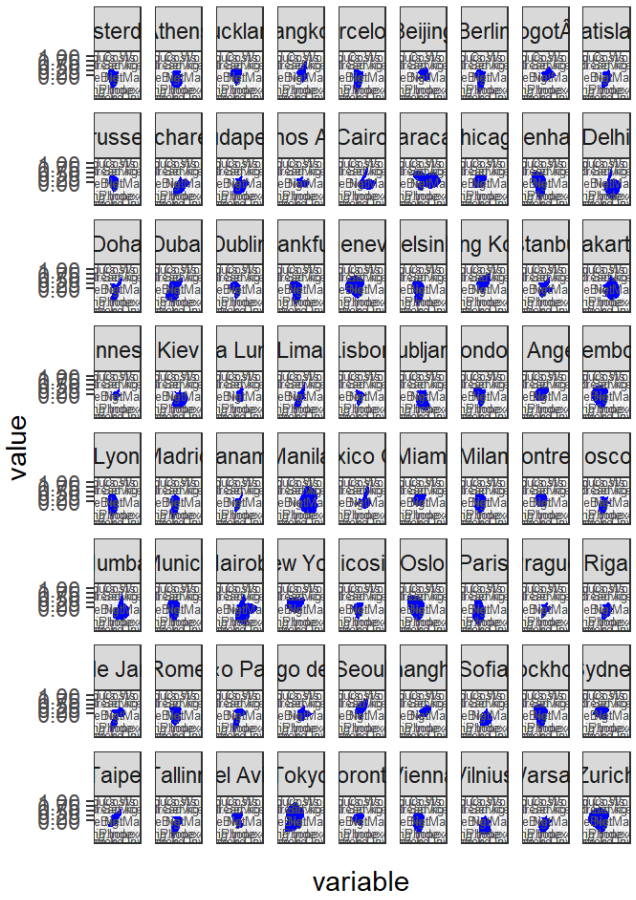
```
p2 <- data %>%
  mutate(data = as.integer(Wage.Net > 48.6875)) %>%
  plot_ly(type = 'parcoords',
            dimensions = list(
              list(label = "Food.Costs...", values =~Food.Costs...),
              list(label = "Clothing.Index", values =~Clothing.Index),
              list(label = "iPhone.4S.hr.", values =~iPhone.4S.hr.),
              list(label = "Hours.Worked", values =~Hours.Worked),
              list(label = "Wage.Net", values =~Wage.Net),
              list(label = "Vacation.Days", values =~Vacation.Days),
              list(label = "Big.Mac.min.", values =~Big.Mac.min.),
              list(label = "Bread.kg.in.min.", values =~Bread.kg.in.min.),
              list(label = "Rice.kg.in.min.", values =~Rice.kg.in.min.),
              list(label = "Goods.and.Services...", values =~Goods.and.Services...)
            ),
              line = list(color = ~as.numeric(data))
  )
p2
```




```
### 6
#Juxtaposed

#Ugly graphics
stars(reordered_euc, key.loc=c(15,2), draw.segments=F, col.stars =rep("Yellow", nrow(data)))
```

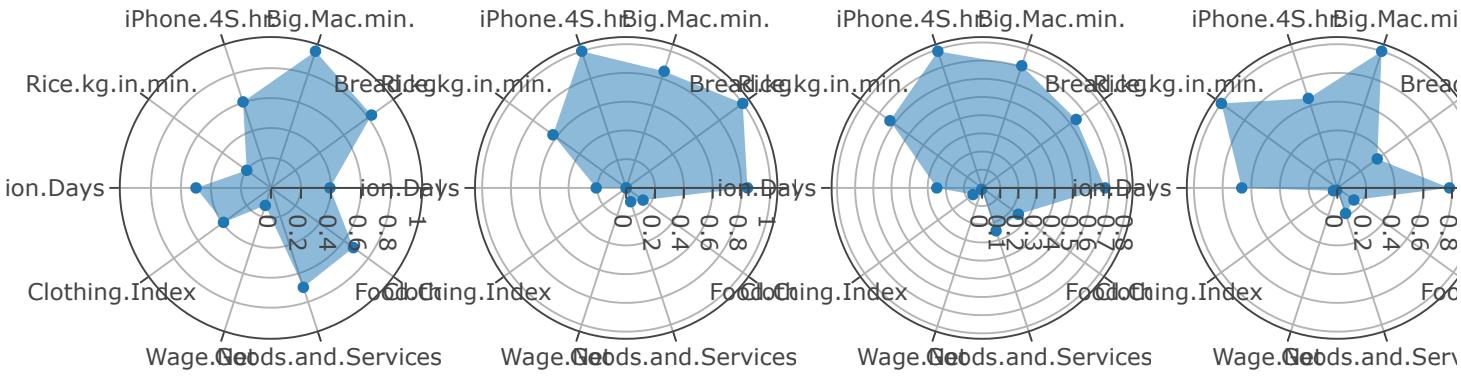



```
##ggplot2
library(scales)
library(magrittr)
library(dplyr)
library(ggplot2)

 data1 <- as.data.frame(reordered_euc) %>% mutate_all(funs(rescale))

data1$name = rownames(reordered_euc)
data2 <- data1%>%tidyr::gather(variable, value, -name, factor_key=T)%>%arrange(name)
p <- data2 %>%
  ggplot(aes(x=variable, y=value, group=name)) +
  geom_polygon(fill="blue") +
  coord_polar() + theme_bw() + facet_wrap(~ name) +
  theme(axis.text.x = element_text(size = 5))
p
```

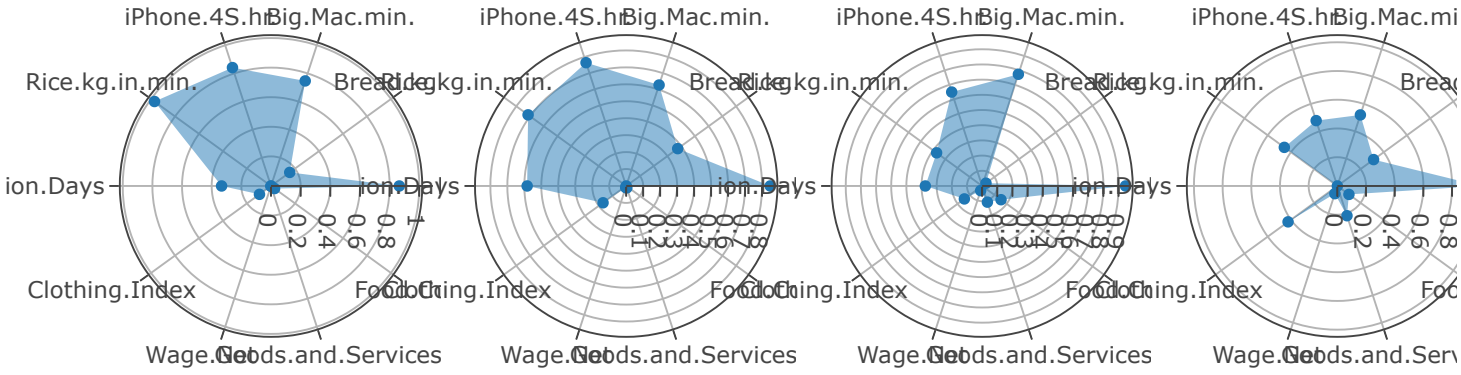



```
## Plotly

library(plotly)
library(dplyr)
library(scales)

Ps=list()
nPlot=72

as.data.frame(reordered_euc) %>%
  add_rownames( var = "group" ) %>%
  mutate_each(funs(rescale), -group) -> reordered_euc_radar

for (i in 1:nPlot){
  Ps[[i]] <- htmltools::tags$div(
    plot_ly(type = 'scatterpolar',
            r=as.numeric(reordered_euc_radar[i,-1]),
            theta= colnames(reordered_euc_radar)[-1],
            fill="toself")%>%
      layout(title=reordered_euc_radar$group[i]), style="width: 25%;")
}

h <-htmltools::tags$div(style = "display: flex; flex-wrap: wrap", Ps)

htmltools::browsable(h)
```

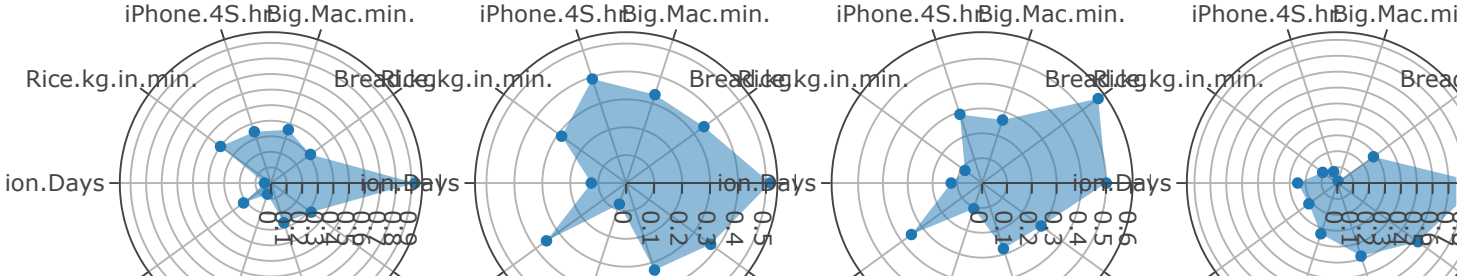
Caracas Manila Jakarta Nairobi



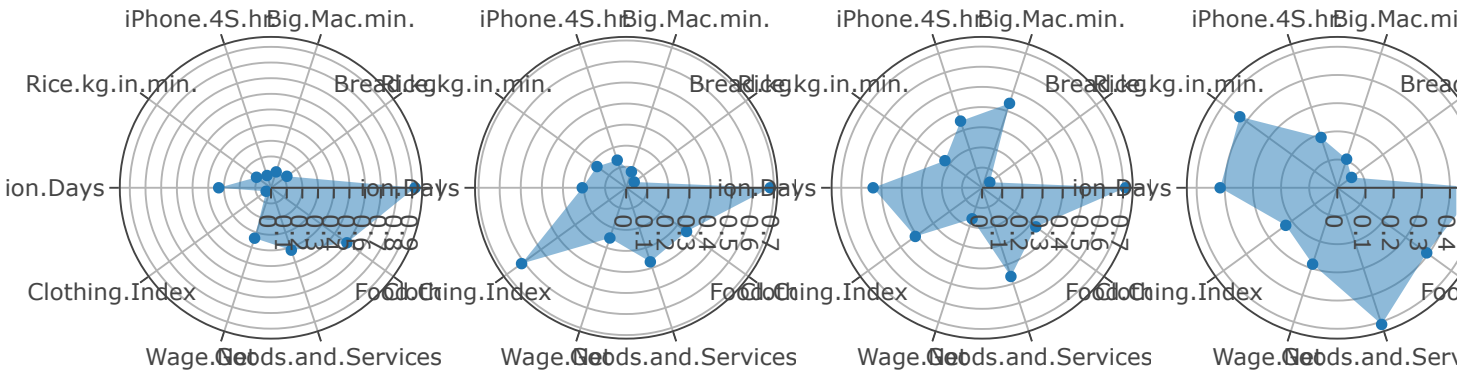
Delhi Mumbai Cairo Mexico City



Bangkok Beijing Shanghai Hong Kong



Seoul

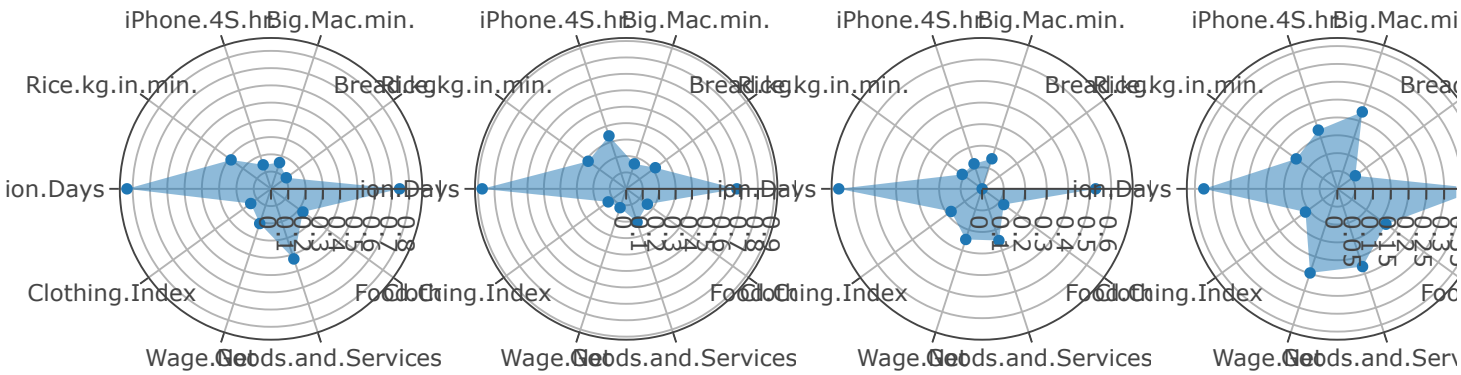


Taipei

Istanbul

Tel Aviv

Doha



Lima

Manama

Johannesburg

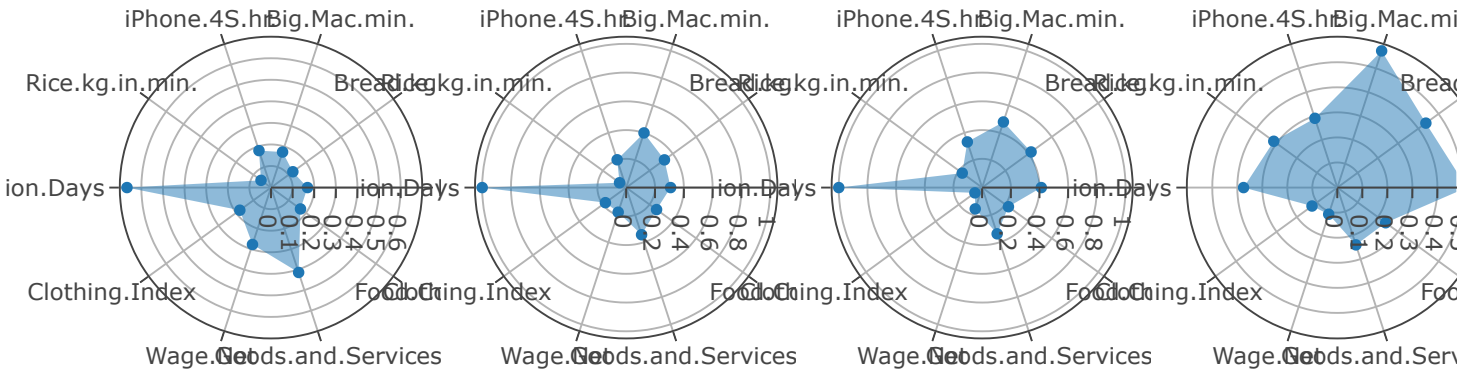
Lisbon

SA□o Paulo

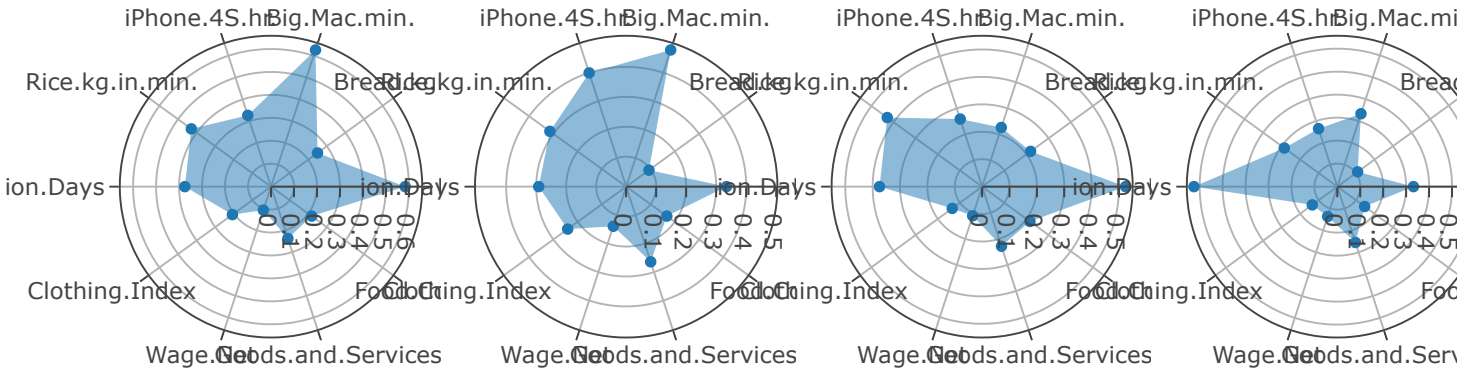
Rio de Janeiro

Bogot□

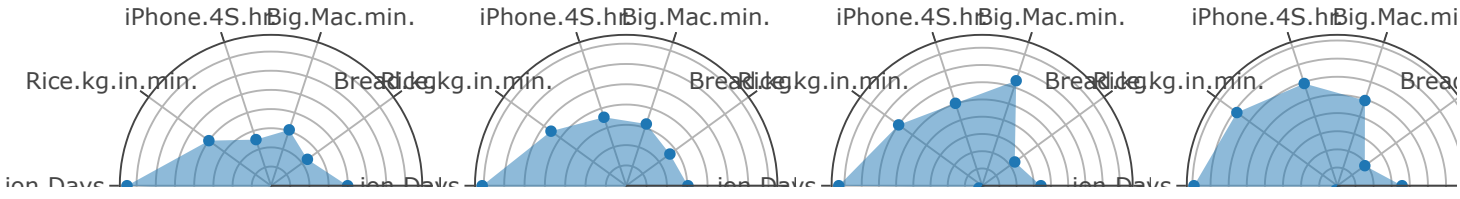
Lisbon SA□o Paulo Rio de Janeiro BogotA□



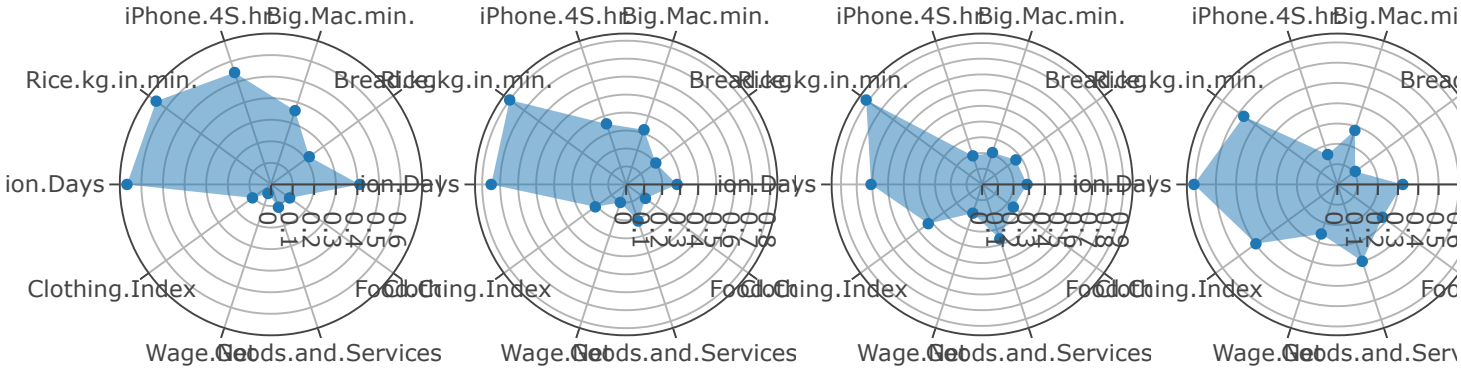
Santiago de Chile Buenos Aires Kuala Lumpur Prague



Bratislava Riga Bucharest Kiev



Sofia

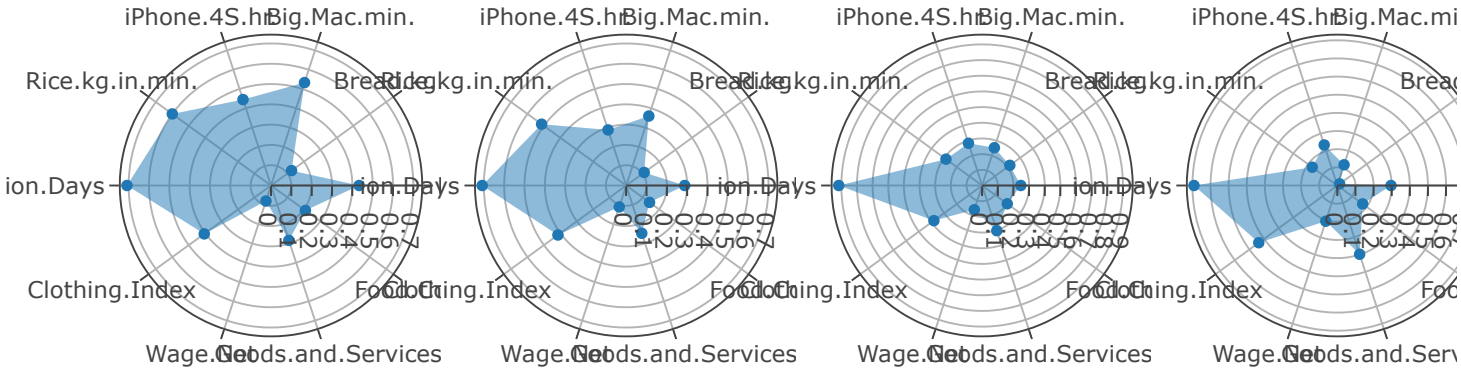


Vilnius

Ljubljana

Athens

Budapest

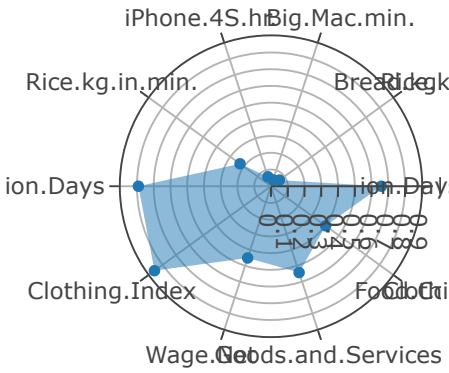


Warsaw

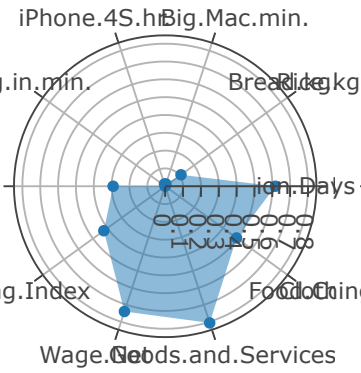
Tallinn

Moscow

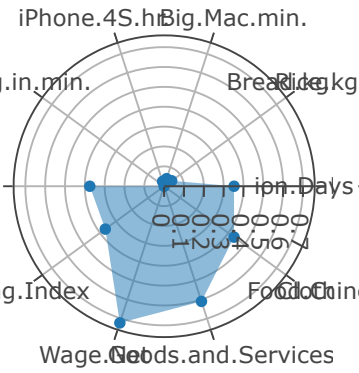
Dubai



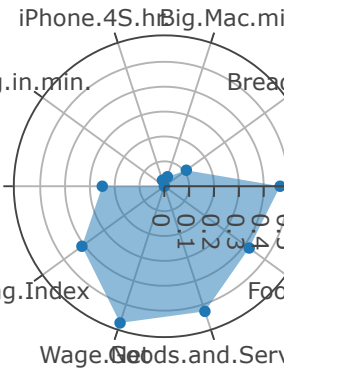
New York



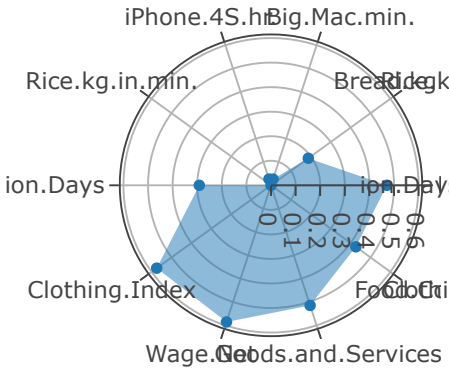
Sydney



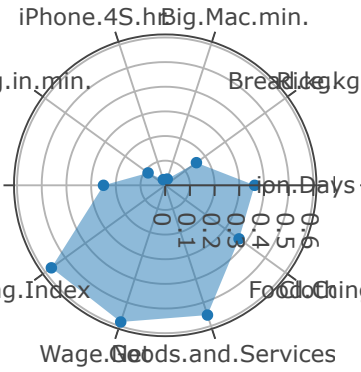
Miami



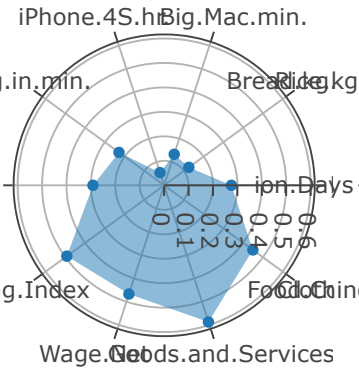
Los Angeles



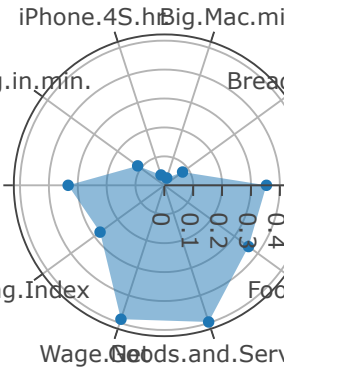
Chicago



Montreal

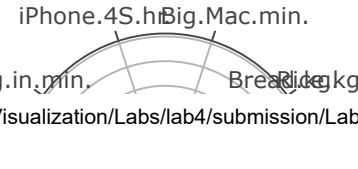


Toronto

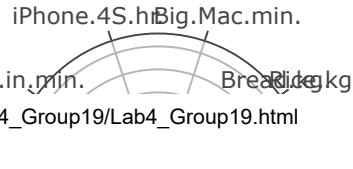


Auckland

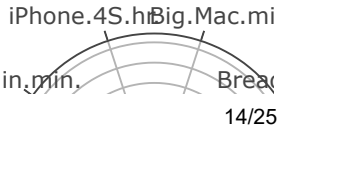
Brussels

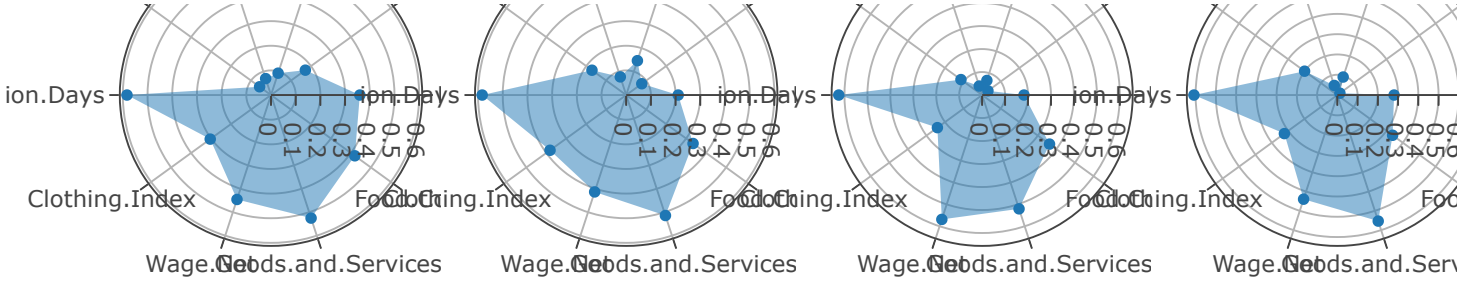


Dublin

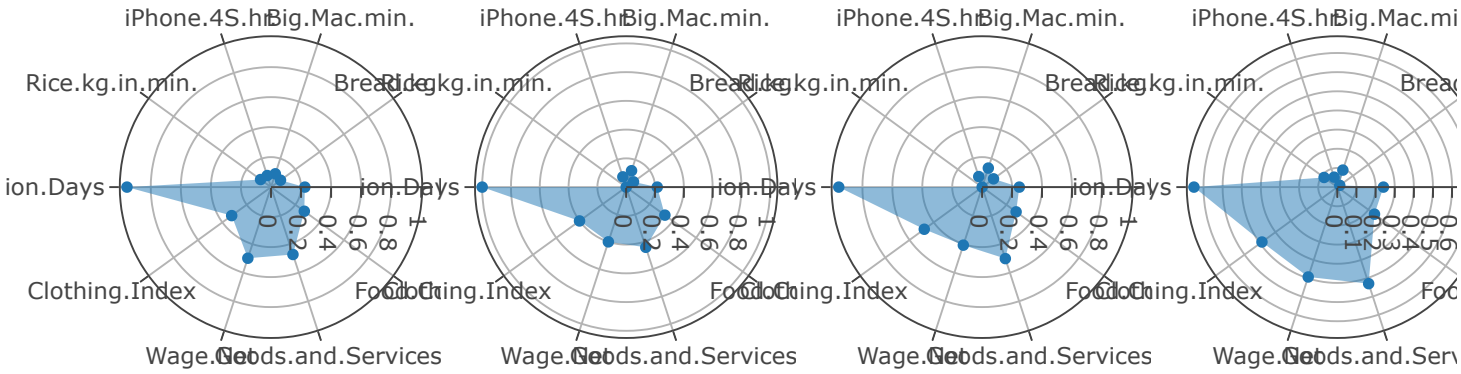


London

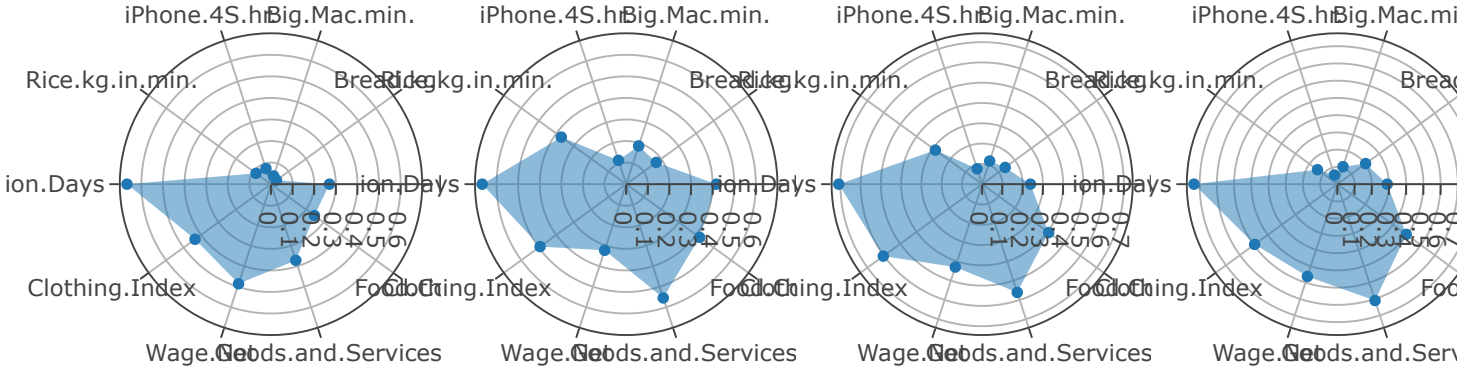




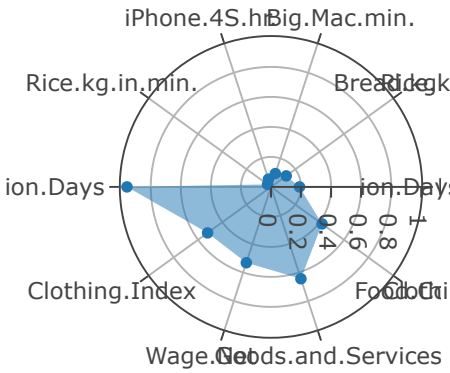
Berlin Madrid Barcelona Amsterdam



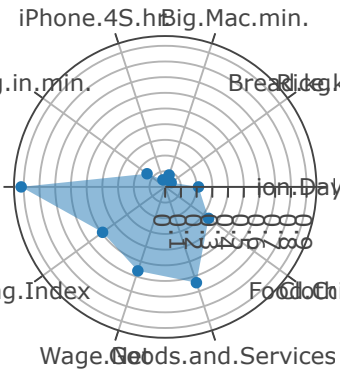
Nicosia Rome Milan Stockholm



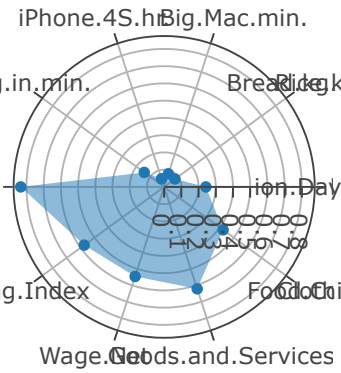
Helsinki



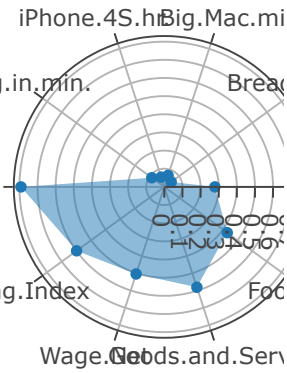
Frankfurt



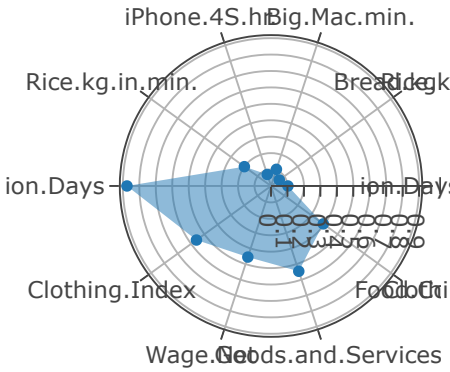
Munich



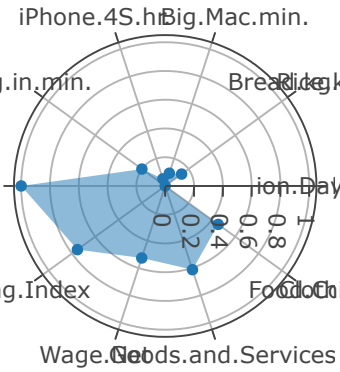
Vienna



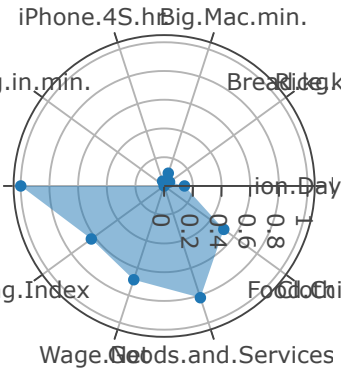
Lyon



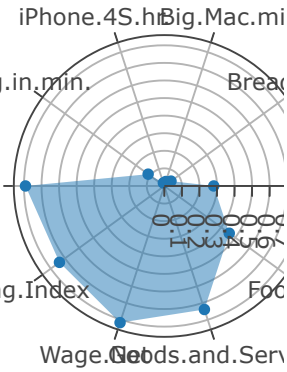
Paris



Copenhagen



Luxembourg

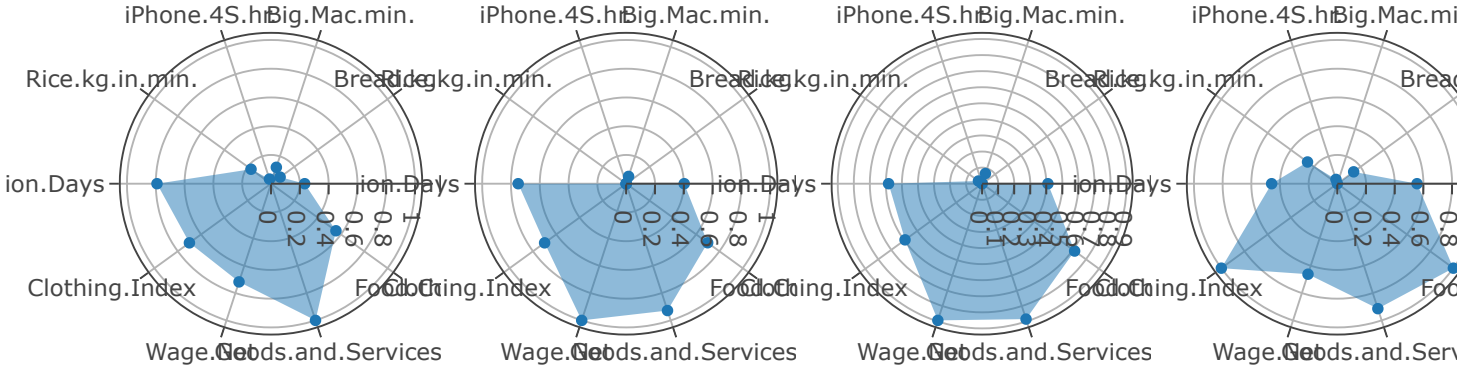


Oslo

Zurich

Geneva

Tokyo



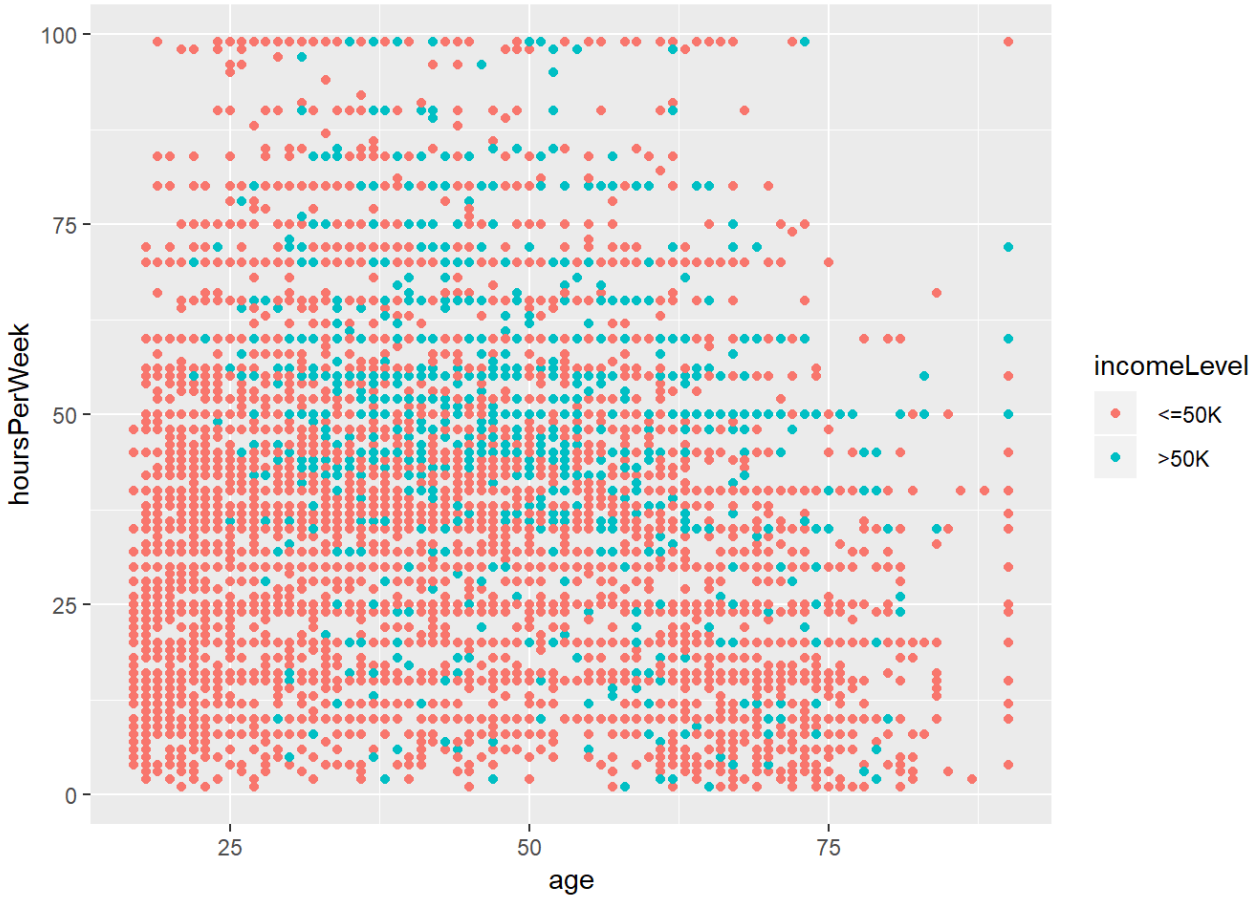
Cluster 1- Lisbon, Sao Paulo, Rio de Janeiro follows a similar pattern i.e. in Vacation days which is high in all the three cities and even other variables follows a similar trend. Cluster 2- Doha, Lima, Manama has similarity in terms of variables Vacation days and Hours worked and other variables also follow a similar trend. Caracas seems to be an outlier since it follows a pattern followed by no other city.

7

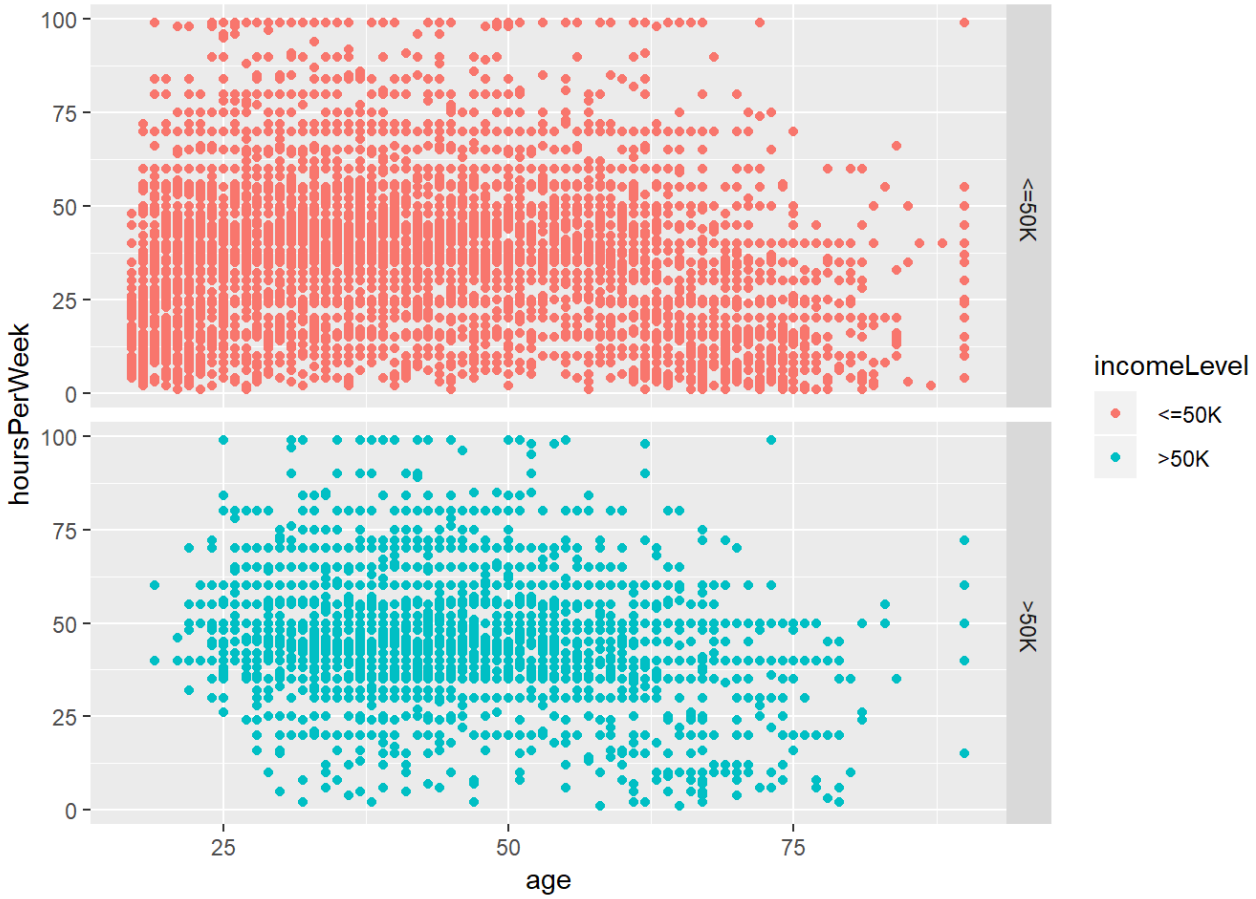
From the perspective of simplicity and to some extent efficiency we found Heatmaps was the best tool to analyze this dataset.

Assignment 2

```
### 1

library(ggplot2)

adult_data <- as.data.frame(read.csv("adult.csv"))
names(adult_data) <- c("age", "workClass", "populationIndex", "education",
                  "educationNum", "maritalStatus", "occupation", "relationship",
                  "race", "sex", "capitalGain", "capitalLoss", "hoursPerWeek", "country", "incomeLeve
l")
plot_2.1a <- ggplot(adult_data, aes(x= age, y= hoursPerWeek, colour= incomeLevel)) + geom_point()
plot_2.1a
```

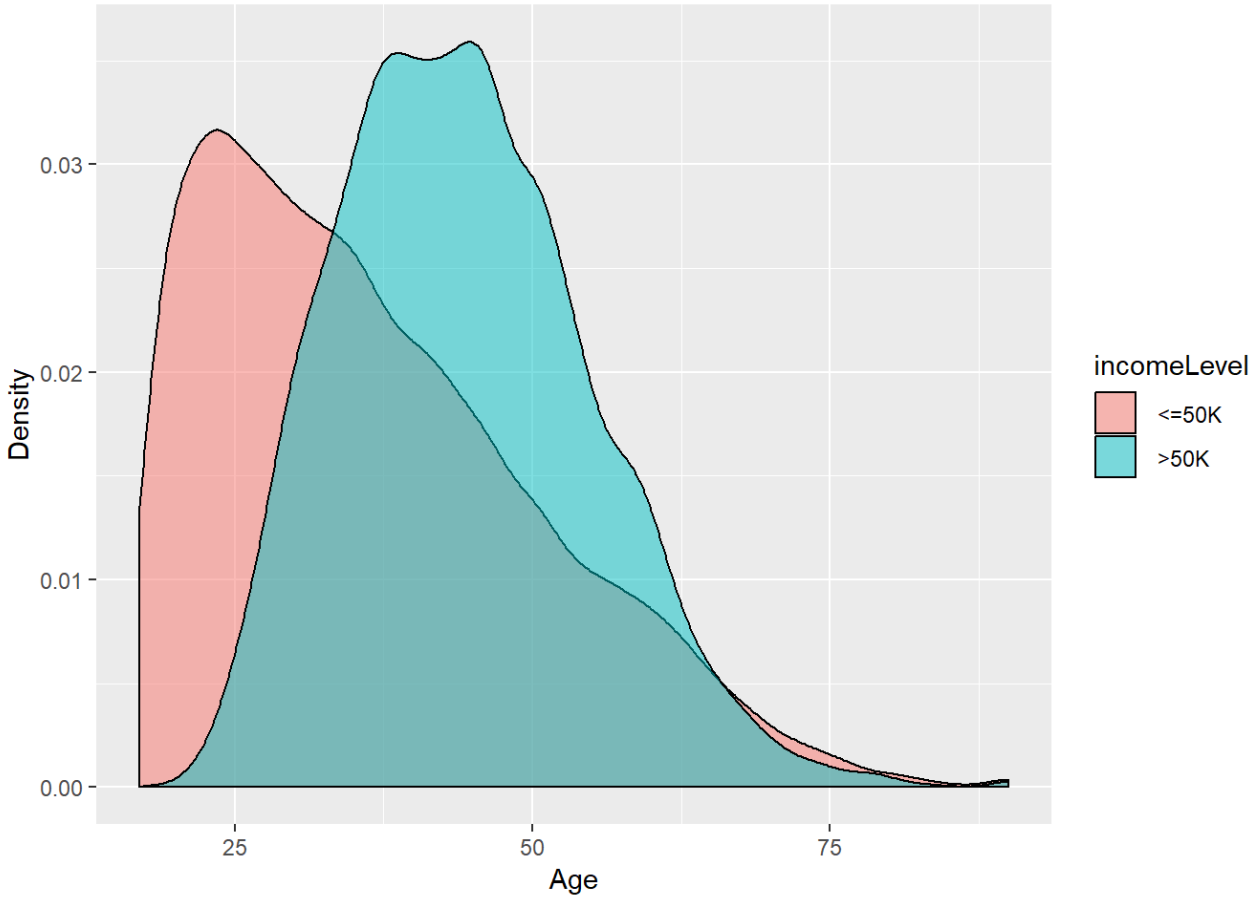



```
plot_2.1b <- plot_2.1a + facet_grid(incomeLevel~.)
plot_2.1b
```

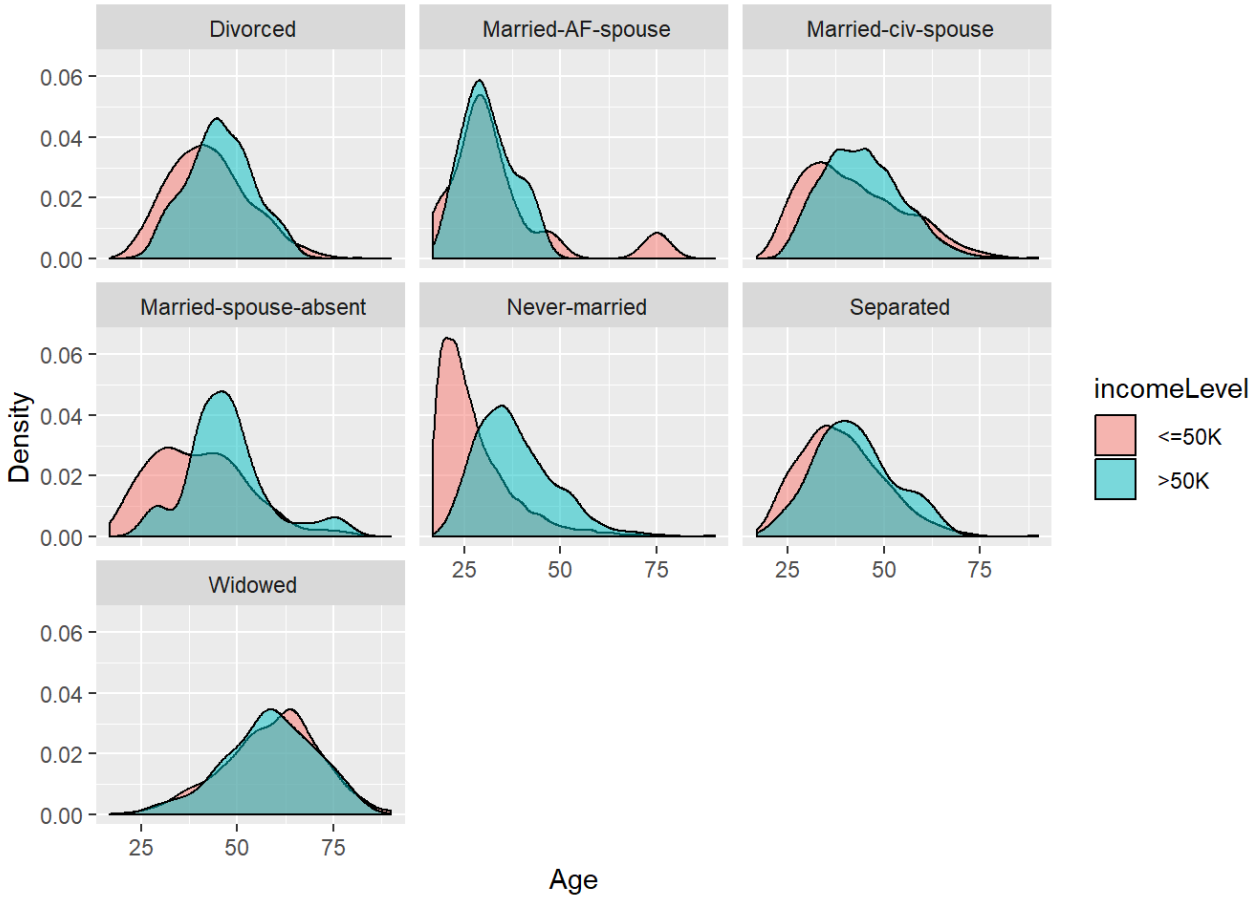


In the first plot the data points of both the catogories i.e. lesser or equal to 50k and more than 50k overlap each other heavily and it is hard to differentiate between data points of these two categories. In the second plot which is a Trellis plot we can see the actual number of data points and where the data points are denser which was not possible in the first plot. In the Trellis plot we can observe that the data in category ‘income level’ more than 50k from the age 25 to 60 and hours per week 25 to 60 is as denser as that of income level lesser than or equal to 50k. This was not easily evident in the first plot.

```
### 2

plot_2.2a <- ggplot()+
geom_density(data=adult_data, aes(x=age, group=incomeLevel, fill=incomeLevel),alpha=0.5) +
    xlab("Age") +
    ylab("Density")
plot_2.2a
```

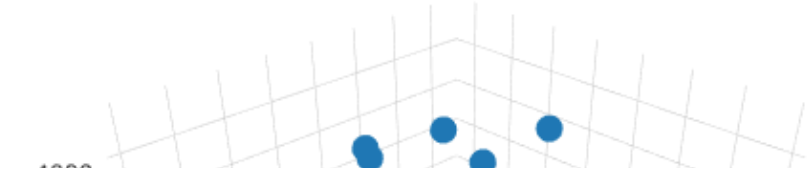


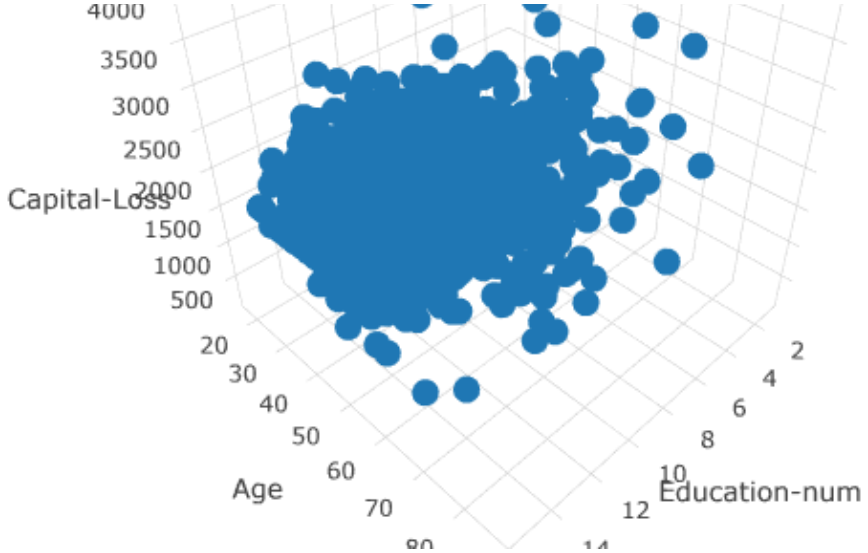

```
plot_2.2b <- plot_2.2a + facet_wrap(maritalStatus~.)
plot_2.2b
```



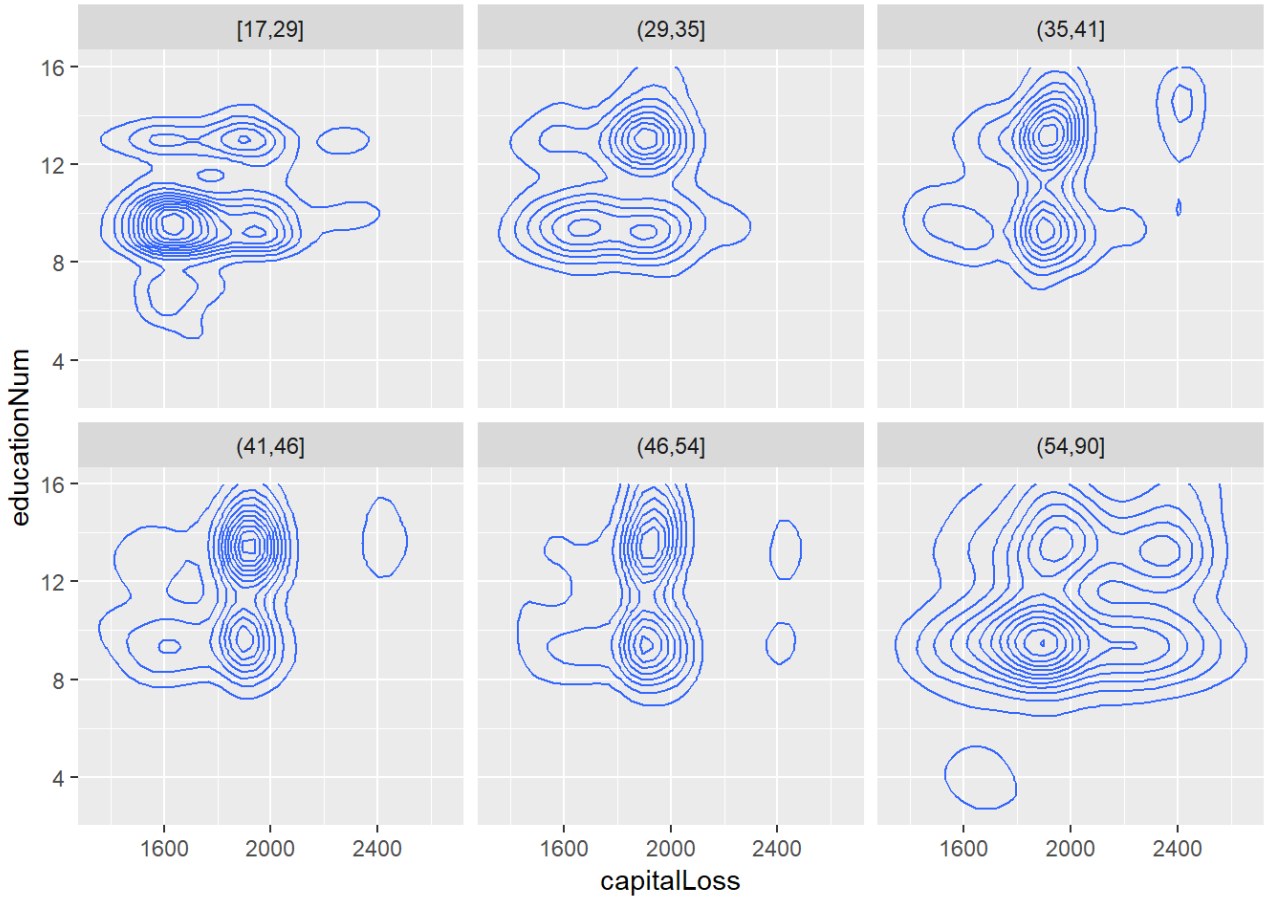
In the first plot we can see that more number of people, whose income level is lesser than or equal to 50k, are somewhere around the age of 22-25 and more number of people, whose income level is more than 50k, are somewhere around the age of 33-40. In the second plot which is a Trellis plot it is evident that people who never got married and age group which had the highest number of people whose income level was less than or equal to 50k was younger i.e. below the age of 25 years. For the plot containing Married-spouse-absent people,there were more amount of people between 35-50 years of age in the category who's income level was more than 50k. In all the other categories of marital status the number of people who's income level was less than or equal to and more than 50k fell under the same age group.

```
### 3

adult_fltdata <- adult_data[adult_data$capitalLoss >0,]

plot_2.3a <- plot_ly(adult_fltdata, x = ~educationNum, y = ~age, z = ~capitalLoss) %>%
  add_markers() %>%
  layout(scene = list(xaxis = list(title = 'Education-num'),
                      yaxis = list(title = 'Age'),
                      zaxis = list(title = 'Capital-Loss')))
plot_2.3a
```





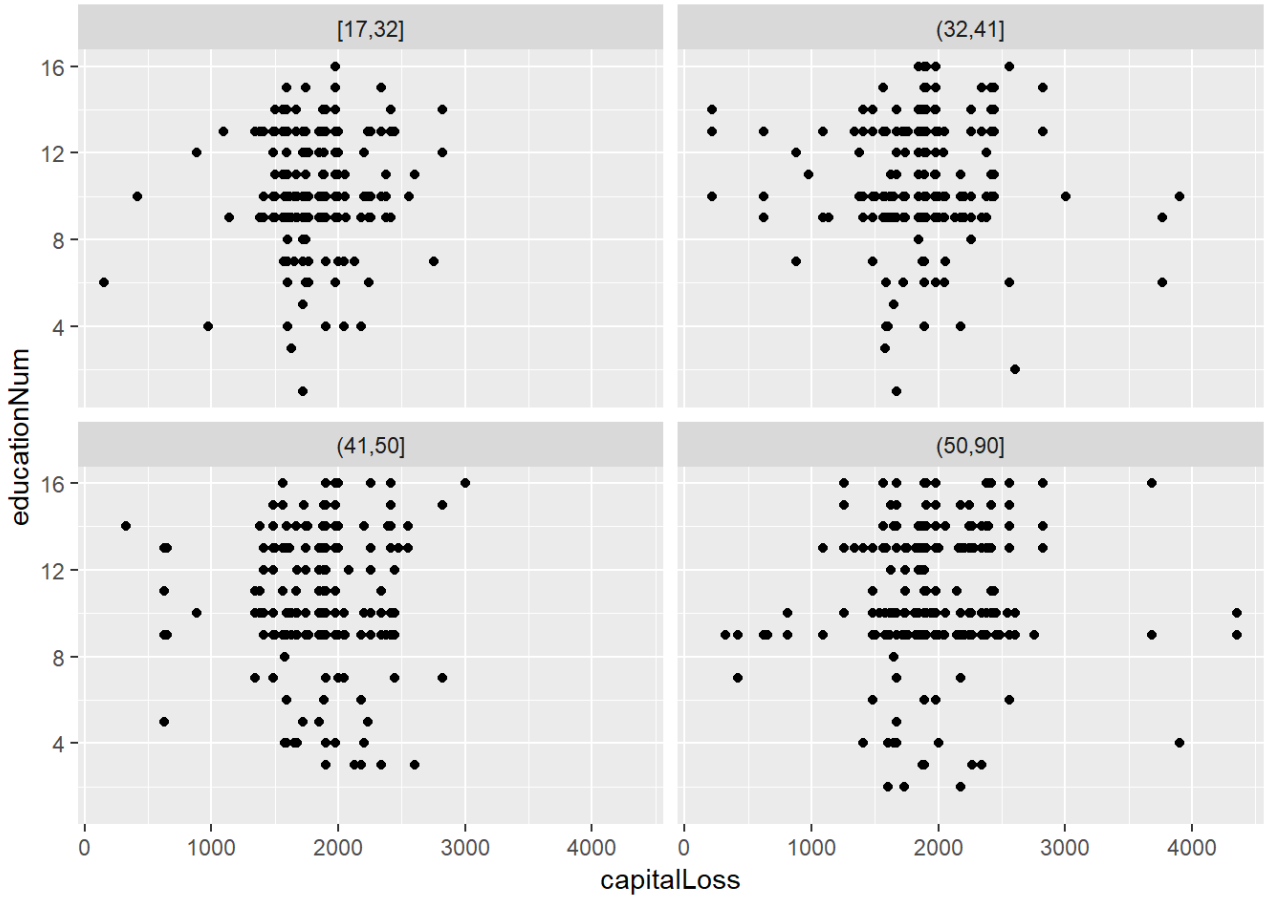

```
plot_2.3b <- ggplot()+
    geom_density2d(data = adult_fltdata, aes(x = capitalLoss, y = educationNum))+
    facet_wrap(cut_number(adult_fltdata$age, 6)~.)
plot_2.3b
```



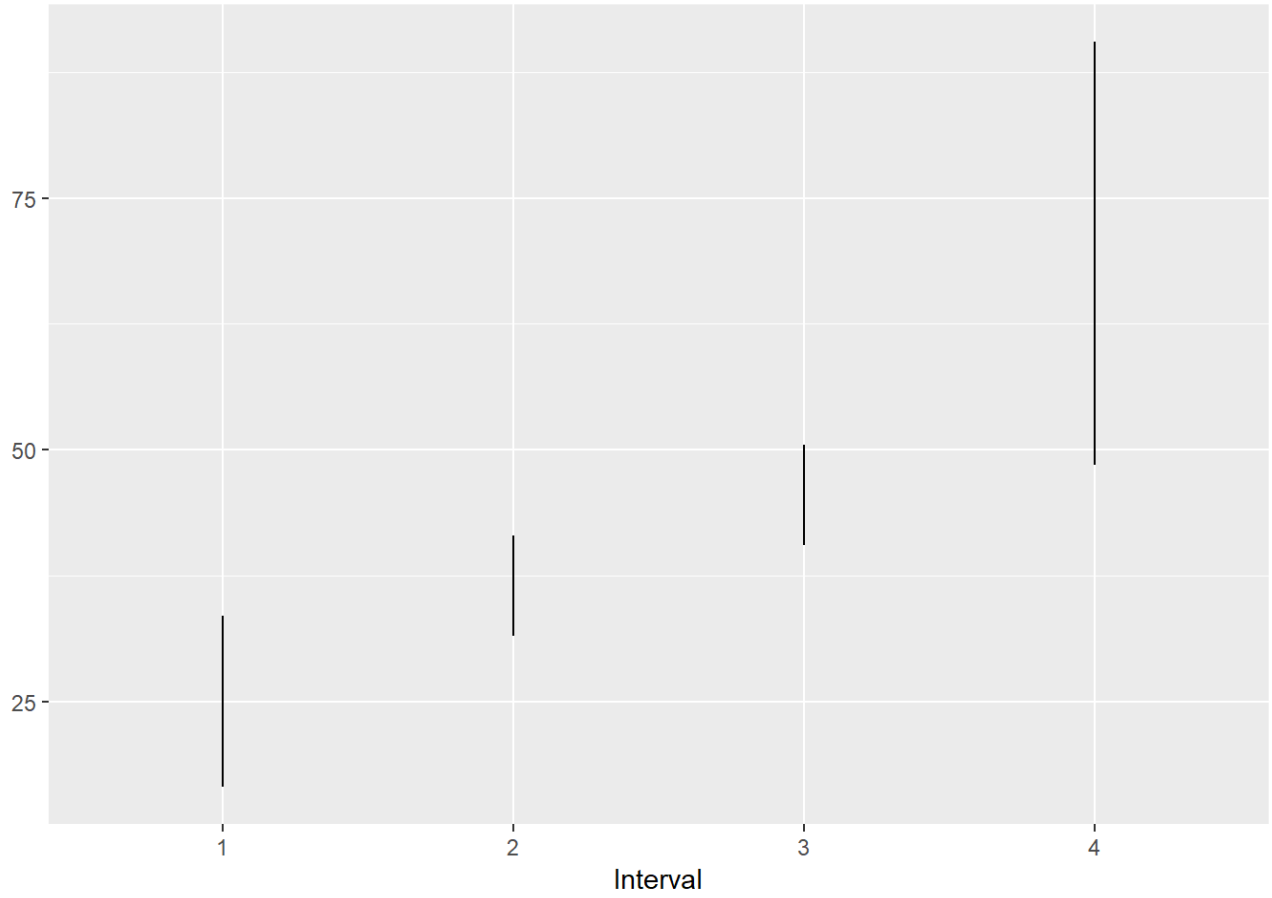
It is difficult to analyze the 3-D plot since all the data seems to be cluttered in the center. There also exists perception problem. The data points are pretty big in size and overlap each other heavily. In the second plot the capital loss seems to follow almost similar trend for all the age groups though the Capital loss for the age group 54-90 is more dispersed.

For different age groups variable educationNum also seems to follow a similar trend. There seems to be two groups for educationNum variable one close to 10 and one close to 14 for all age groups.

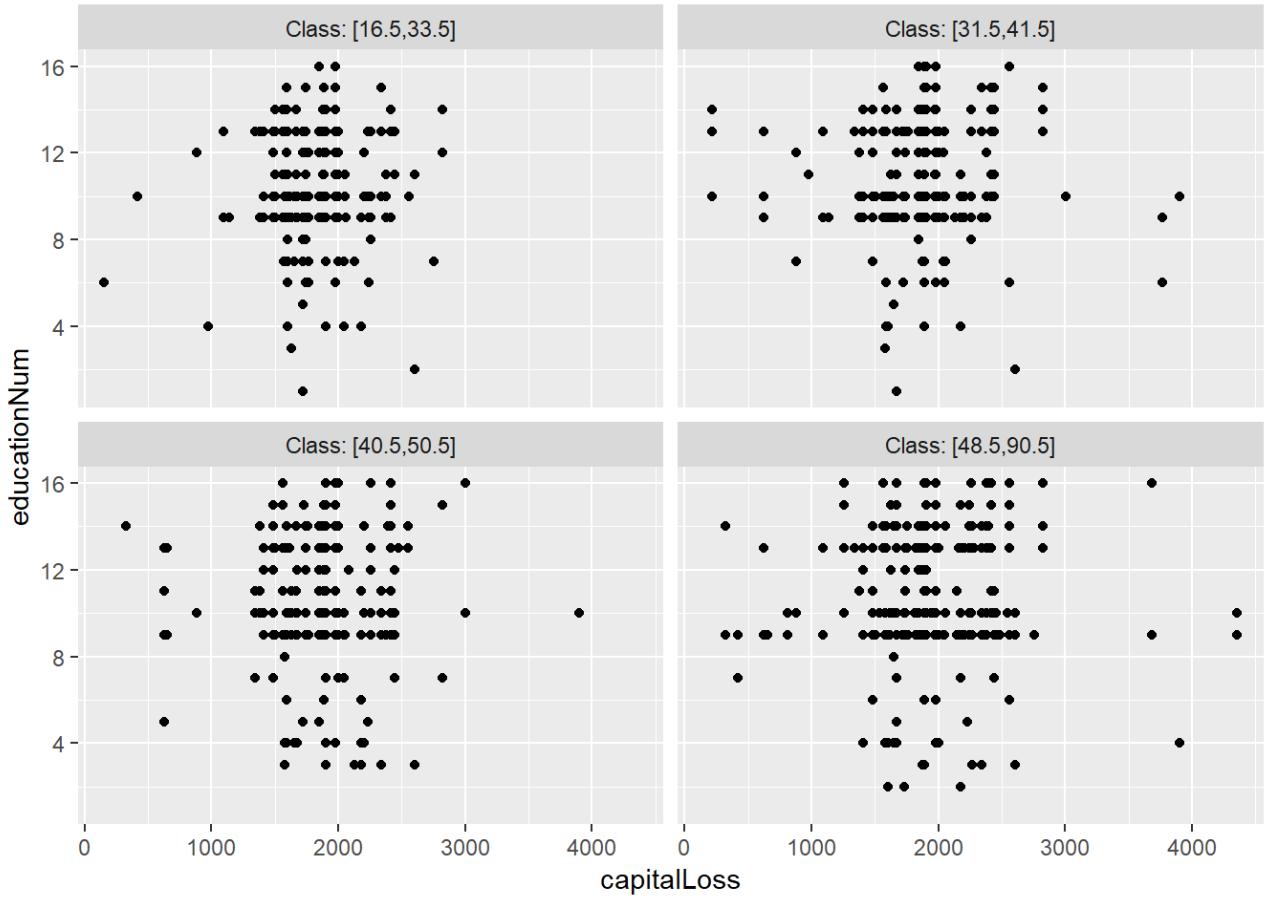
```
### 4

plot_2.4a <- ggplot()+
  geom_point(data = adult_fltdata, aes(x = capitalLoss, y = educationNum))+
  facet_wrap(cut_number(adult_fltdata$age, 4)~.)
plot_2.4a
```




```
Agerange <- lattice::equal.count(adult_fltdata$age, number=4, overlap=0.10)
L<-matrix(unlist(levels(Agerange)), ncol=2, byrow = T)
L1<-data.frame(Lower=L[,1],Upper=L[,2], Interval=factor(1:nrow(L)))
ggplot(L1)+geom_linerange(aes(ymin = Lower, ymax = Upper, x=Interval))
```




```
index=c()
Class=c()
for(i in 1:nrow(L)){
  Cl=paste("[", L1$Lower[i], ",", L1$Upper[i], "]", sep="")
  ind=which(adult_fltdata$age>=L1$Lower[i] &adult_fltdata$age<=L1$Upper[i])
  index=c(index,ind)
  Class=c(Class, rep(Cl, length(ind)))
}

df<-adult_fltdata[index,]
df$Class<-as.factor(Class)

ggplot(df, aes(x=capitalLoss, y=educationNum))+
  geom_point()+
  facet_wrap(~Class, labeller = "label_both")
```



Advantages- Using Shingles creates overlap of plots with respect to age in this case. Boundary effects can be avoided by using Shingles. As we can see a few data points from the preceding plot and succeeding plot will be included in the plot to avoid boundary effect. Disadvantages- More data points than that compared to a plot without using shingles, seem to appear in each of the plot which is a result of overlapping of data points.

Lab 5

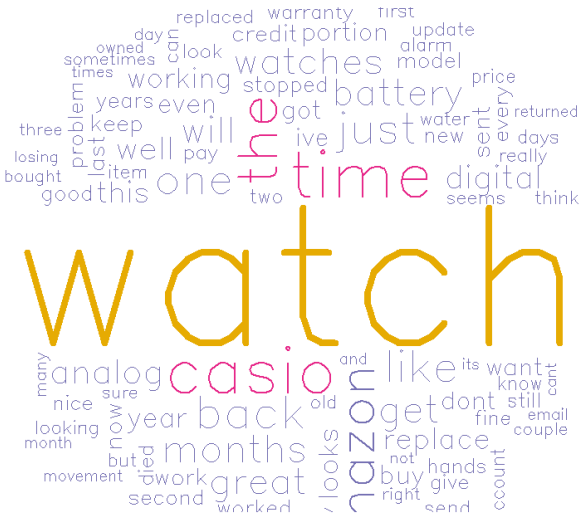
Group 19

Assignment 1

1.1

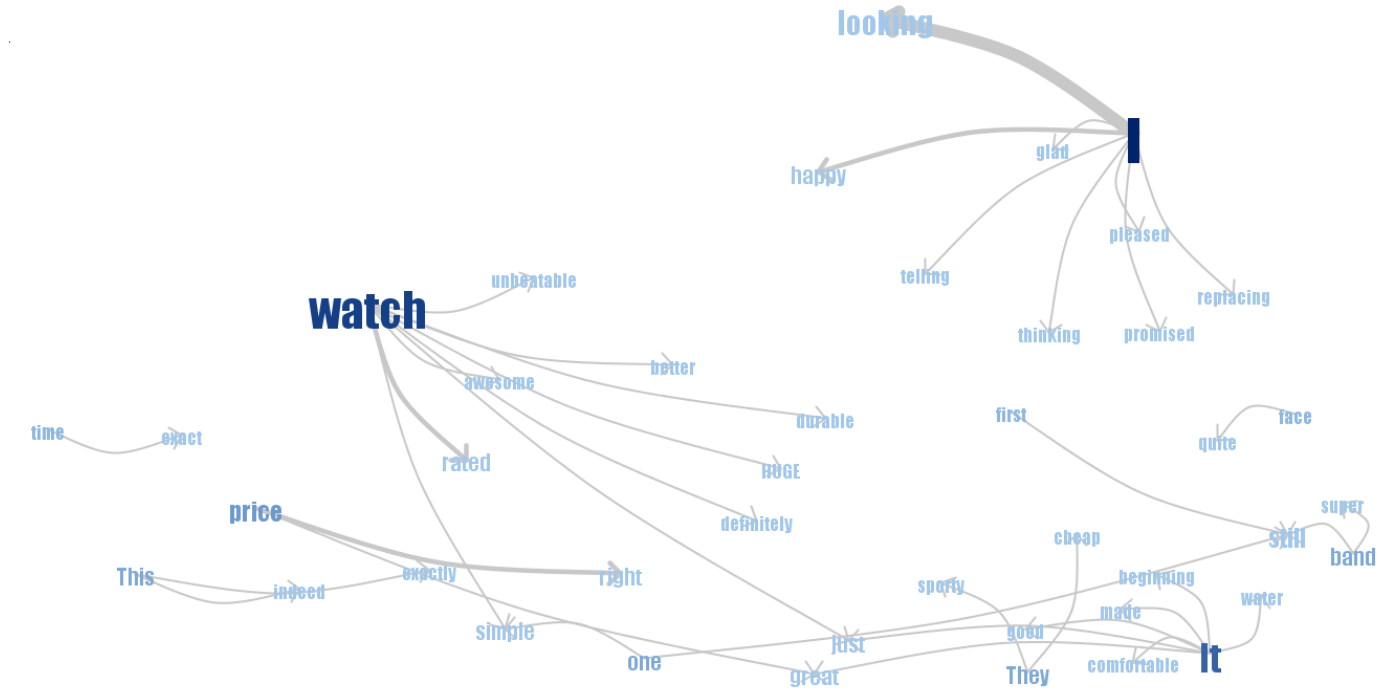
```
library(tm)
library(wordcloud)
library(RColorBrewer)

data_1<-read.table("Five.txt",header=F, sep='\n') #Read file
data_1$doc_id=1:nrow(data_1)
colnames(data_1)[1]<-"text"

#Here we interpret each line in the document as separate document
mycorpus_1 <- Corpus(DataframeSource(data_1)) #Creating corpus (collection of text data_1)
mycorpus_1 <- tm_map(mycorpus_1, removePunctuation)
mycorpus_1 <- tm_map(mycorpus_1, function(x) removeWords(x, stopwords("english")))
tdm_1 <- TermDocumentMatrix(mycorpus_1) #Creating term-document matrix
m_1 <- as.matrix(tdm_1)

#here we merge all rows
v_1 <- sort(rowSums(m_1),decreasing=TRUE) #Sum up the frequencies of each word
d_1 <- data.frame(word = names(v_1),freq=v_1) #Create one column=names, second=frequences
pal_1 <- brewer.pal(6,"Dark2")
pal_1 <- pal_1[-(1:2)] #Create palette of colors
wordcloud(d_1$word,d_1$freq, scale=c(8,.3),min.freq=2,max.words=100, random.order=F, rot.per=.15, colors=pal_1, vfont
=c("sans serif","plain"))
```

```
data_2<-read.table("OneTwo.txt",header=F, sep='\n') #Read file
data_2$doc_id=1:nrow(data_2)
colnames(data_2)[1]<-"text"

#Here we interpret each line in the document as separate document
mycorpus_2 <- Corpus(DataframeSource(data_2)) #Creating corpus (collection of text data_1)
mycorpus_2 <- tm_map(mycorpus_2, removePunctuation)
mycorpus_2 <- tm_map(mycorpus_2, function(x) removeWords(x, stopwords("english")))
tdm_2 <- TermDocumentMatrix(mycorpus_2) #Creating term-document matrix
m_2 <- as.matrix(tdm_2)

#here we merge all rows
v_2 <- sort(rowSums(m_2),decreasing=TRUE) #Sum up the frequencies of each word
d_2 <- data.frame(word = names(v_2),freq=v_2) #Create one column=names, second=frequences
pal_2 <- brewer.pal(6,"Dark2")
pal_2 <- pal_2[-(1:2)] #Create palette of colors
wordcloud(d_2$word,d_2$freq, scale=c(8,.3),min.freq=2,max.words=100, random.order=F, rot.per=.15, colors=pal_2, vfont
=c("sans serif","plain"))
```

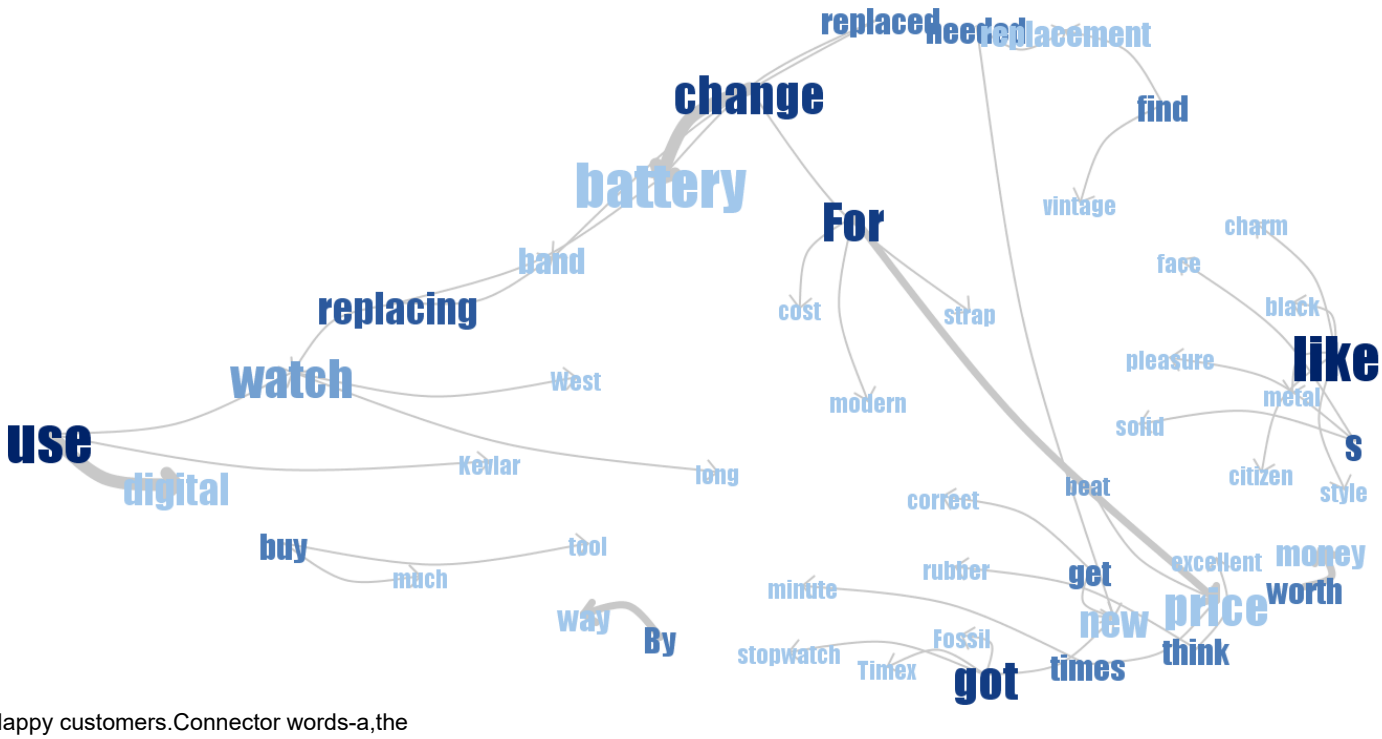


First wordcloud corresponds to feedbacks of people who are pleased with the product. We can see that certain words are bigger than others which means they appear more number of times than other words in the wordcloud.By looking at the words we can get an idea about what features or aspects of the watch which pleased the customers. Here we see that the following features of the watch were found to be good by the customers-Face,looks,price,band,dial. We can also guess the features of the watch by looking at the words like dual,digital,analog,alarm,appearence,quality,water resistant,stopwatch,durability etc. Second wordcloud corresponds to feedbacks of people who are not pleased with the product. Here we see that the following features of the watch were found to be bad by the customers-Analog,battery,replacement.

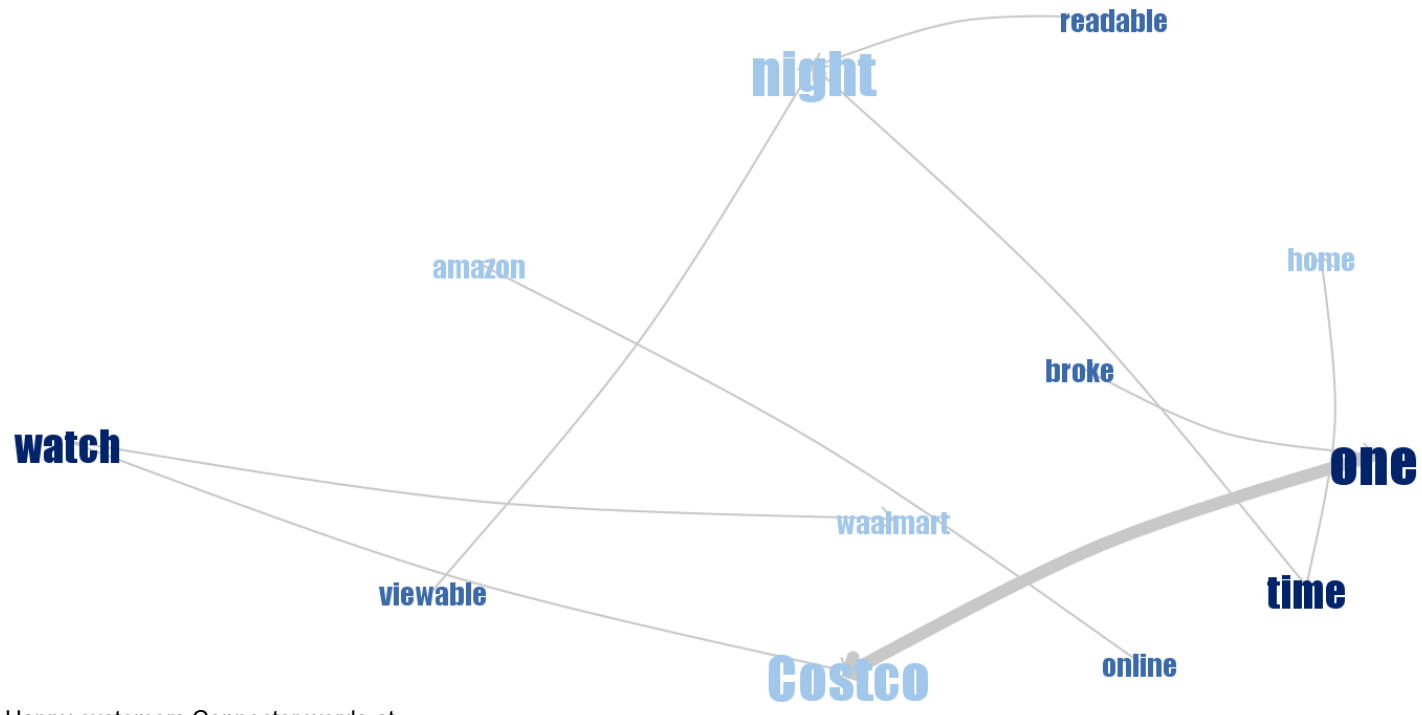
1.2



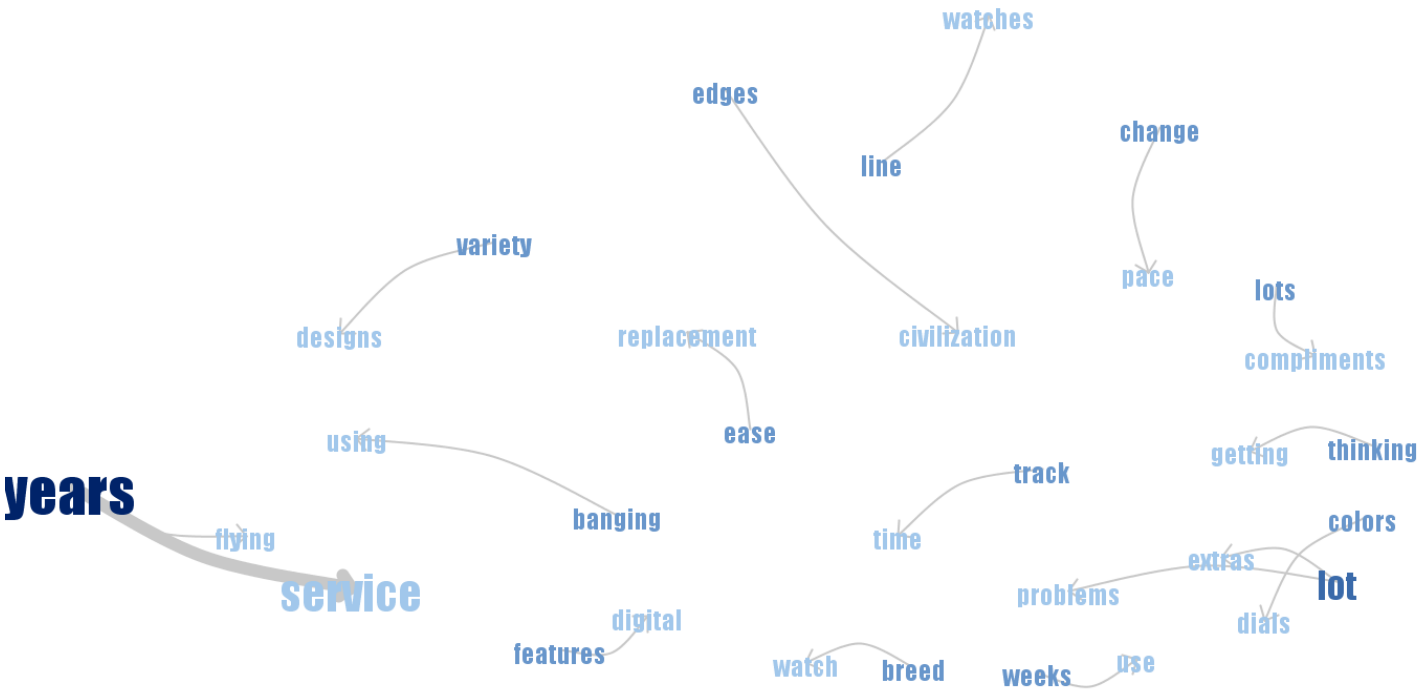
Happy customers.Connector words-am,is,are,was,were



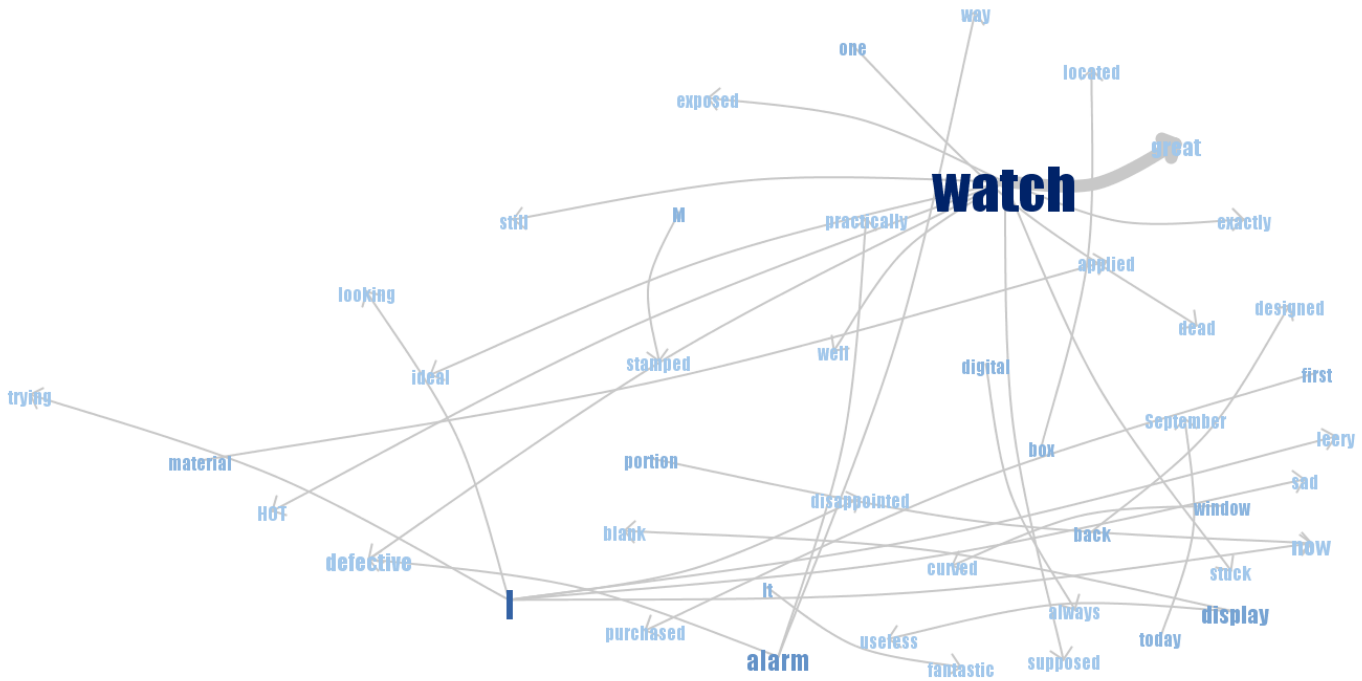
Happy customers.Connector words-a,the



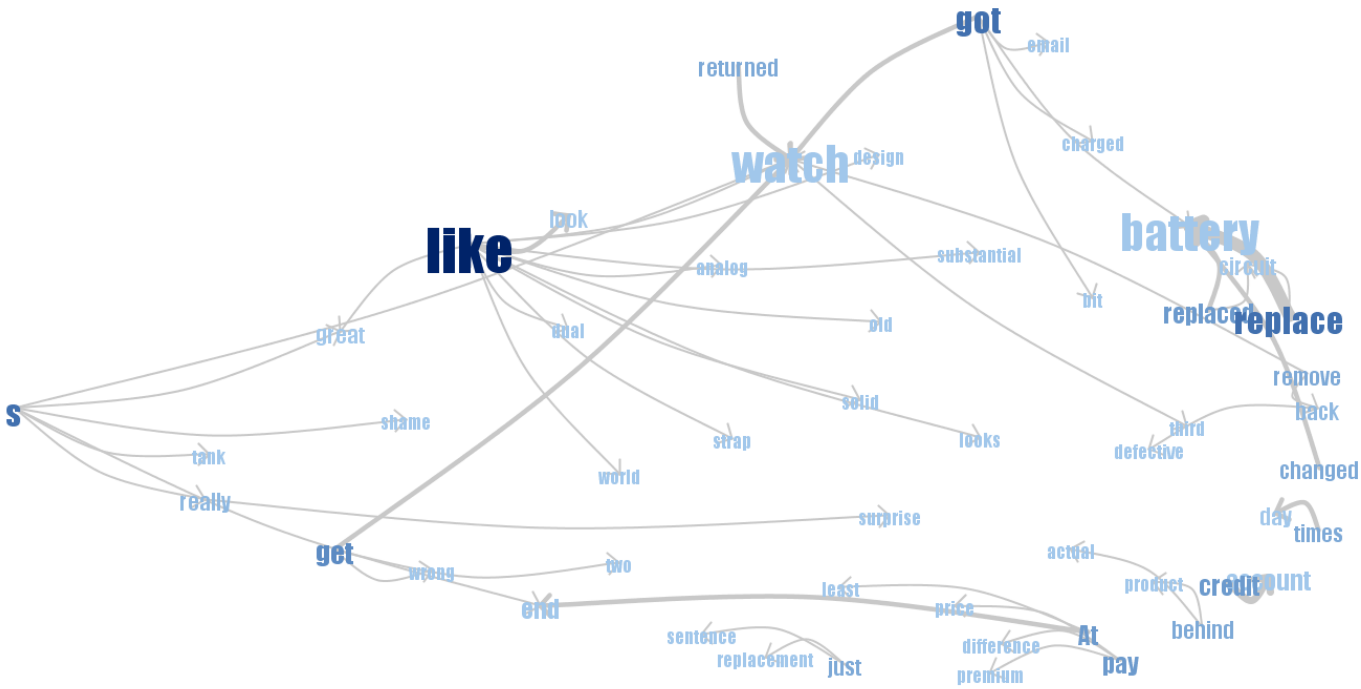
Happy customers.Connector words-at



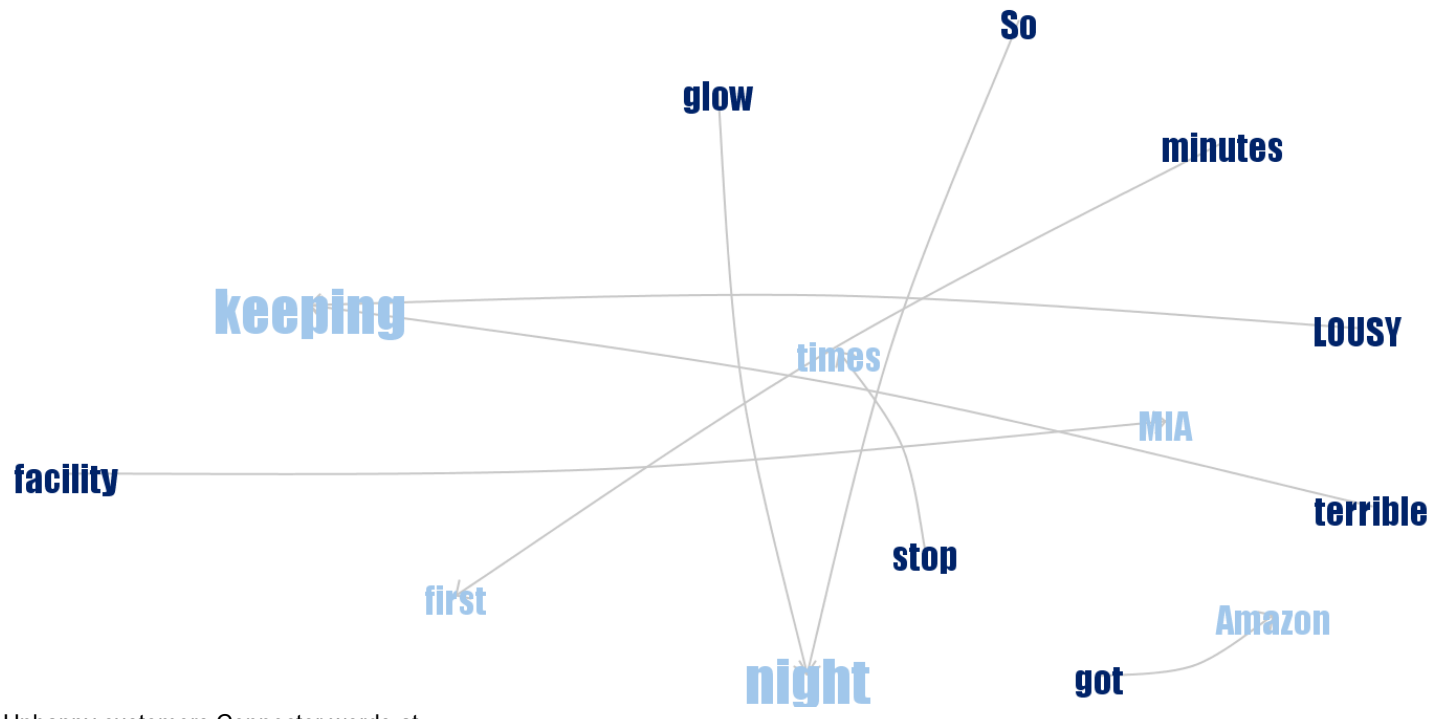
Happy customers.Connector words-of



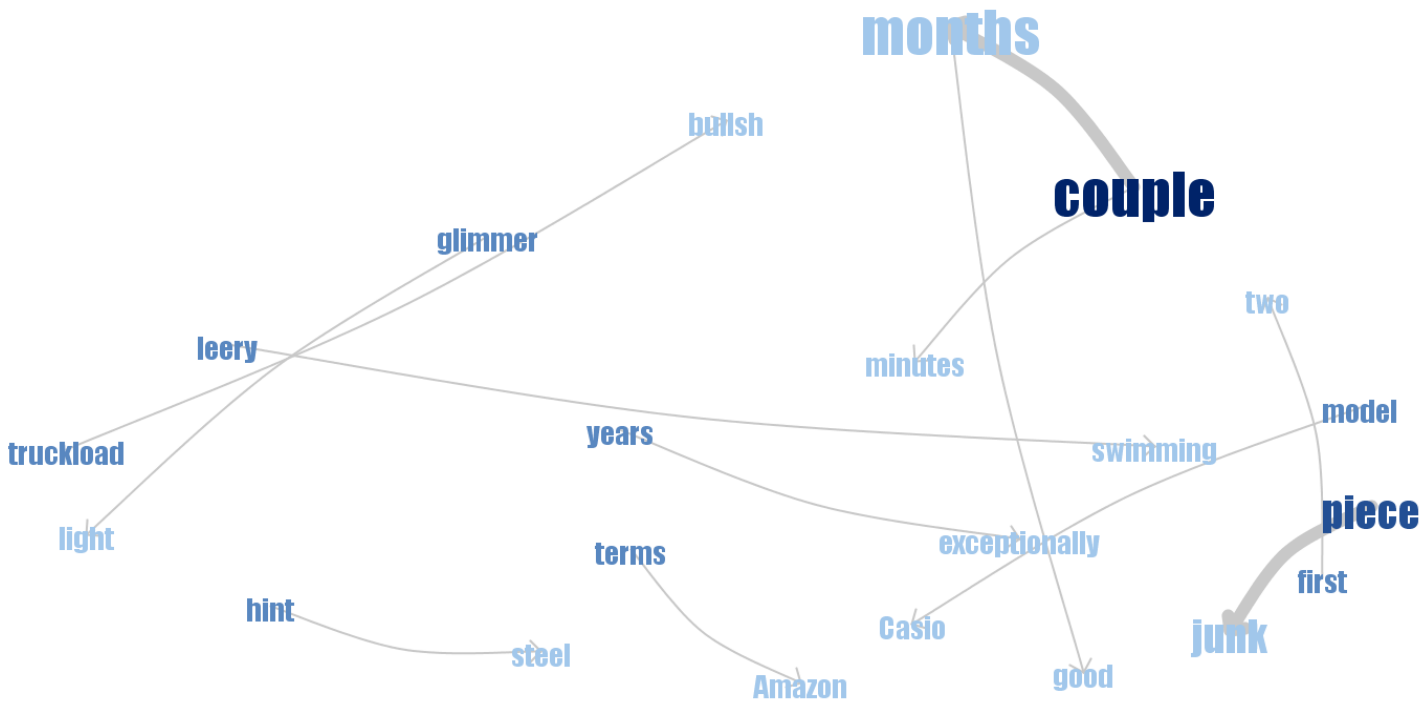
Unhappy customers.Connector words-am,is,are,was,were



Unhappy customers.Connector words-a,the



Unhappy customers.Connector words-at



Unhappy customers.Connector words-of

Properties of this watch that are often mentioned

Appearence- Black, Gold and White face

Quality

cost

Display features- Analog and Digital display

Luminousity

Satisified customers are talking about

Good appearence

Simple and sporty look

Durability

Cheap pricing

Comfortable to wear

Durable watch band

Availability of different types of bands like Velcro band and metal band

Water resistant

Easy to read face

Toughness of the watch

Unsatisified customers are talking about

Low luminosity

Defective alarm

Buttons get broken and defective easily

Chronometer does not work properly

Low alarm sound

Analog display is defective

Tough to get the defective product replaced

Bad at keeping accurate time

Doesnt work when immersed deep in water

Good and Bad properties

Good properties

```
Appearence due to face and size

Cheap

Water resistant

Sporty look

Tough

Durable
```

Bad properties

```
Luminousity

Alarm

Bad quality buttons

Accuracy of time of Analog display

Replacement
```

Characteristics of the watch

Display- Digital and analog display

Luminous display

Face- Black,Gold or White

Water resitant

Sporty look

Velcro nylon or metal band

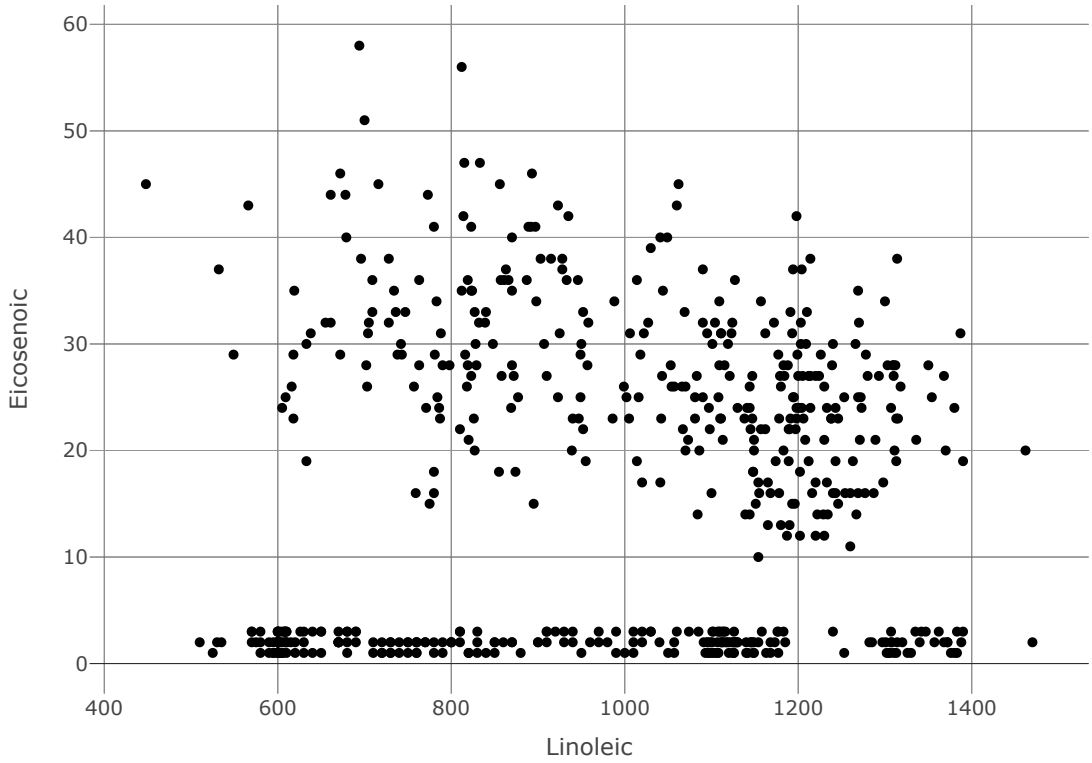
Chronometer

Assignment 2

```
library(plotly)
library(crosstalk)
library(tidyr)
library(ggplot2)
library(GGally)

olive <- read.csv("olive.csv")
d <- SharedData$new(olive)
```

2.1

```
scatterOlive <- plot_ly(d, x = ~linoleic, y = ~eicosenoic) %>%
  add_markers(color = I("black")) %>%
  layout(xaxis=list(title="Linoleic"), yaxis=list(title="Eicosenoic"))
scatterOlive
```



The values at the bottom of the plot have Eicosenoic values between one and three. There are 249 data points in the bottom part of the plot.

2.2

```
barOlive <- plot_ly(d, x=~as.factor(Region)) %>% add_histogram() %>%
  layout(barmode="overlay",xaxis=list(title="Region"))

bscols(widths=c(2, NA),filter_slider("stearic", "values of stearic", d, ~stearic)
       ,subplot(scatterOlive,barOlive,titleY = TRUE,  titleX = TRUE)%>%
         highlight(on="plotly_select", dynamic=T, persistent = T, opacityDim = I(1))%>%hide_legend())
```

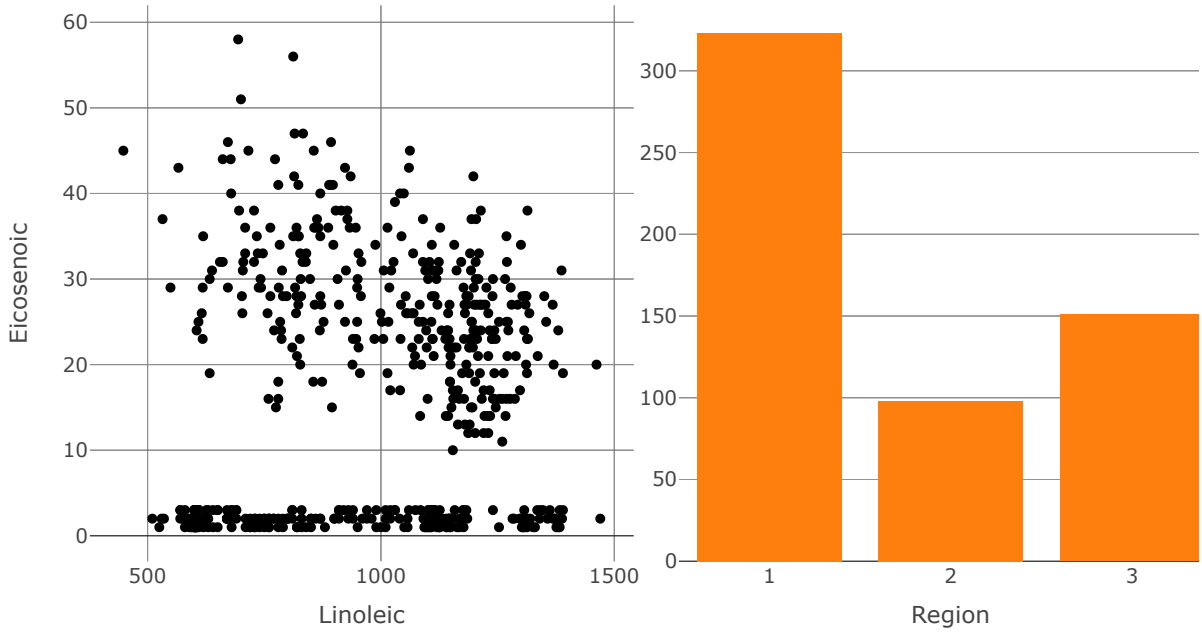
values of stearic

152 375

152 200 248 296 344

Brush color

rgba(228,26,



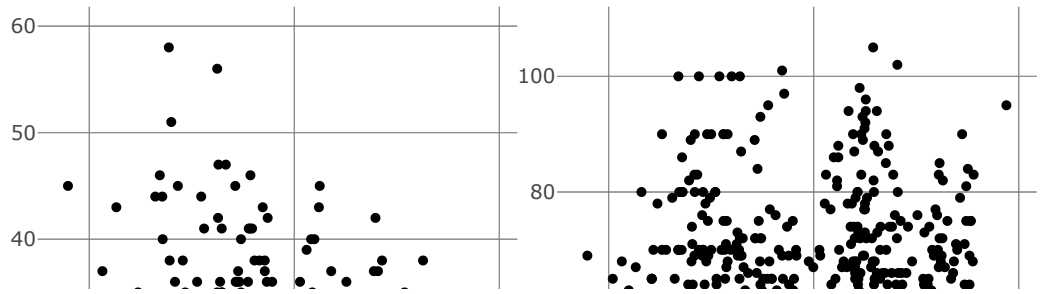
All the observations which have eicosenoic values less then three are from the region two and three. Region two have linoleic value above 1050 and region three have below 1050.By using the slider it is found that most of the data in region 2 and 3 is between stearic values 180 to 280.We are using three interaction operators i.e. filtering operator for filtering stearic values, selection operator to select low values of Eicosenoic and connection operator to link the above plots together.

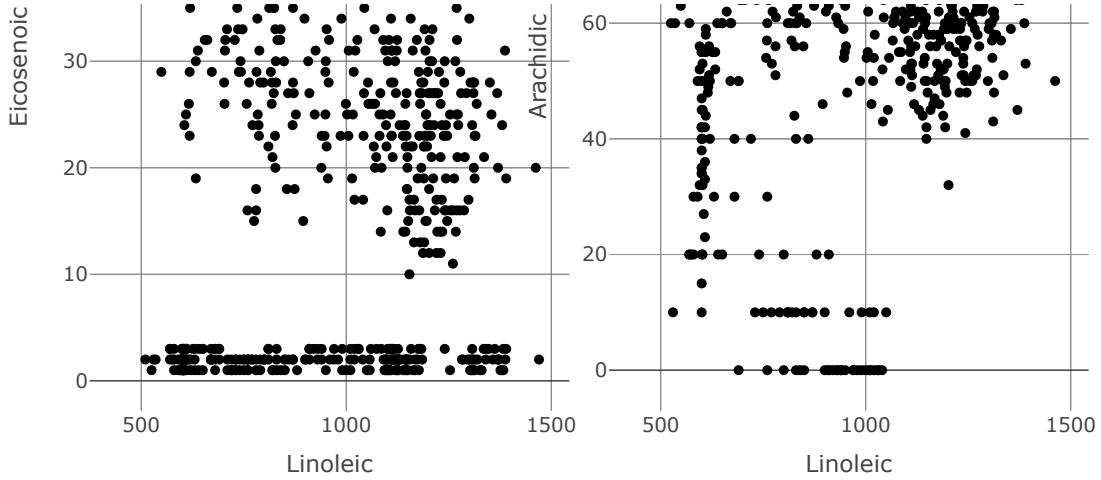
2.3

```
scatterOlive_2 <- plot_ly(d, x = ~linoleic, y = ~arachidic) %>%
  add_markers(color = I("black")) %>%
  layout(xaxis=list(title="Linoleic"), yaxis=list(title="Arachidic"))

subplot(scatterOlive,scatterOlive_2,  titleY = TRUE,  titleX = TRUE)%>%
  highlight(on="plotly_select", dynamic=T, persistent = T, opacityDim = I(1))%>%hide_legend()
```

Brush color

rgba(228,26,





The Arachidic values which are less then 40 and less then 1050 on Linoleic scale are also at the bottom of the Eicosenoic VS Linoleic plot which are the outliers and they are grouped. Also two points which are greater then 1400 in Linoleic scale are also outliers in both plots which are not grouped.

2.4

```
p<-ggparcoord(olive, columns = c(4:11))
d<-plotly_data(ggplotly(p))%>%group_by(.ID)
d1<-SharedData$new(d, ~.ID, group="olive")
p1<-plot_ly(d1, x=~variable, y=~value)%>%add_lines(line=list(width=0.3))%>%
  add_markers(marker=list(size=0.3),text=~.ID, hoverinfo="text")

olive2=olive
olive2$.ID=1:nrow(olive)
d2<-SharedData$new(olive2, ~.ID, group="olive")
p2<-plot_ly(d2, x=~factor(Region) )%>%add_histogram()%>%layout(title = "Region",barmode="overlay")

ButtonsX=list()
for (i in 4:11){
  ButtonsX[[i-3]]= list(method = "restyle",
                        args = list( "x", list(olive[[i]])),
                        label = colnames(olive)[i])
}
ButtonsY=list()
for (i in 4:11){
  ButtonsY[[i-3]]= list(method = "restyle",
                        args = list( "y", list(olive[[i]])),
                        label = colnames(olive)[i])
}
ButtonsZ=list()
for (i in 4:11){
  ButtonsZ[[i-3]]= list(method = "restyle",
                        args = list( "z", list(olive[[i]])),
                        label = colnames(olive)[i])
}


p3 <- plot_ly(d2, x=~palmitic, y=~stearic, z=~oleic, alpha = 0.8) %>%
  add_markers() %>%
  layout(xaxis=list(title=""), yaxis=list(title=""), zaxis=list(title="")),
         title = "Select variable:",
         updatemenus = list(
           list(y=1.00, buttons = ButtonsX),
           list(y=0.85, buttons = ButtonsY),
           list(y=0.70, buttons = ButtonsZ)
         )  )

bscols(p1%>%highlight(on="plotly_select", dynamic=T, persistent = T, opacityDim = I(1))%>%
         hide_legend(),
       p3%>%highlight(on="plotly_click", dynamic=T, persistent = T)%>%hide_legend(),
       p2%>%highlight(on="plotly_click", dynamic=T, persistent = T)%>%hide_legend())
```

Brush color

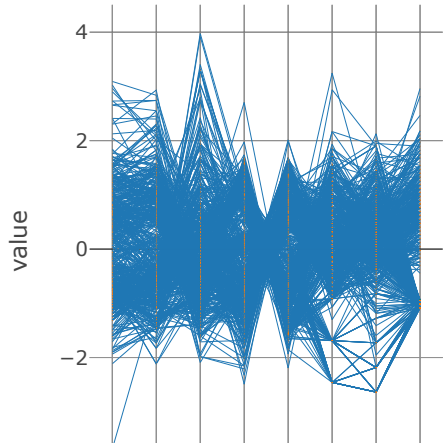
rgba(228,26,

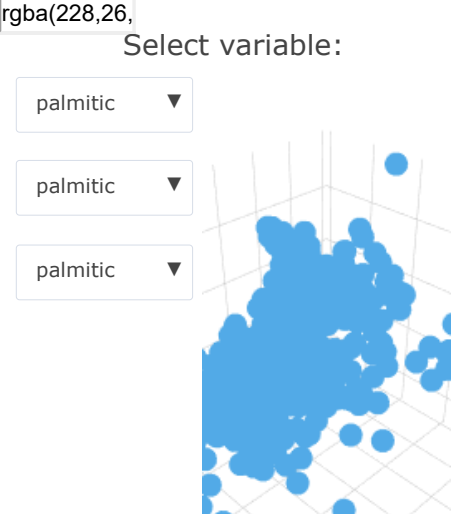
Brush color

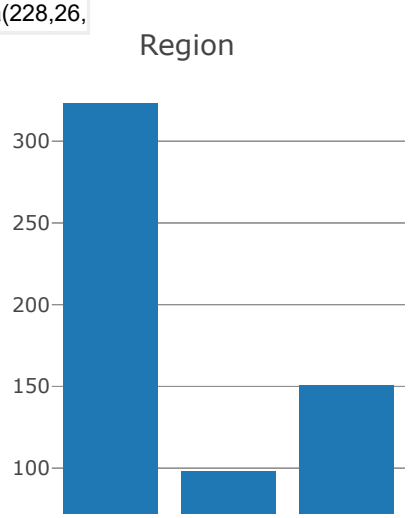
rgba(228,26,

Brush color

rgba(228,26,







After brushing the different regions in the bar plot with different colours it was evident that region 1 can be defined using eicosenoic values. Linoliec values can be used to differentiate between region 2 and 3. All the linoliec values of region 2 are higher than that of region 3. Oleic values servers the same purpose. So these 3 variables can be used to define a region and can be considered influential variables. After brusing the different regions in diferent coulours finding clusters in the parallel co-ordinate plot beacame easier. There are 2 clusters in region 2 that can be seen when variable linoliec is observed. By selecting the influential variables in the dropboxesit can be seen that infact each region corresponds to one cluster.

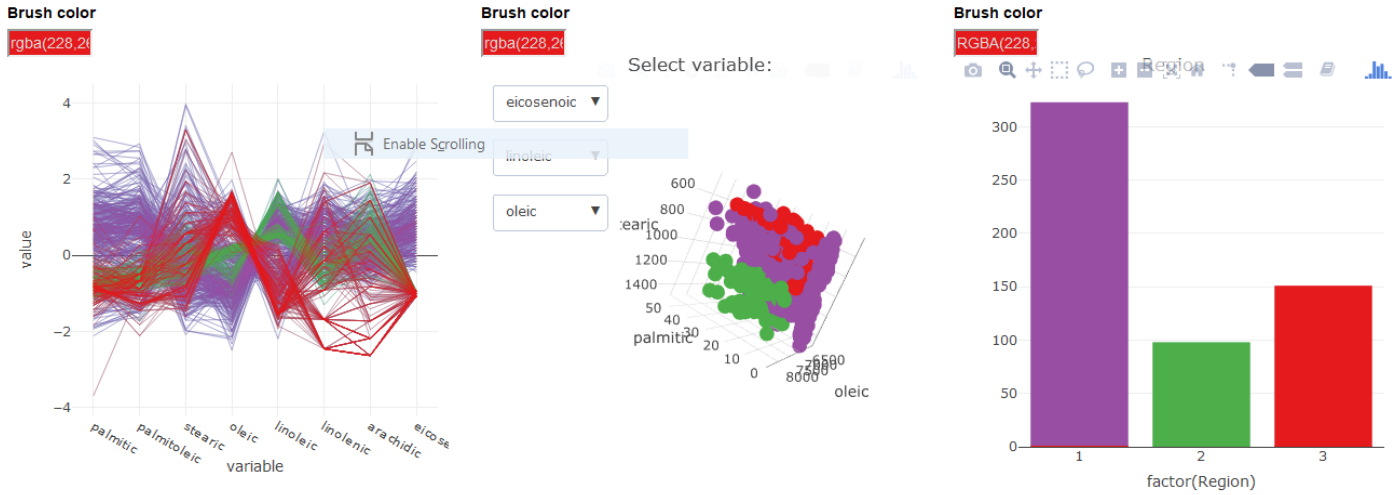


Fig 1

2.5

The interaction operators used in the above plot are:

Selection

Connection

Navigation

Filtering operators can be used by providing drop boxes for parallel co-ordinate plot like in the case of 3d scatter plot.

Visualization Lab-6

fahha780,vinbe289

October 16, 2018

Assignment 1

```
library(tourr)
```

```
## Warning: package 'tourr' was built under R version 3.5.1
```

```
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 3.5.1
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.5.1
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(igraph)
```

```
## Warning: package 'igraph' was built under R version 3.5.1
```

```
##
## Attaching package: 'igraph'
```

```
## The following object is masked from 'package:plotly':
##
##     groups
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

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

```
library(visNetwork)
```

```
## Warning: package 'visNetwork' was built under R version 3.5.1
```

```
library(seriation)
```

```
## Warning: package 'seriation' was built under R version 3.5.1
```

```
##
## Attaching package: 'seriation'
```

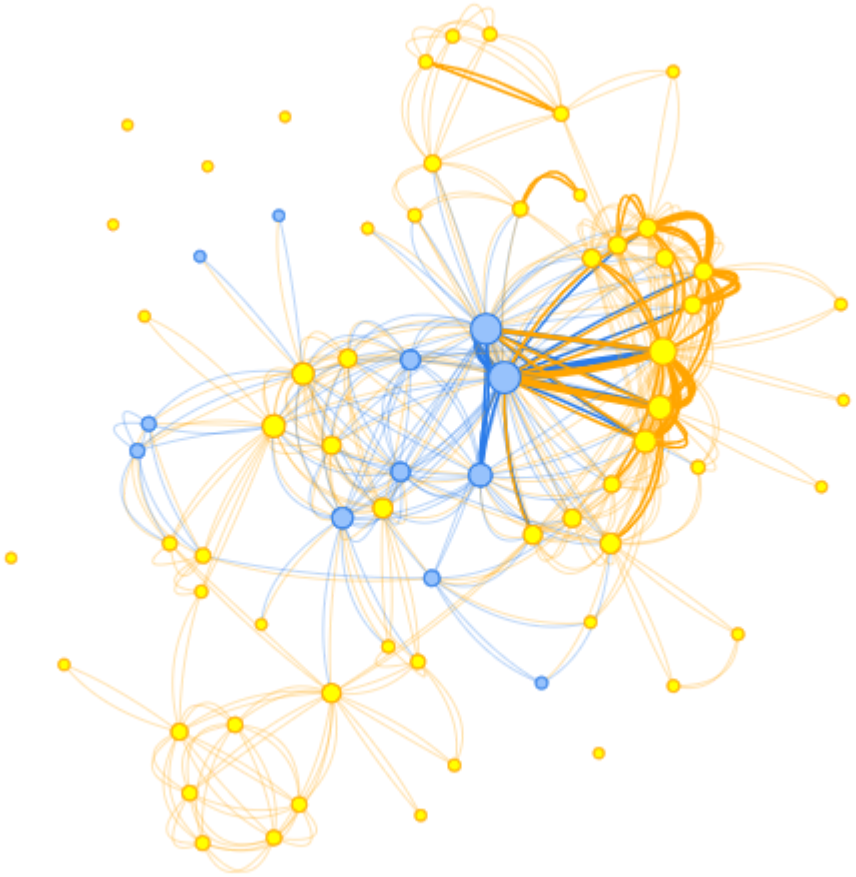
```
## The following object is masked from 'package:igraph':
##
##     permute
```

```
## The following object is masked from 'package:tourr':
##
##     path_dist
```

1

```
nodes <- read.delim("trainMeta.dat",sep = " ",header = FALSE)
colnames(nodes) <- c("label","group")
nodes$title <- nodes$label
nodes$id <- 1:dim(nodes)[1]
nodes2 <- nodes[,c(4,2,1,3)]

edges <- read.delim("trainData.dat", sep = " ")
edges <- edges[,c(2,3,4)]
colnames(edges) <- c("from","to","value")

graph <- graph_from_data_frame(d=edges, vertices=nodes2, directed=T)
nodes2$value <- strength(graph)

Network <- visNetwork(nodes2, edges)%>%
  visPhysics(solver='repulsion') %>%
  visOptions(highlightNearest = list(enabled = TRUE, degree = 1,
                                     labelOnly = FALSE, hover = TRUE),
             nodesIdSelection = TRUE)
Network
```

Select by id



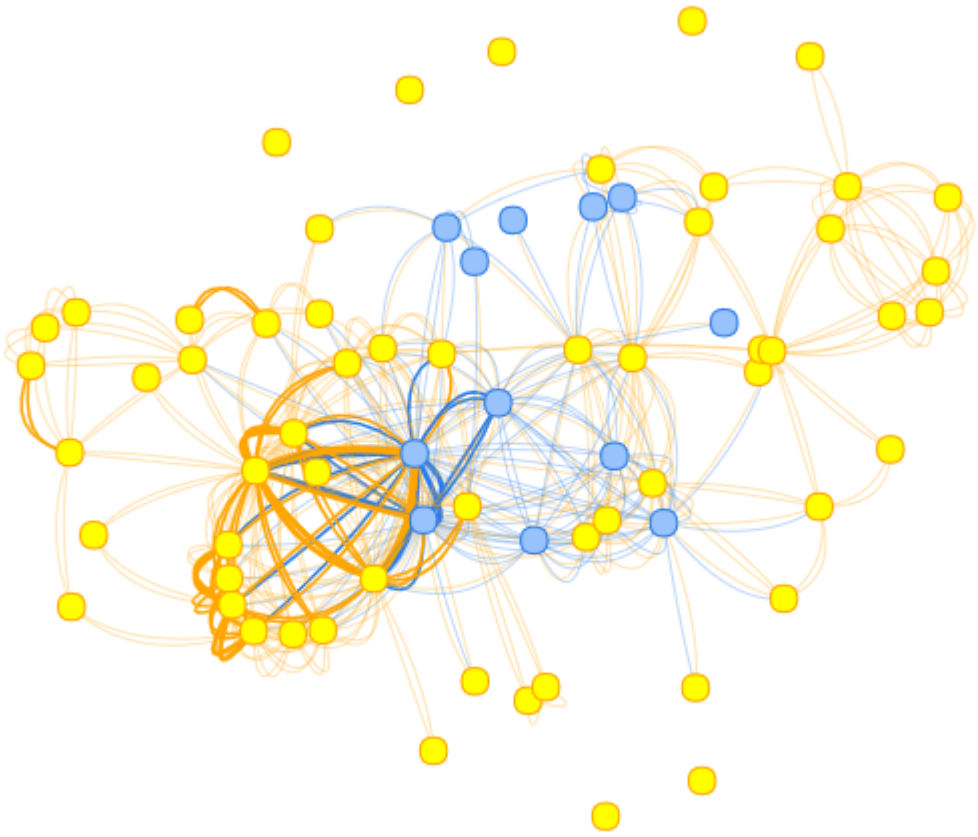
It can be seen that most of the terrorists involded in the bombing are connected to each other directly. There are a few people though who are not directly connected to any of the terrorists involved in bombing. It is seen that the two terrorists with most number of connections, Jamal Zougam and Mohamed Chaoui have not used their close

allies for bombings.We can see a number of clusters which appears around Jamal Zougam, Mohamed Chaoui, Said Berrak, Mohamed Chedadi, Basel Ghayoun and it is evident that each cluster is connected to atleast one on the top 3 terrorists with highest number of connections.

2

```
Network2 <- visNetwork(nodes, edges)%>%
  visPhysics(solver='repulsion') %>%
  visOptions(highlightNearest = list(enabled = TRUE, degree = 2,
                                     labelOnly = FALSE, hover = TRUE),
             nodesIdSelection = TRUE)
Network2
```

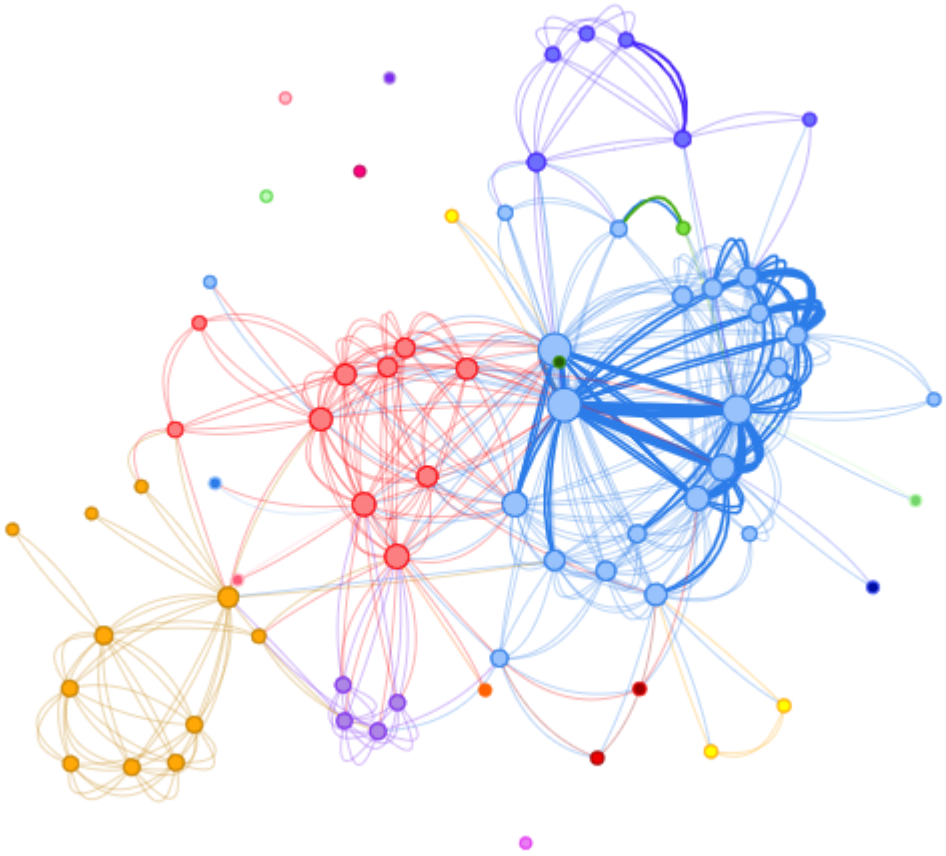
Select by id



It seems like Jamal Zougam and Abdeluahid Berrak both had equal and best opportunity to spread the information in the network. Jamal Zougam was believed to be the person who sold telephones which were used to detonate the bombs in the attack. This may be somehow one of the reasons why he was the center of the network. He also reportedly helped construct the bombs and was one of the first to be arrested.

3

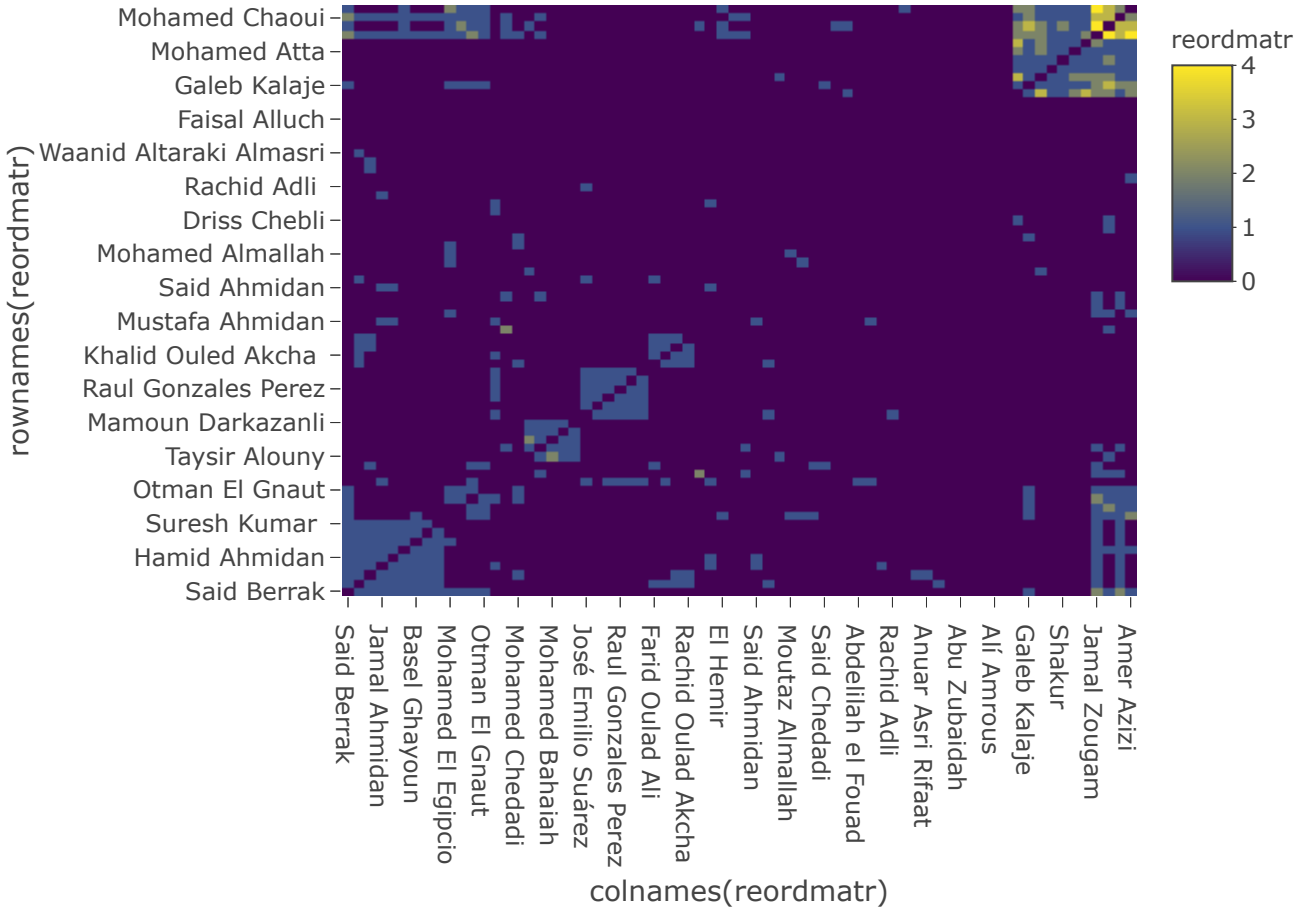
```
new_nodes1 <- nodes2
graph2 <- graph_from_data_frame(d=edges, vertices=nodes2, directed=F)
ceb <- cluster_edge_betweenness(graph2)
new_nodes1$group=ceb$membership

Network3 <- visNetwork(new_nodes1,edges) %>%
  visPhysics(solver='repulsion') %>%
  visOptions(highlightNearest = list(enabled = TRUE, degree = 1,labelOnly = FALSE, hover = TRUE
))

Network3
```



A few of the clusters that were identified manually in step 1 were discovered by this clustering method in addition to a few more clusters.

4

```
netm <- get.adjacency(graph2, attr="value", sparse=FALSE)

colnames(netm) <- V(graph2)$label
rownames(netm) <- V(graph2)$label

rowdist<-dist(netm)

order1<-seriate(rowdist, "HC")
ord1<-get_order(order1)

reordmatr<-netm[ord1,ord1]

plot_ly(z=~reordmatr, x=~colnames(reordmatr),
        y=~rownames(reordmatr), type="heatmap")
```



The most pronounced cluster seems to appear at the top right corner. These clusters seems to appear mostly around the top terrorists i.e. who had the most number of connections and these clusters were also discovered in the 1st and 3rd plot.

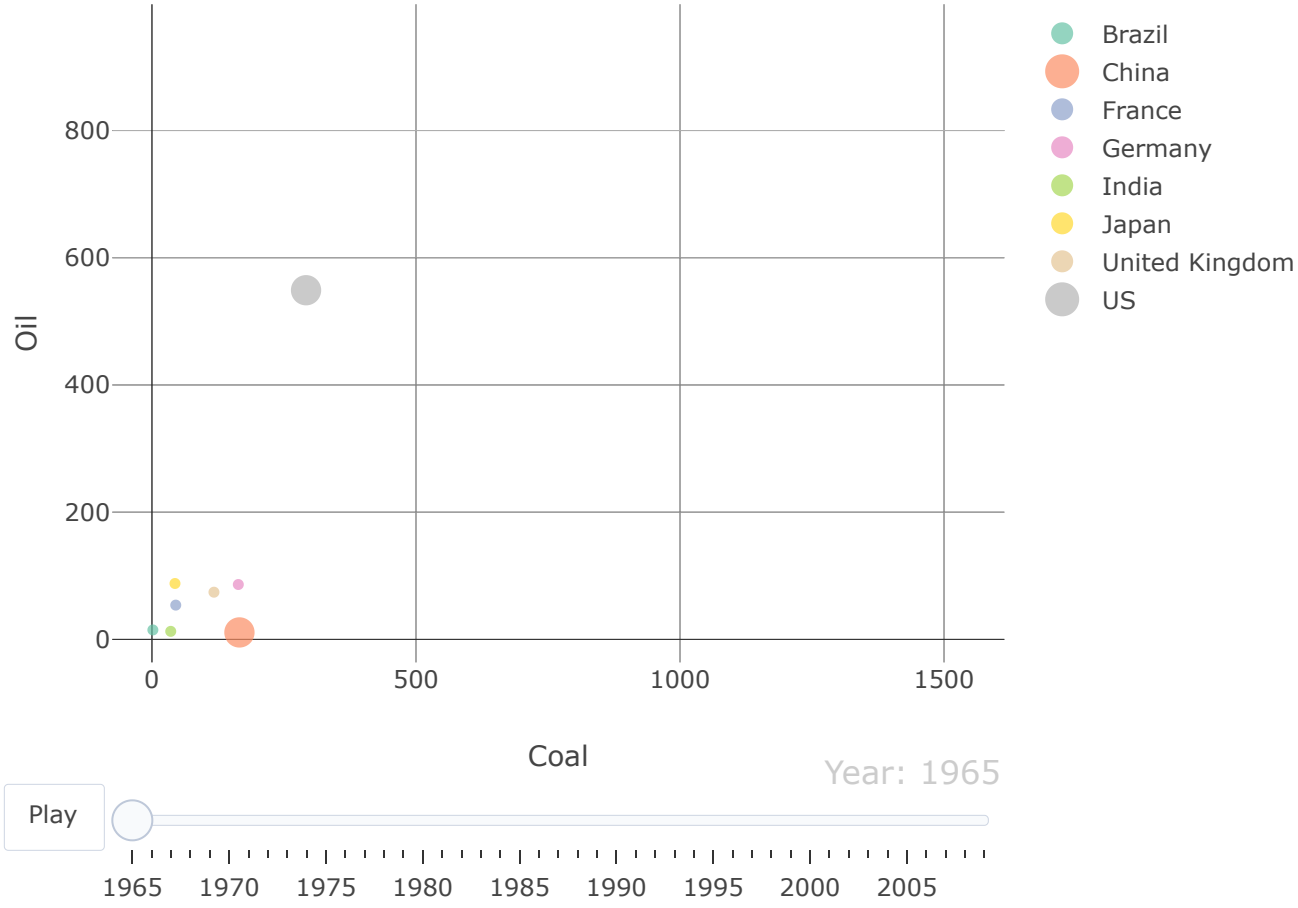
Assignment 2

1

```
Oilcoal <- read.csv2("Oilcoal.csv")
Oilcoal <- as.data.frame(Oilcoal)
Oilcoal$Coal <- as.numeric(Oilcoal$Coal)
Oilcoal$Oil <- as.numeric(Oilcoal$Oil)
Oilcoal$Marker.size <- as.numeric(Oilcoal$Marker.size)

plot_ly(Oilcoal, x=~Coal, y=~Oil, frame =~Year, size =~Marker.size, color =~Country) %>%
  animation_opts(100, easing = "elastic", redraw = F)
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

```
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
```



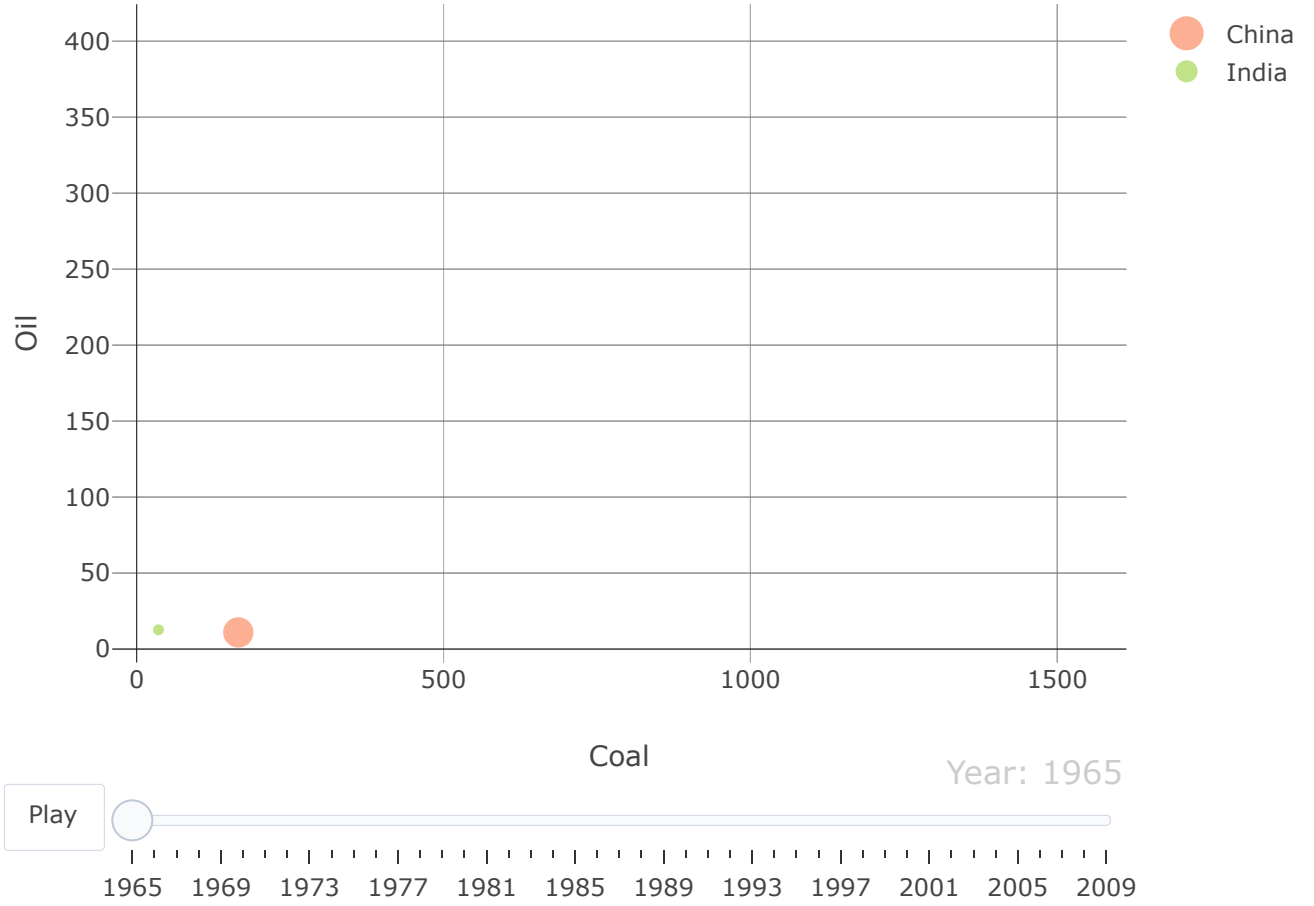
By analyzing the animation we can see that all countries seem to have increased its coal and oil consumption. US is being the biggest consumer of Oil all along. China has increased the cunsumption of coal drastically and more than any other country, while the rate of increse of its oil consumption is not as big as that of coal. India seems to be the only other country which increased its Oil and Coal consumtion at a reasonable rate to each other. All the other country's Coal and Oil consumption seem to remain close to as it was in the beginning and has minimal increase.

2

```
df <- Oilcoal[Oilcoal$Country %in% c("India","China"),]
plot_ly(df, x=~Coal, y=~Oil, frame =~Year, size =~Marker.size, color =~Country) %>%
  animation_opts(100, easing = "elastic", redraw = F)
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

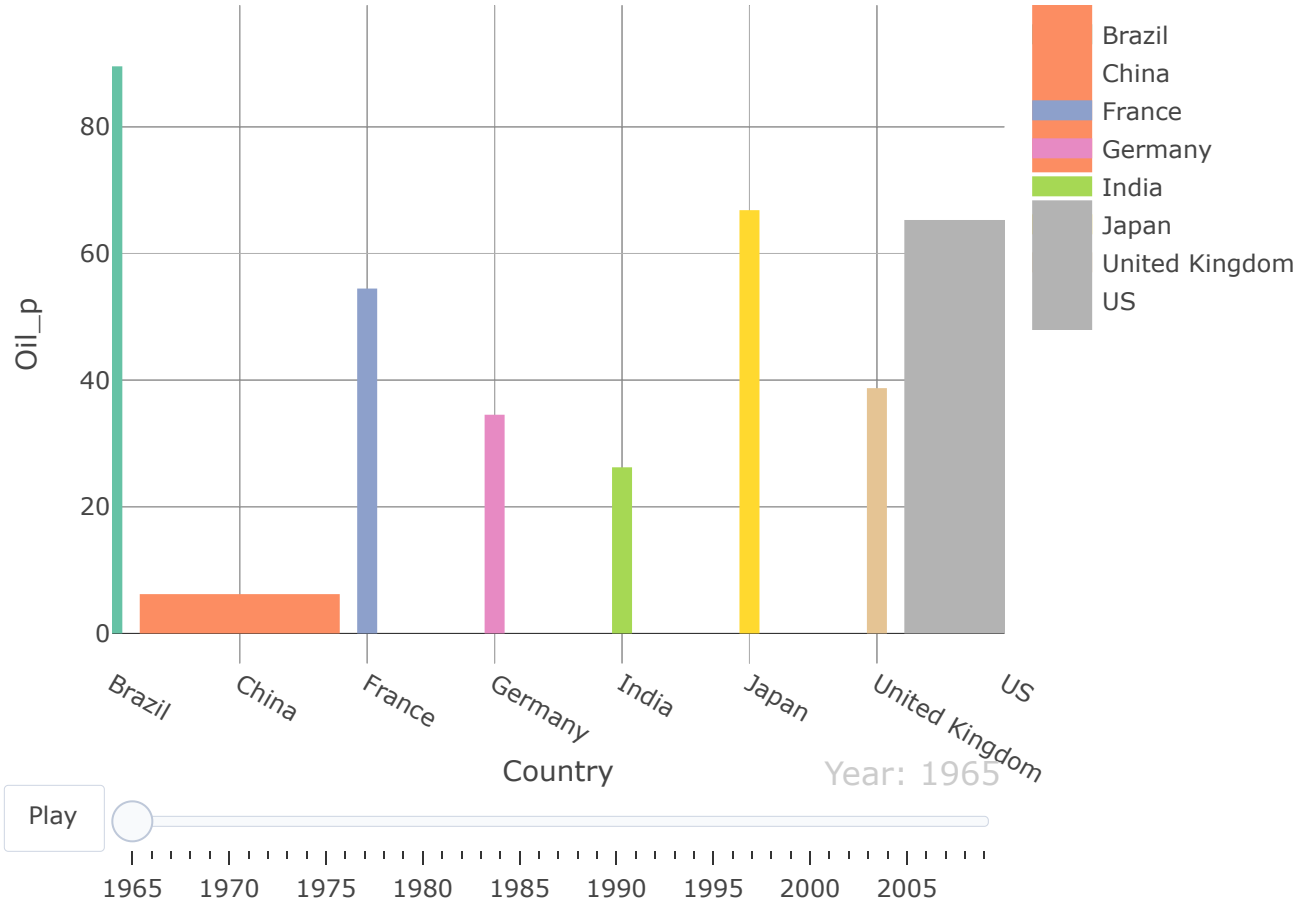
```
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
```



India and China have similar motion pattern both of the countries are increasing the oil and coal consumption whereas China's increase is much more then that of India. Both the countries due to increase in polulation are in need of more energy resources. High economic growth in China and India being a developing country increased its oil consumption. China's and India's energy use is projected to double from 2005 onwards as both of them need resources to meet there energy demand.

3

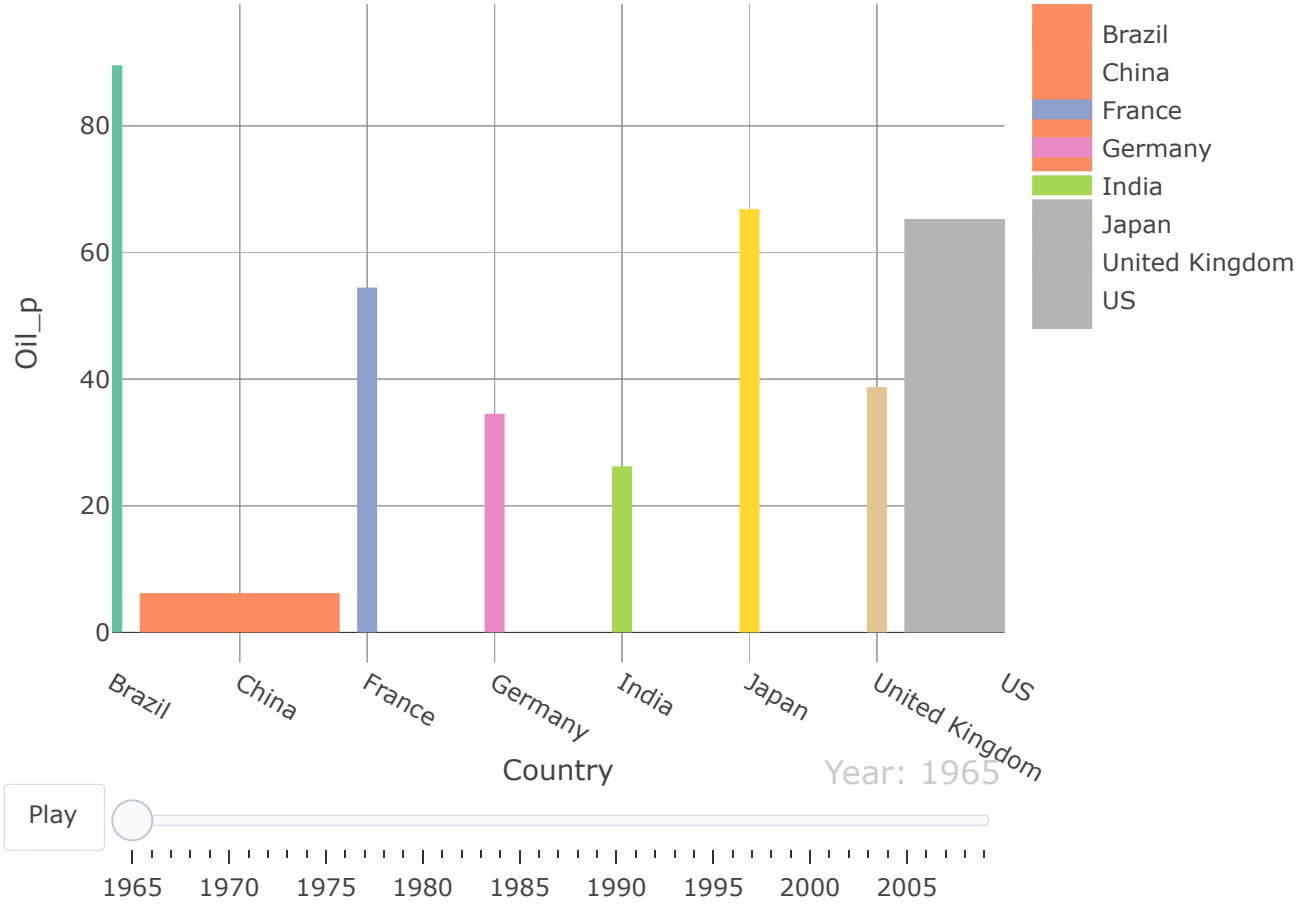
```
Oilcoal$Oil_p <- Oilcoal$Oil/(Oilcoal$Oil + Oilcoal$Coal)*100
Oilcoal_new <- Oilcoal[rep(row.names(Oilcoal),2), ]
Oilcoal_new[361:720,]$Oil_p <- 0
Oilcoal_new <- Oilcoal_new[,c(1,2,3,4,5,7)]
Oilcoal_new <- Oilcoal_new[order(Oilcoal_new$Year, Oilcoal_new$Country),]

p3 <- Oilcoal_new %>%
  plot_ly(x = ~Country, y = ~Oil_p, size = ~Marker.size, color = ~Country,
    frame = ~Year, text = ~Country, type = 'scatter',mode = "lines",showlegend = T
  )
p3
```



Using animated Bar chart we can easily recognize the increase in the consumption of the resources by country over time and see which country have more increase in consumption as compared to the others. Whereas bubble chart provides more exact information about the countries and their resource consumption, also it shows both the variables oil and coal. In Bar chart the drawback is that we can only see increase for one variable which is the relation between oil and coal.

4

```
p4 <- p3%>%animation_opts(
  easing = "elastic", redraw = F)
p4
```



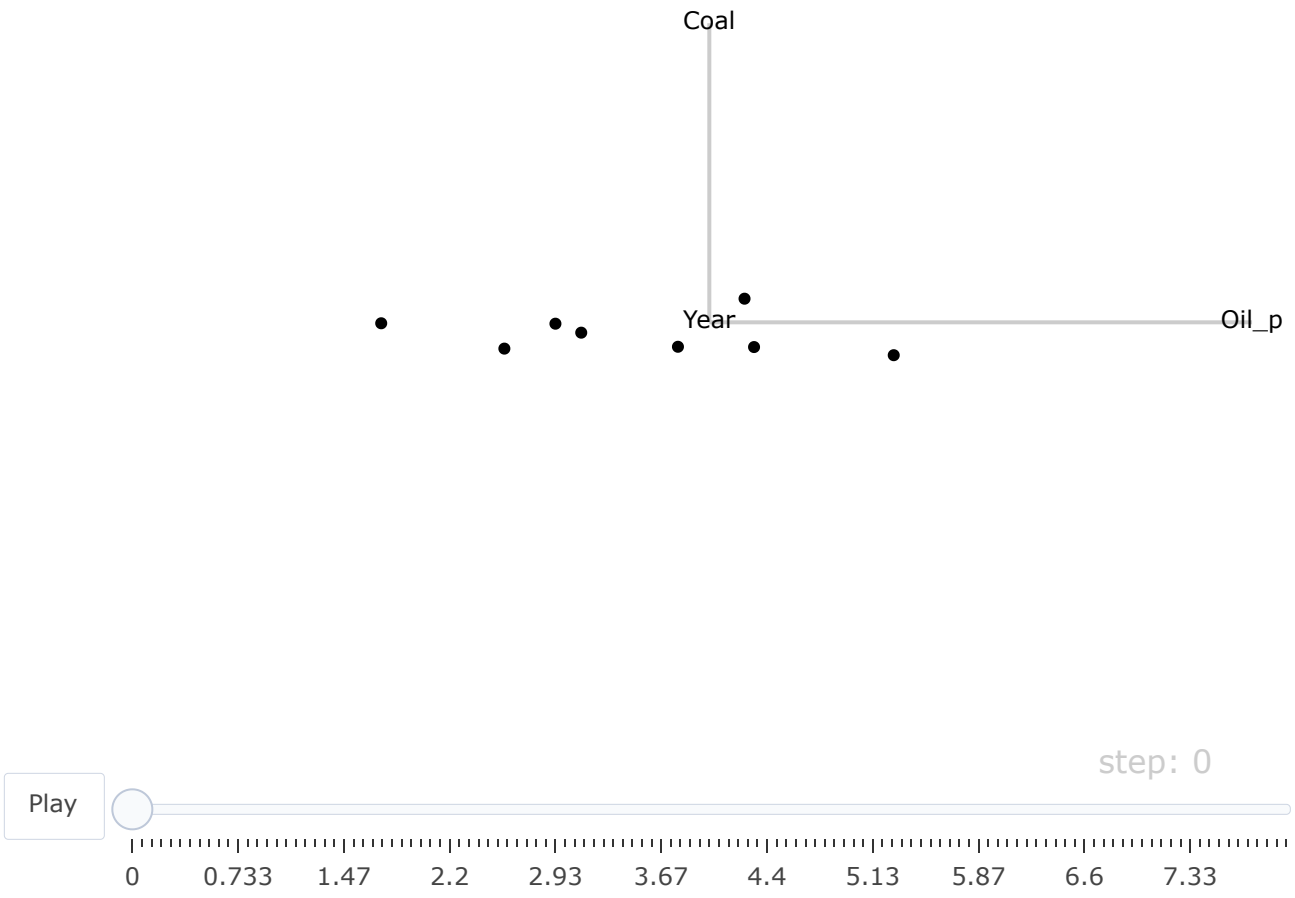
By adding the easing to the bar chat we can see that the plot is same but the transition is stepwise. We can compare it with easeInOutBounce as it is also in step wise. By using easing we get advantage of change in the countries at the specific year big changes are easily visible. The disadvantage is that we can not see the time transition by using easing it is a sudden change which is not good for analyzing the transition between years.

5

```
mat <- rescale(Oilcoal[,c(7,3,2)])
row.names(mat) <- as.character(Oilcoal$Country)
set.seed(12345)
tour<- new_tour(mat, guided_tour(cmass), NULL)
steps <- c(0, rep(1/15, 120))
Projs<-lapply(steps, function(step_size){
  step <- tour(step_size)
  if(is.null(step)) {
    .GlobalEnv$tour<- new_tour(mat, guided_tour(cmass), NULL)
    step <- tour(step_size)
  }
  step
})

# projection of each observation
tour_dat <- function(i) {
  step <- Projs[[i]]
  proj <- center(mat %*% step$proj)
  data.frame(x = proj[,1], y = proj[,2], state = rownames(mat))
}

# projection of each variable's axis
proj_dat <- function(i) {
  step <- Projs[[i]]
  data.frame(
    x = step$proj[,1], y = step$proj[,2], variable = colnames(mat)
  )
}

stepz <- cumsum(steps)

# tidy version of tour data

tour_dats <- lapply(1:length(steps), tour_dat)
tour_datz <- Map(function(x, y) cbind(x, step = y), tour_dats, stepz)
tour_dat <- dplyr::bind_rows(tour_datz)

# tidy version of tour projection data
proj_dats <- lapply(1:length(steps), proj_dat)
proj_datz <- Map(function(x, y) cbind(x, step = y), proj_dats, stepz)
proj_dat <- dplyr::bind_rows(proj_datz)

ax <- list(
  title = "", showticklabels = FALSE,
  zeroline = FALSE, showgrid = FALSE,
  range = c(-1.1, 1.1)
)

# for nicely formatted slider labels
options(digits = 3)
tour_dat <- highlight_key(tour_dat, ~state, group = "A")
tour <- proj_dat %>%
  plot_ly(x = ~x, y = ~y, frame = ~step, color = I("black")) %>%
```

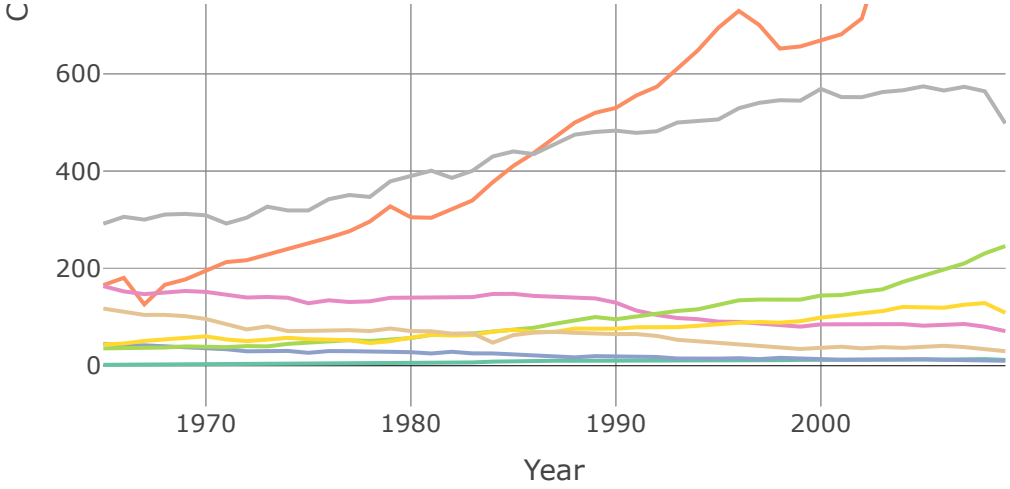
```
  add_segments(xend = 0, yend = 0, color = I("gray80")) %>%
  add_text(text = ~variable) %>%
  add_markers(data = tour_dat, text = ~state, ids = ~state, hoverinfo = "text") %>%
  layout(xaxis = ax, yaxis = ax, showlegend = F)
```

```
tour
```



Clusters does correspond on year ranges to some extent here. China seems to have largest contribution on this projection.

```
plot_ly(Oilcoal, x = ~Year, y = ~Coal, type = 'scatter', mode = 'lines', color = ~Country)
```



By looking at the plot it is evident that China’s coal consumption is on the rise continously and way more than any other country in the group and hence had the largest contribution to the projection.

Lecture 1

Introduction to Data Visualization

1

About the course

Course structure

- 7 lectures (presentations)
- 6 labs, work in groups 2 persons
- 3 seminars
- Star-marked assignments in 3 occasions – to be solved individually, optional.

Examination

- Submission of lab reports
- Presentation of lab reports and opposition
- Computer-based written exam
- **Star-marked assignments passed+ earned at least 14 points at the exam =get 2 points more**

2

Introduction

Visualization in Statistics and Machine Learning…

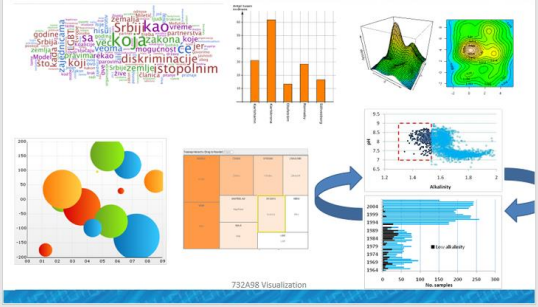
… is a methodology that allows for discovering or confirming a useful information about the data by constructing and examining the graphical output

Course contents

- **Topic 1**: Introduction to Data Visualization. Introduction to Ggplot2, Plotly, Shiny.
- **Topic 2**: Perception and Visualization. Data preprocessing.
- **Topic 3**: Basic graphs. Geospatial visualization.
- **Topic 4**: Multivariate data visualization.
- **Topic 5**: Interactive visualization. Text visualization.
- **Topic 6**: Graph visualization. Animation.

5

Viualizations



6

About the course

Information & Lab reporting

- LISAM is used
- Good lab practices
  - Supervision time is limited (2h)
  - Lab is normally put at LISAM a day before the lab supervision session
  - Start doing lab before the supervision session
  - Possible strategy: one individual in the group works with assignment 1, one with assignment 2 during the supervision time, then help each other later
- Deadlines
- Seminars are obligatory – speakers and opponents selected randomly

3

About the course

Course literature:

- **"Interactive Data Visualization"** by M.O. Ward et al., Second Edition.
- Papers, software documentation & manuals

- Decide groups
  - https://docs.google.com/spreadsheets/d/1GbN6K4dZp2MtgTX5QiKc53QHvYygzd7Iyb2zy3Tls4o/edit?usp=sharing

4

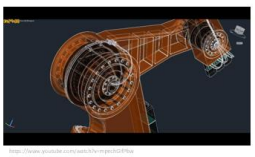
Different types of visualization

- In this course, we focus on **visualization=information visualization**
  - Data→Visualization→Analysis

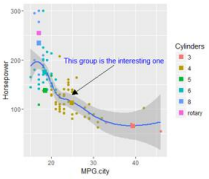
- Related concepts
  - **Computer graphics**: Data is not necessary present, analysis is not normally assumed
    - Example: Computer games
  - **Scientific visualization**: similar to information visualization, often engineering data, statistical/machine learning analysis is normally not assumed
    - Example: Industrial robots

7

Different types of visualization



Scientific visualization



Information visualization

8

Challenges in information visualization

- Which graphs can be used for analysis of my data?
- How to create these graphs?
- How should these graphs be analysed?
- How to make these graphs looking good enough for publication or presentation?

9

Why is visualization important?

- Human sight = primary resource for information understanding
- Visualization is often the **quickest** way for data understanding
- The way of data visualization may affect decision making dramatically

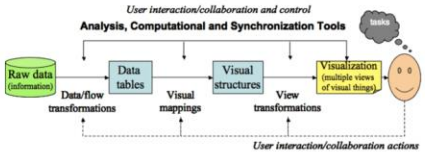
10

Visualization aims

- Visualization for **exploration**
  - Clusters
  - Trends
  - Anomalies
  - …
- **Confirmatory** visualization
  - Example 1: Perform linear regression, analyse residuals→ was linear regression reasonable
  - Example 2: Discover clusters by K-means, visualize clusters→ are they clusters actually?
- Visualization for **presentation**

13

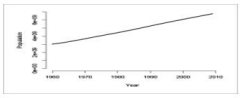
Visualization pipeline



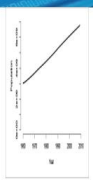
Key ingredient: mapping data columns to visual structures (aesthetics)

14

Why is visualization important?



Decision here: population does not increase so much, no intervention needed

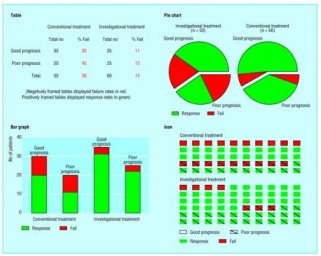


Decision here: population increases quickly, intervention is required

Visual perception problem

11

Why is visualization important?



12

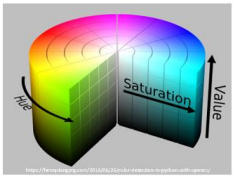
The role of perception

- Human visual system has limitations
- These limitations may lead to wrong/incomplete analysis of graphs
- Understanding how we see→ better displays
- Misleading graphics needs to be avoided

15

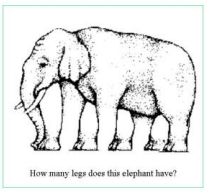
Colors

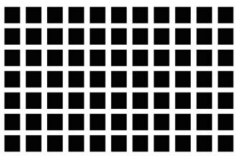
- Color= hue + saturation + value (lightness)
- 8% of males are color deficient→ what are good colors?



16

Illusions

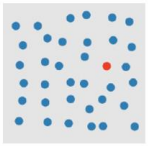




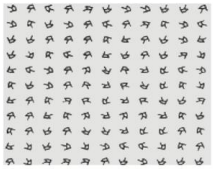
17

Preattentive processing

- Certain aesthetics are fast to process



How quickly can you identify a red dot?



How quickly can you identify a square of right-handed Rs?

18

Data preprocessing

- Normalization
  - Converting column to range [0,1]. Useful in for ex. color mapping
  - Centering and scaling 0/1
  - Nonlinear transformations: log, sqrt
- Segmentation
  - Split data according to some column

21

Data preprocessing

- Sampling, subsetting and expanding
  - Random sampling reduces size of data and facilitates overplotting (for ex. scatterplots)
  - Interpolation: linear (one dimension), bilinear (two dimensions), nonlinear. Select necessary amount of intrepolation points.
- Dimension reduction
  - PCA
  - MDS
  - Other techniques (ex. ICA, Autoencoders), welcome to **Machine Learning** course..

22

The role of perception

- How can this affect analysis?

19

Data preprocessing

- Viewing raw data is often prefered
- Sometimes some preprocessing is needed
- Missing values and Data cleaning
  - Discard the bad record → may remove almost all data
  - Assign sentinel value
  - Column mean imputation
  - Nearest neighbor imputation
  - Other imputations

20

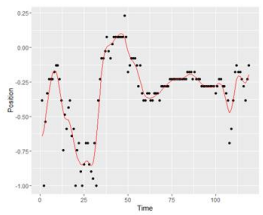
Data preprocessing

- Mapping nominal dimensions to numbers
  - Random mapping should never be done unless intrinsic ordering is present
  - Use other numeric variables to measure in the data to measure "closeness" of values in the nominal variable
  - Correspondence analysis
- Aggregation and summarization
  1. Grouping observations
  2. Computing summary statistics per group

23

Data preprocessing

- Smoothing and filtering
  - Replace original values with a smoothed versions



24

Software

Commercial:

- SAS and SAS JMP – environment. Special visual tools are available (JMP), require separate license. Well documented. Good even for large sets. **SAS Enterprise Guide** has many visual static tools
- Spotfire – Many static and interactive visualization tools
- Tableau– Many static and interactive visualization tools
- InfoScope – visualizing maps, interactive visualizations

Free:

- R – programming language. Set of packages is constantly updated. A lot of statistical tools (even the newest methods) Badly documented
- Plotly – a tool for interactive and dynamic graphics, R interface available
- Shiny - – a tool for R-based web applications using graphics
- GraphViz – visualization of graph data, coding needed
- Jigsaw – Text analysis

25

Software

Tools for the web (used by web designers):

- ActionScript
- JavaScript
- Prefuse
- VTK

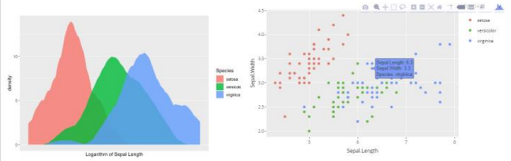
... much more references given in the course book

26

Graphical tools in R

Ggplot2 package: based on **grammar of graphics**, close to publication quality

Plotly package: Ggplot2 + interactivity



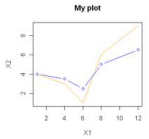
29

Graphical procedures

Base R graphical procedures:

- plot(x,..) plots time series
- plot(x,y) scatter plot
- plot(x,y) followed by points(x,y) plots several scatterplots in one coordinate system
- hist(x,..) plots a hitogram
- persp(x,y,z,...) creates surface plots
- cloud(formula,data..) creates 3D scatter plot

```
x<-c(1,4,7,8, 12);
y<-c(4,3,1,6,9);

plot(x,y, type="l", col="orange",
     main="My plot", xlab="X1", ylab="X2");
points(x, y/2+2, type="b", col="blue");
```



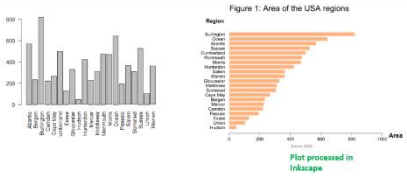
30

Publication quality graphics

- Visualization for exploration
  - Default settings
- Visualization for presentation for publication
  - Higher quality graphics is required
    - Improve the graph quality in the software (often requires quite a bit of programming)
    - Use postprocessing tools, such as Inkscape or Adobe Illustrator

31

Publication quality graphics



Example: Compare two plots and state what is improved in the second plot.

32

Making publication quality graphics

- Install **Inkscape**
  - http://inkscape.org/
  - Inkscape is an open source, *SVG*-based vector drawing program
  - file format that Inkscape uses is compact and quickly transmittable over the Internet.
  - Vector graphics: Image is defined in terms of lines, not pixels
    - Benefit: can be enlarged without loss of picture quality

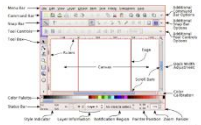
1. Save your R plot as PDF and import it to Inkscape
2. Make changes and export your plot as a PNG-file or save it as PDF.

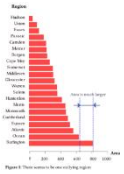
Bitmap image and vector image (enlarged)

33

Inkscape

- Menu bar (**Important:** File, Object, Path, Text, View→Grid)
- Command bar (Zoom to fit page, Edit object's colors)
- Tool controls
- Tool box (Select and transform, zoom, Text, Eraser, Fill)
- Color palette
- Status bar





34

Home reading

- Course book, chapters 1.1, 1.3-1.8 and 2
- Manual to InkScape: http://tavmjong.free.fr/INKSCAPE/MANUAL/html/index.php

35

Perception and visualization. Preprocessing.

Lecture 2

732A98

Human perception

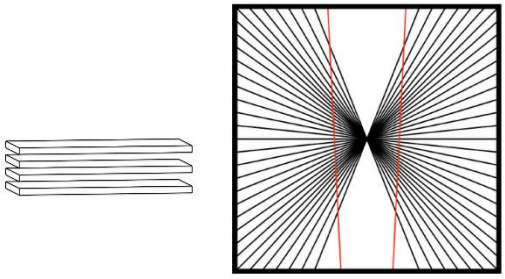
- How are visualizations perceived by different humans?
- How do we know that a given visualization is correctly interpreted?

Perception:

- Recognizing
- Organizing (gathering, storing)
- Interpreting (binding to knowledge)

Illusions

- Human perceptual system is not perfect

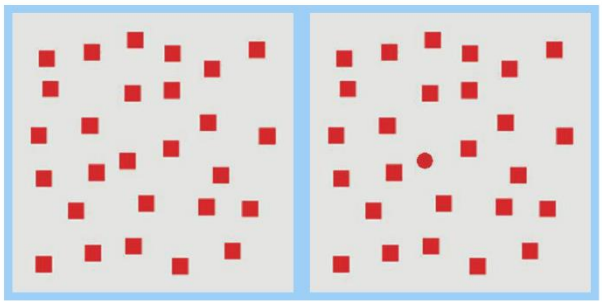


Perception mechanism

- Preattentive
  - Fast (250 ms)
  - Performed in parallel
- Attentive
  - Slow
  - Uses short term memory
  - Transforms simple visual features into structured objects
  - Compares to memory models (ex. door)

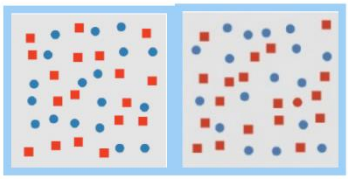
Preattentive processing

- Preattentive feature= shape
- How quickly do you see a red circle?



Preattentive processing

- **Important**: Combination (conjuction) of nonunique features can not be detected preattentively
  - Many red objects
  - Many circle objects
- How quickly can you find a unique object here?



Preattentive features

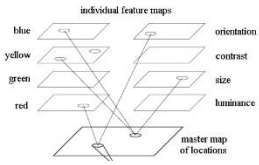
- Length
- Width
- Size
- Curvature (shape)
- Hue
- Intensity
- Flicker
- Direction of motion
- 3D depth
- Lighting direction

Preattentive visual tasks

- Presense or absense of object with a unique visual feature among distractors is detected preattentively
- Boundary between two groups of elements with the same visual feature is detected preattentively
- Movement of an object with a unique visual feature is tracked preattentively
- Amount of elements with a unique visual feature is estimated preattentively

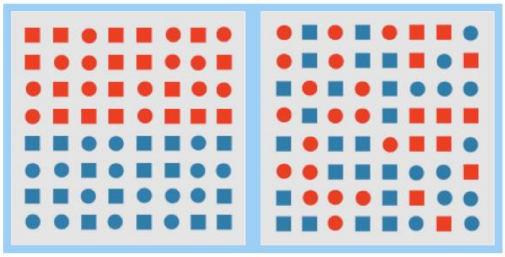
Treisman's theory of preattentive processing

- A figure is processed in parallel by checking individual feature maps
- A specific preattentive task is performed in each feature map
- Conjuction of features requires searial search between maps - takes time



Treisman's theory of preattentive processing

- How quickly can you identify a boundary?



Metrics

- What graphical features can be accurately perceived by humans?
- How many distinct entities can be visualized without confusion?
- How should we use color?
- How should we combine features in a complex phenomenon?

Channel capacity: how many different levels of a feature we can perceive

- 8 levels = 3 bits

Metrics

- Position on a line: 10-15 levels (3.25 bits)
- Size of squares: 4-5 levels (2.2 bits)
- Color: hue 10 levels, brightness: 5 levels (3.1 bits, 2.1 bits)
- Line length: 2.8 bits
- Line orientation: 3 bits
- Line curvature: 1.6-2.2 bits

Summary: 6-7 unique values max.

Metrics

Note: Combining metrics does not sum up the capacity!…

- Hue and saturation: 3.6 bits
- Size, brightness and hue: 4.1 bits
- Position in a square: 4.6 bits

Metrics

Relative judgement: comparing two values of a feature

Errors (in increasing order)

- Position along a common scale
- Length
- Angle
- Area
- Volume
- Color hue

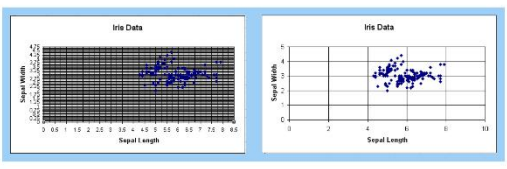
–> *Pie charts* are less effective than *Bar Charts*

Principles of good visualization

- Use intuitive mapping to aesthetics
  - Visualization type is adopted to user's background
  - Geographical coordinates –> X,Y, temperature–>color
  - Use correct mapping
    - Ordinal variables- X,Y, saturation, orientation
    - Nominal variables - shape, texture, hue
- Support view modifications
  - Scrolling, zooming
  - Color map
  - Mapping aesthetics
  - Scales

Principles of good visualization

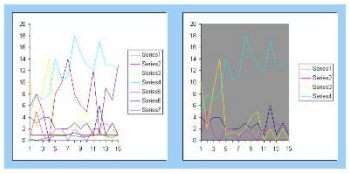
- Do not put too much information in the display (occlusion)
- Add keys, labels, legends, grids with care
- Use display efficiently (0%-100% scale vs actual domain)



Principles of good visualization

Color:

- Keep the number of colors low (5-6 distinct)
- Use redundant mappings (color+size)
- Include labeled color key
- Use resonant colors



Principles of good visualization

Aesthetics:

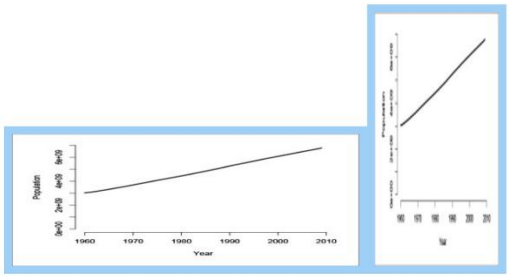
- Important findings should be visually emphasized
- Most important components in the center
- Do not put much information into one display

Other:

- The size of the plot should be normally Horizontal:Vertical=1.5:1
- Text in the graph is normally horizontal
- Caption and Source should be present and informative
- In bar charts, bars are normally sorted
- Axis labels present

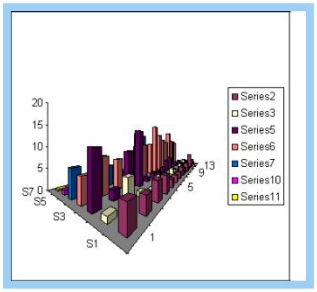
Misleading graphs

- Scaling and perspective problem



Misleading graphs

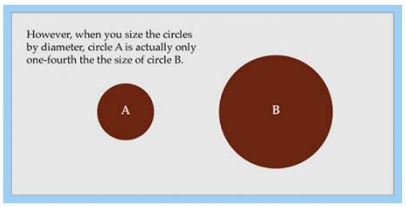
- Scaling and perspective problem



Misleading graphs

Abusing dimensionality/wrong mapping

- A scalar is mapped to a size of a cube
- Mapping is wrong: a scalar is mapped to radius, not area
  - R1=2R2, A1=4A2 !



Misleading graphs

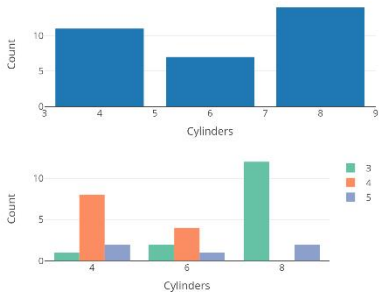
- Mixing data of different nature/scales
  - Ex: One time series plots with two series: Price and Amount both on Y axis
- Smoothed/filtered data interpreted as raw data
  - How good was the smoothing?
- Using of insufficient sampled data

Basic plots

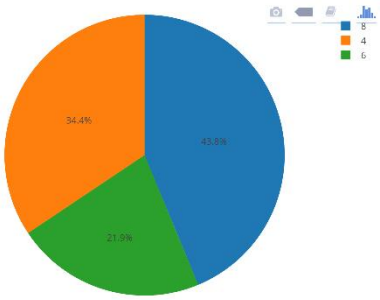
- Quantiative variable:
  1. Computing summaries (ex. frequencies)
  2. Visualizing as bar or pie charts
- What to analyse:
  - Largest and smallest bar or slice
  - For sorted bars, sudden shifts in level
  - Compare first within groups and then difference between groups

Basic plots

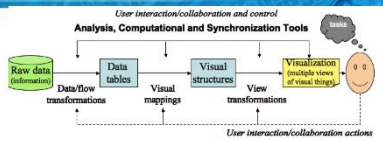
Example: Visualizing number of gears and number of cylinders in cars



Basic plots

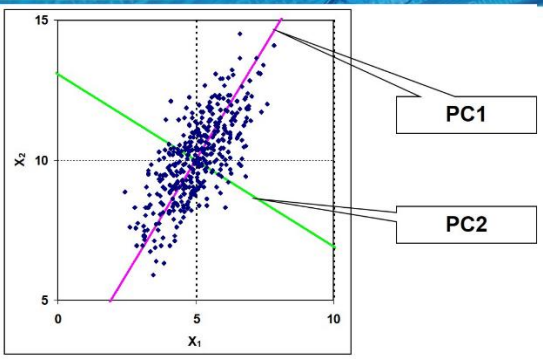


Visualization pipeline



- Dimension reduction
  - PCA
  - MDS
  - Correspondence analysis (nominal)
  - Other techniques (ex. ICA, Autoencoders), welcome to Machine Learning course..

Principal Component Analysis (PCA)



Distance between objects

- Meaning of "two objects are close"?
- Measure of proximity (ex: quantiative vars, Euclidian distance)
- Similarity measure s_{rs} (=1 if same object, <1 otherwise)
  - Ex: correlation
- Dissimilarity measure δ_{rs} (=0 if same object, >0 otherwise)
  - Ex: euclidian distance
- Problem of cosntructing the measures of proximity:
  - What if the variable is qualitative?
  - What if the object is a text document?

Multidimensional scaling (MDS)

Given n objects with known matrix of similarities or dissimilarities. Each object i is characterized by p-dimensional vector X_i

The aim:

- Present these objects in lower dimensions ($p' = 2$ or 3) such that the distance between the new points d_{rs} would reflect the matrix of similarities (or dissimilarities δ_{rs})
- See neighbour observations
- See clusters and outliers
- Have a "map" of your data

MDS

Two types of MDS:

- Metric MDS
- Non-metric MDS

Metric MDS
(algorithm is not discussed here)

Seaching for points $\chi_1, \ldots, \chi_n$, such that distances between $||\delta_{rs}||$ and $||d_{rs}||$ are minimized

Non-metric MDS

Given n objects $X_1, \ldots, X_n$ with known matrix of similarities $||\delta_{rs}||$ of dissimilarities.

For some configuration $\chi_1, \ldots, \chi_n$ (in lower dimension) with matrix $||d_{rs}||$, define **stress** $S(\chi_1, \ldots, \chi_n)$ by

1. Computing d'_{rs} as a a monotonic regression of $||d_{rs}||$ on $||\delta_{rs}||$
2. Computing $S = \sqrt{\frac{\sum_{r,s} (d_{rs} - d'_{rs})^2)}{\sum_{r,s} d_{rs}^2}}$

How to find optimal configuration?

- Use numeric optimization to minimize $S(\chi_1, \ldots, \chi_n)$

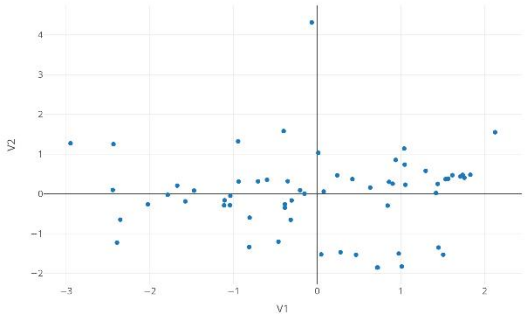
MDS- examples

Music data

- Artist (abba. Beatles. Wiwaldi, Mozart, Beethoven, Enya)
- Type (rock, classical, new wave)
- lvar, lave, lmax, lfener, lfreq - parameters of the music signal

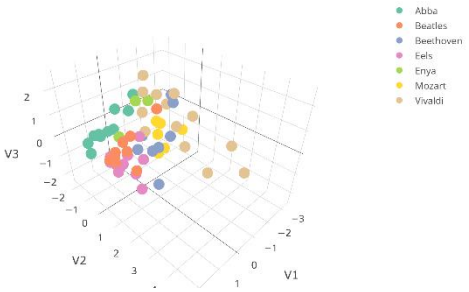
Metric MDS

- Mapping into two dimensions and using scatterplot



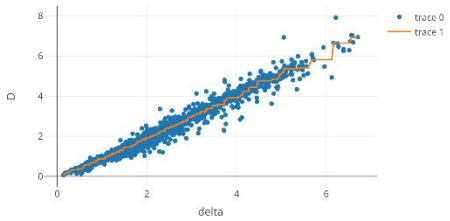
Non-metric MDS

- Mapping into three dimensions, coloring by Artist and using 3D-scatter:



Shephard plot

- Plot of d_{rs} vs δ_{rs}
- Displays also δ'_{rs} for non-metric MDS
- Shows the quality of MDS fit-> Best if scatter reminds a monotonic curve



Read at home

- Book, chapters 3.1, 3.3, 3.5, 13
- Cox, AA, and Cox, T.F.: "Multidimensional scaling." Handbook of data visualization. Springer, Berlin, Heidelberg, 2008. 315-347.
- Plotly book, ch 2.3

Density plots and box plots

What should be analysed?

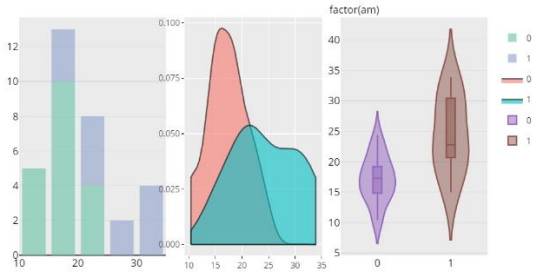
- Density plot, histogram, violin plots
  - Mean value or typical value
  - Symmetry
  - Variation
  - Whether reminds some distribution
  - Heavy/Light tailed
  - One ore more modes
  - Skewness

Density plots and box plots

What should be analysed?

- Box plot
  - Median
  - Variation
  - Outliers
  - Symmetry
  - Quantiles

Example: Visualizing miles per gallon depending on transmission type



Scatter plot

- Y: dependent variable, X: independent variable
- Smoother is a good idea to have

Analysis:

- Shape (data=true+error, true=linear, quadratic, cubic, exponential, .., empirical)
  - How to find the right model?
  - Fitting the data (regression)
  - Analysis of residuals or model selection methods
- Strength (how close observations to a hypothized model)
  - If linear, Correlation r or coefficient of determination R2

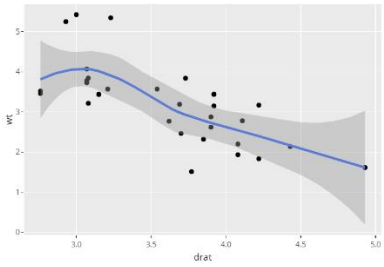
Scatter plot

Analysis:

- Direction (if monotonic, decreasing or increasing; if not monotonic, which parts increasing, which decreasing)
- Density (dense areas, sparse areas)
- Outliers
- Clusters

Scatter plot

Example: Visualizing weight and rear axle ratio



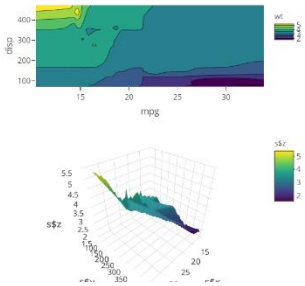
Scatter plot

- More variables can be mapped
- Mark shape
- Mark size
- Mark color
- Mark orientation
- Juxtaposed displays or superimposed displays
- If juxtaposed displays used, we get scatterplot matrix

3D Surface plots and contour plots

- Remember: interpolated data used
- Analysis:
  - Peaks and draughts
  - Trends
  - Additivity
  - Always check the underlying data after

3D Surface plots and contour plots



Geospatial data

- Geographical coordinates are involved
- Used in many applications
  - Climate modeling/analysis
  - Economic/social data analysis
  - Transaction data
  - …

Types of maps

- Symbol/dot maps (nominal/ordinal point data)
- Land use maps/Choropleth maps (nominal/ordinal area data)
- Line diagrams (nominal/ordinal line data)
- Isoline maps (ordinal surface data)
- Surface maps (ordinal volume data)
- Note: Different maps can be used for the same data
  - Choropleth map / Dot map
  - Density surface /dot map

Spatial phenomena

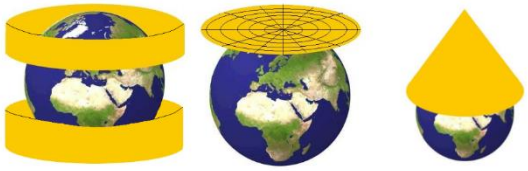
- Point phenomena (ex: building location, city location)
- Line phenomena (paths, roads)
- Area phenomena (counties)
- Surface phenomenon (mountains)

What is map?

- Map coordinates:
  - longitude $\lambda = [-180, 180]$, negative=west
  - lattitude $\phi = [-90, 90]$, negative= south
- Challenge: $[\lambda, \phi] \rightarrow [x, y]$
- Different map projections
  - Conformal projection: retains angles (shapes) but not area
  - Equal area: retains areas but not angles (shapes)
  - …

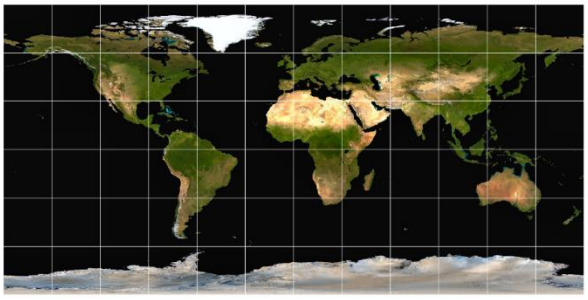
What is map?

- Cylindrical projection, plane projection and cone projection
- Cylindrical projection used by Google, standard now



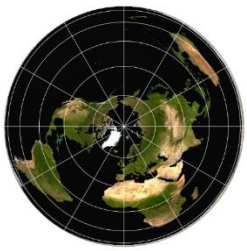
Cylindrical projection

- Conformal projection: far northern/far southern areas inflated
- Defined by $x = \lambda, y = \phi$

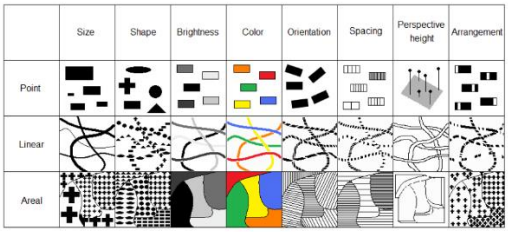


Cone projection

- Albers Equal-area projection
  - Preserves areas
  - Shapes or distances are not correct



Visual variables for spatial data



Symbol/Dot maps

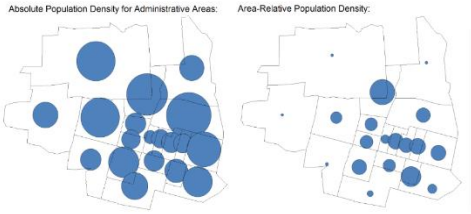
- Data= Lattitude, Longitude+ Other variables
- Latt, Long->Coord, Other variables–>Visual aesthetics
  - Amount is limited! (perception problems)
- Another approach: multiple parameters on multiple maps

Symbol/dot maps

- **Analysis**:
  - Density in geogr areas and between geogr areas
  - Spatial pattern of density (north, south)
  - Clusters, outliers
- **Problems**:
  - Overplotting in highly populated ares
  - If several observations have the same coordinate
  - Size aesthetics used–> perception problem
  - Perceived size depends on local neighborhood (Ebbinghaus illusion)
- Color used: color perception problems

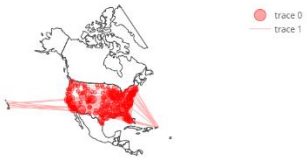
Symbol/dot maps

- **Problems**:
  - Absolute vs relative mapping (proportional to population)



Line diagrams

- Observation: set of (Latt, Long) pairs+ other variables
- Often: start, end point

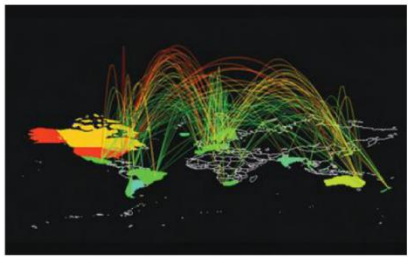


Line diagrams:

- Same as in network analysis plus
- geographical relationships between links and their density (size)
  - Where dense links located?
  - How links are directed?
- Problems:
  - Overplotting
  - If line length analysed -> length perception problem
  - If width analysed -> volume perception problem
  - Colors analysed ->color perception problem

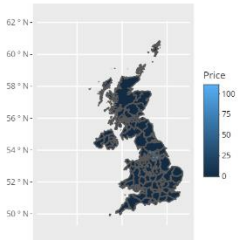
Line diagrams

- Overplotting - possible solution:
  - Using curved lines, minimize edge crossing



Visualizing area data

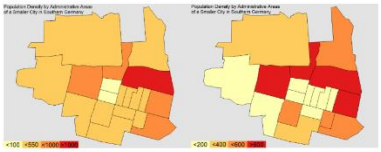
- Data: Name/Coordinates of geographic area+ other variables
- Choropleth maps: variables=color or shaded region on map



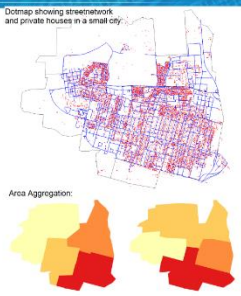
Choropleth maps

- Analysis:
  - Find clusters of regions that are similar
  - Find unusual regions (compared to neighbor regions)
  - Find patterns on the map
- Problems affecting perception:
  - Color/grayscale mapping
  - Choice of regions (county, state,…)
  - Larger region with the same color looks dominating
  - Patterns in small/densely populated areas hard to see

Choropleth maps



Choropleth maps



Visualizing area data

- Isarithmic maps: show areas of phenomenon on the map (density)
  - Contour map
  - Topographic map

Software for geospatial visualization

- Plenty of commercial/Noncommercial software
  - ArcGIS, Google, Yahoo, Microsoft map API
- Plotly
  - plot_geo()
  - Using MapBox + plot_mapbox()
- To use Mapbox:
  - Register with your email, find your token
  - Run in R `Sys.setenv('MAPBOX_TOKEN' = 'your_mapbox_token_here')`
- Ggplot2
  - geom_sf()
  - ggmap

Using maps

- A few countries available through plotly
- Downloading map of a country:
  - Finding a country map http://gadm.org/
  - Decide what level of detalization is needed (region, county,…)
  - Download R(sf) file.
  - Load the file to R using readRDS function
    - e.g. rds <- readRDS('filename.rds')
  - Use with ggplot()+geom_sf(data=rds)
  - Use with Plotly: plot_ly()+add_sf(data=rds)

Multivariate data visualization

Continious variables involved in the following:

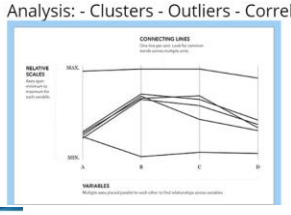
- Parallel coordinate plots
- Heatmaps
- Star charts

Parallel coordinates

Construction:

- Vertical axis: Values
- Horizontal Axis : Variables
- 1 trace line = 1 observation

Analysis: - Clusters - Outliers - Correlated variables



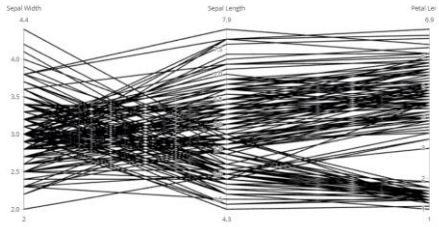
Analysis

- Positive correlation between two adjacent variables: almost all segments are parallel to each other
- Clusters in some variable space: several trace lines that are near each other and have similar pattern
- Outliers: trace lines that have unusual pattern and/or fall out outside the common plot area

Problems

- Trace lines overlap each other -> difficult to find patterns, difficult to follow a specific trace line
- Analysis depends much on the order of variables (correlation, clusters) -> a proper reordering may improve the analysis

Example: Iris dataset - How many clusters do you see?



- Sometimes clusters overlap with categories given by some variable
  - Non-mixing groups is not the same as clustering!

Ordering problem

- Problem of ordering (variables, observations) is one of the key problems in multidimensional visualization
  - Sometimes has a huge impact on perception (heatmaps)
- A lot of approaches exist

Problem formulation: Given data set
$\chi = (x_{ij} | i = 1, \ldots, n, j = 1, \ldots, p)$

- Select order $\Psi = i_1, \ldots, i_p$ that optimizes visual perception (analysis) -> this defines reordering of data columns
  $\Psi : \chi \rightarrow \chi'$

Note: $p!$ possible orderings exist…

Ordering problem

Solution

- early approaches (for ex. Ankerst et al. 1998):

1. Choose a distance (proximity) matrix $D = \{d_{ij} = d(x^i, x^j)\}$ between variables (columns)
   - Euclidian distance on scaled columns
   - 1- correlation
2. This defines graph with vertices $1, \ldots, p$ and edge weights d_{ij} -> **Hamiltonian path** (Traveling Salesman Problem)

$\min_\Psi \sum_{j=1}^{p-1} d'_{j,j+1}$

- TSP is NP-complete -> Approximate solutions are used

Ordering problem

Solution: modern approaches

- Based on:
  - Decreasing visual clutter
  - Clustering data points/dimensions
  - Outlier detection
  - Dimensionality reduction (for ex. MDS)
  - …
- Note: most of these can be applied both for ordering observations and ordering variables
  - Just transpose the data matrix…

Ordering problem

Objective functions:

- Gradient measures (anti-Robinson)
- Hamiltonian path length
- Least squares
- …

They based on $\min_\Psi L(\Psi(D))$

Optimization algorithms:

- Partial enumeration
- Traveling salesman solvers
- Hierarchical clustering

Gradient measures

Aim: distances should increase from diagonal

$$d_{ik} \leq d_{ij} \text{ for } 1 \leq i < k < j \leq n$$

$$d_{kj} \leq d_{ij} \text{ for } 1 \leq i < k < j \leq n$$

Objective function:

$$L(D) = \sum_{i<k<j} f(d_{ik}, d_{ij}) + f(d_{kj}, d_{ij})$$

where

$$f(z, y) = sign(z - y) \text{ or } f(z, y) = z - y$$

Other objectives

Hamiltonian path length:

$$L(D) = \sum_{i=1}^{n-1} d_{i,i+1}$$

Least squares criterion (PCA)

- Solution is similar to first PCA component

$$L(D) = \sum_i \sum_j (d_{ij} - |i - j|)^2$$

Optimization algorithms

Partial enumeration methods

- Ex: Branch and bounds and dynamic programming
- Constructing solutions by parts

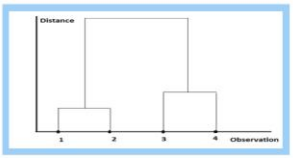
TSP solver

- Suitable for hamiltonian path objective
- Find shortest path by dynamic programming or heuristics

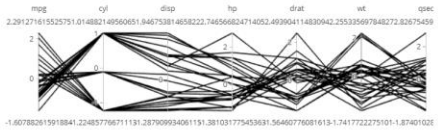
Optimization algorithms

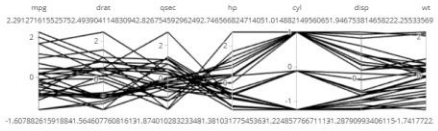
Hierarchical clustering

- Observations are joined into clusters
- Clusters are joined in larger clusters
- Until only one cluster left
- Leaves and branches are permuted to minimize given objective

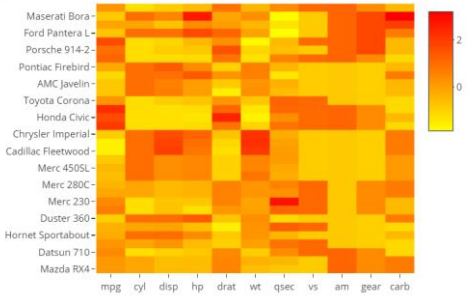


Effect of ordering





Heatmaps



Heatmaps

A heat map visualizes a matrix [n x m]

- Normally rows=observations, columns= parameters
- Heatmap has the corresponding size
- Each cell of the matrix corresponds a cell in the heatmap
- High values correspond intense colors in this map (or visa versa for other color schemes!)
- Names of variables and observations are shown

Heatmaps

Analysis:

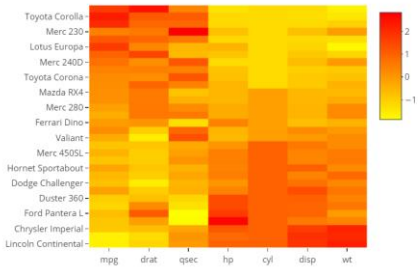
- Compare the values of a parameter for different observations (row)
- Compare the values for a single observation (column)
- Compare the patterns for different rows or columns
- Find similar observations (areas with the same color intensity)
- Find which variables define similarity for a group
- Find correlated variables (similar pattern within a column)

Heatmaps

- Exercise (last picture):
  - How many clusters do you see?
  - Which variables define clusters?
  - Which variables are correlated?

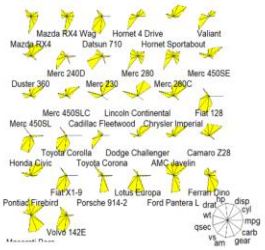
Effect of reordering

- Gradient measure objective used
- See new analysis possibilities



Radar charts

- Use polar coordinate system
- Map column value as a coordinate in certain direction



Radar charts

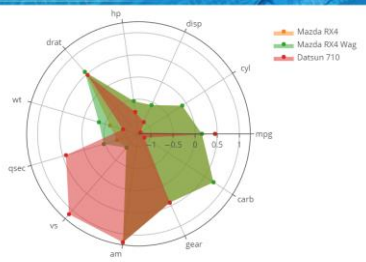
If juxtaposed, analyse:

- Clusters
- Outliers
- Outlying directions

If superimposed,

- Comparing variable length
- Seeing similar and outlying observations

Radar charts



Radar charts

Problems:

- Difficult to judge orientations
- Number of dimensions are observations is very limited
  - Number of observations is extremely limited if superimposed
- More close radar charts easier to compare
- Perception is much affected by observation ordering

Ordering:

- Same as before plus
- Dimensions can be sorted to promote more symmetric charts

Radar charts

- Now with reordering by Gradient Measures

Trellis plots / facets

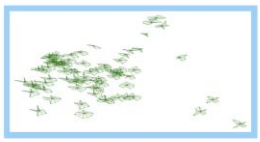
Idea:

1. Make same kind of plot for subsets of data
2. Plot together
3. See patterns/differences

Analogy: cutting a sausage

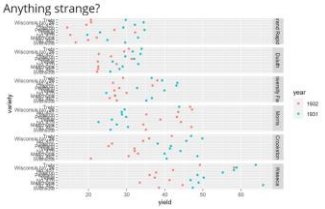
Radar charts

Other positioning possible - PCA/MDS



Trellis plots / facets

- Example: Barley data
  - Anything strange?



Trellis plots / facets

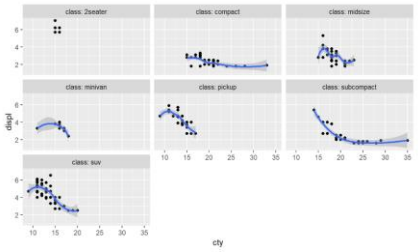
- Faceting = one more aesthetics
- What can be analysed?
  - Patterns within/between plots
  - Conditional dependence $Y \sim X|Z$
  - Variable interaction, additivity

–> Useful tool for modeling!

- Compare : 3D- scatter plots

Trellis plots / facets

- Another car data: is there additivity?



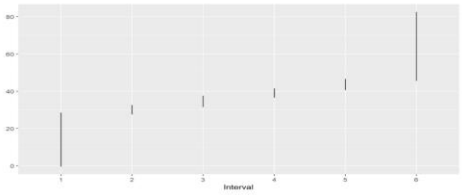
Trellis plots / facets

- Design issues:
  - How to order rows/columns in trellis?
    - A: X=one var, Y0=another var (`facet_grid`)
    - B: independently of aes (`facet_wrap`)
  - How to handle categorical vars?
    - One value/panel
    - Group
    - Ordering? (R: decide factor levels)
  - How to handle real-valued vars?
    - Split equal size/length
    - Shingles

Shingles

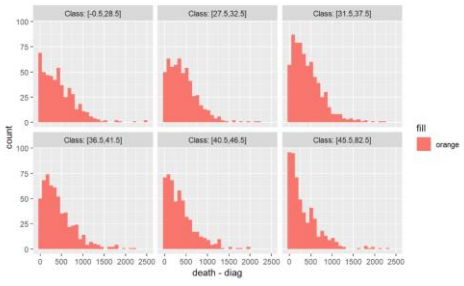
- Creates overlap
- To avoid boundary effects

Example: Aids data (Age, Time of Death, Time of Diag)



Shingles

- Aids data: conclusions?



Interactive graphics. Text. Visualization

Lecture 5

732A98

Interactive graphics

- Key tool for visual analytics
- Much more efficient than static graphics

Examples:

- Navigation (panning, rotation, zooming)
- Selection (highlighting)
- Connecting (linked views)
- Filtering (sample)
- Reconfiguring (change aesthetics)
- …

Theory of interaction

- Current visualization systems contain certain interaction
  - Limited Features
- Why do we need a theory?
  - To understand what can be interacted and how
  - To see limitations of existing visualization software
  - To be able to propose new relevant interaction tools missing

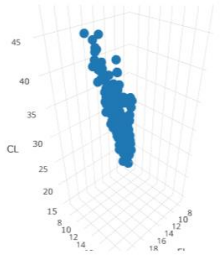
Interaction operators

Navigation operator:

- Camera location
- Viewing direction
- Level of details (e.g. hierarchical representations)

Interaction operators

Example:
Australian crabs

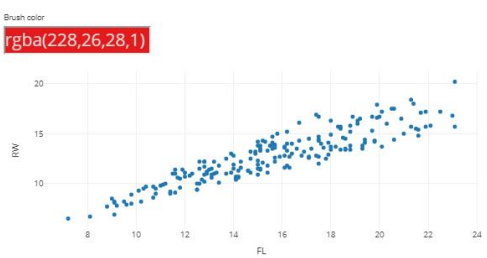


Selection operators

- User isolates a subset of objects
  - Highlighting
  - Masking
  - Focusing
- How to implement?
  - Click
  - Click+hold
  - Bounding box, lasso

Selection operator

Example:
Australian crabs



Connection operators

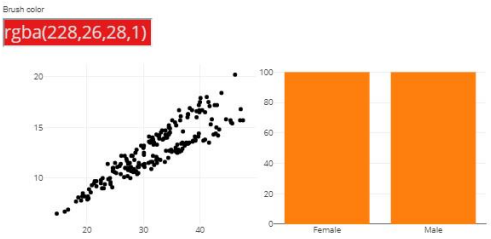
- Related observations are linked in corresponding views
- Selection operator+Connection operator = Brushing
  - Persistent and transient

Connection operators

- Challenges
  - How to link data with various mappings
    - Ex: Histogram and scatter plot
    - Ex2: Contour plot and bar chart
    - Ex3: link in which direction? (Hierarchically connected)
  - How to define the corresponding link (key)?
  - Allow for disconecting views?

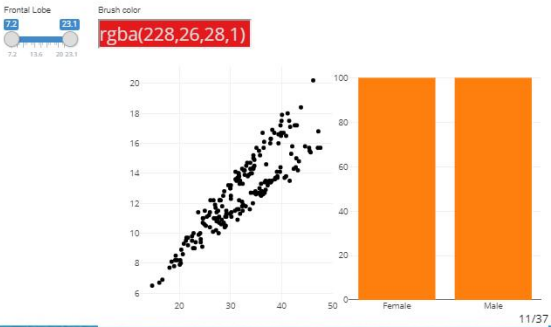
Connection operators

- Australian crabs:
- Which sex do the upper-cluster-crabs have?



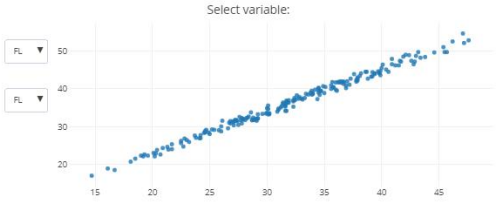
Filtering operators

- Reducing data acc. to specifications



Reconfiguring operators

- Transforming data
  - Sorting rows, reorder columns, MDS
  - Change aesthetics mapping

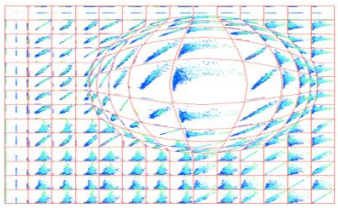


Encoding operators

- Changing the visualization type
- Changing aesthetics
- Another color map
- Change shapes
- …

Abstraction operators

- Distoring objects locally or globally



Interaction operands

- What can the operators be applied to?
- What actions does it imply?

Screen space:

- Navigation: pixel-wise actions
- Selection: sets of pixels (boxes, polygons,…)
- Abstraction: distortion of image (fisheye)
- Filtering: removing some pixels

Interaction operands

Data value space

- Operate observations instead of pixels
- Navigating: translate the axis range
- Zooming: increase/decrease axis range
- Filtering: sample the data, sample dimensions
- Reconfiguring: sorting observations, dimensions, nonlinear transformations

Interaction operands

Data structure space

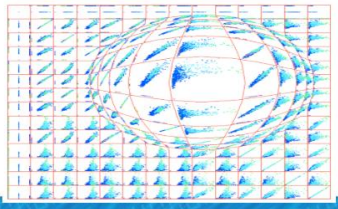
- Different data structures exist: matrix, list, graph,…
- Navigation operator: how navigate the view in the long tree?
- Selection operator: node in the tree is selected -> subbranches must be selected
- Filtering operator: Social network: click on node - nodes that are X links away disappear
- Abstraction/Elaboration: histogram with zooming - recompute bars?

Attribute space

- Working with aesthetics
- Navigation: change range of aesthetics to certain interval (show certain range of colors)
- Encoding operator: change shapes of symbols, non-linear color mapping
- Selection: highlight certain ranges of aesthetics (highlight stars)

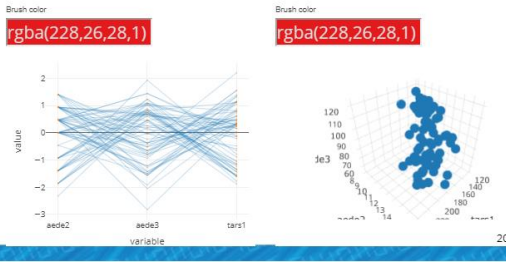
Visualization structure space

- Dashboards, scatter plot matrices
  - How to navigate user in these?
  - Which components of dashboard can user hide, move, rearrange?
  - Distortion of certain elements



Example: flea data

- Measurements of fleas and their types
- Which operators are available here?
- Which clusters do you see?



Interactive visualization in Plotly

Without Shiny: key ingridients

- Create cross-talk objects from your dataframes by `d<-SharedData$new(data, key, group)`
  - key parameter should point to same observations in dataframes
    - Not specified: row id used
  - Group - a unique name related to the same data

Interactive visualization in Plotly

Without Shiny: key ingridients

1. `highlight`-function:
   - applied to Plotly object
   - parameters:
     - on: 'plotly_click', 'plotly_selected',…
     - persistent: TRUE/FALSE
     - dynamic:T/F - enables color selector
     - selectize: T/F text field for selection

Interactive visualization in Plotly

```
d <- SharedData$new(crabs)
scatterCrab <- plot_ly(d, x = ~CL, y = ~RW) %>%
  add_markers(color = I("black"))

barCrab <-plot_ly(d, x=~sex)%>%add_histogram()%>%layout(barmode="overlay")

subplot(scatterCrab,barCrab)%>%
  highlight(on="plotly_select", dynamic=T,
            persistent = T, opacityDim = I(1))%>%
  hide_legend()
```

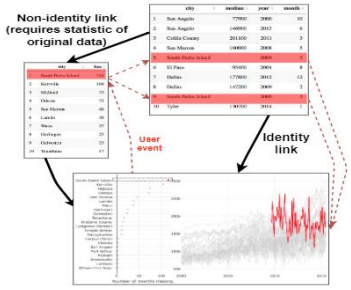

Interactive visualization in Plotly

```
plotly_data(barCrab)
```

```
## # A tibble: 200 x 9
##    species sex   index    FL    RW    CL    CW    BD .crossTalkKey
##  * <fct>   <fct> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <chr>
##  1 Blue    Male      1   8.1   6.7  16.1  19     7   1
##  2 Blue    Male      2   8.8   7.7  18.1  20.8   7.4 2
##  3 Blue    Male      3   9.2   7.8  19    22.4   7.7 3
##  4 Blue    Male      4   9.6   7.9  20.1  23.1   8.2 4
##  5 Blue    Male      5   9.8   8    20.3  23     8.2 5
##  6 Blue    Male      6  10.8   9    23    26.5   9.8 6
##  7 Blue    Male      7  11.1   9.9  23.8  27.1   9.8 7
##  8 Blue    Male      8  11.6   9.1  24.5  28.4  10.4 8
##  9 Blue    Male      9  11.8   9.6  24.2  27.8   9.7 9
## 10 Blue    Male     10  11.8  10.5  25.2  29.3  10.3 10
## # ... with 190 more rows
```

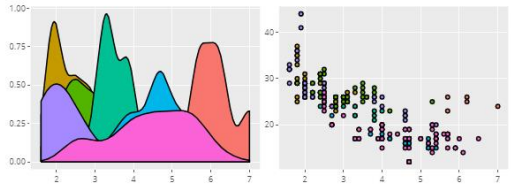
Interactive visualization

- Link can be one-to-many or even dynamic (scatter/barchart)



Interactive visualization

```
m <- SharedData$new(mpg)
p1 <- ggplot(m, aes(displ, fill = class)) + geom_density()
p2 <- ggplot(m, aes(displ, hwy, fill = class)) + geom_point()
subplot(p1, p2) %>% highlight("plotly_click") %>% hide_legend()
```



Text visualization

Applications:

- Articles, books
- Emails, blogs, websites
- Program Logs
- Collections (corpus) of books, blogs,..

Analysis:

- Understanding structure/context of text
- Group similar documents
- Developent of contents/topics over time

Text processing

- Vector Space Model
  - Count occurance of each word in a document
  - Columns: frequencies of words
  - **Term/document matrix** (for collection of documents): columns=words, rows=documents
- TF-IDF model
  - Measures relative importance of word in document
  - $Tf(w)$=frequency of word, $df(w)$=frequency of documents, N=#documents

$$TFIDF(w) = Tf(w) \cdot log(N/df(w))$$

Word cloud

- Words are placed in 2D
- Layout decided algorithmically
- $Tf(w) \rightarrow$ size of words
- Another aesthetics: color

Word cloud

Issues:

- Stopwords need to be removed
- Words sharing the same stem aggregated
- Synonyms
- "Satisfied"/"not satisfied" example
- Incorrect spelling?
- Hyphens and apostrophes
- Size mapping inaccurate (long words)
- Does not show the structure

WordTree

- Root= word of interest
- Branches = contexts the word used
  - Type a combination of words and observe the tree, check its branches

https://www.jasondavies.com/wordtree/

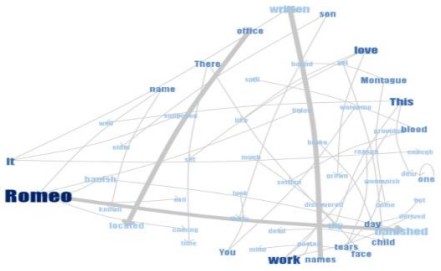
Phrase net

- Graph that analyses co-occurances
  - 'A and B', 'A of B', 'A's B', 'A B', 'A is B',…
- Understanding of context without reading

1. Go https://www.cg.tuwien.ac.at/courses/InfoVis/HallOfFame/2011/Gru
2. Download soft (need Java installed)
3. Upload File
4. Define connector word (is, are, [space], of, and,…)
5. Filter stop words

Phrase net

- Example "Romeo and Julia" (is, are)



Phrase net

- Size of the word = word freq
- Thickness of the connection =co-occurance freq
- Color: dark colors -> word often to the left
- Close on map -> often close in the document

Analysis:

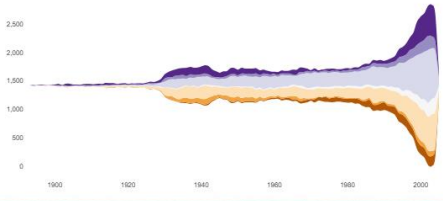
- Analyse most frequent
- Analyse connections and paths
- Analyse strength of paths and intensity of colors
- Click a specific word and look at the respective paths
- Zoom in and out

Steam graphs

- Analyses thematic changes in document collections over time

https://hrbrmstr.github.io/streamgraph/

- Example: #movies by genre,different years
- Perception problem: angles in areas



Text visualization

- Other tools for document collection:
- Place similar documents close in 2D (MDS, clustering, SOM)
- Graph/network visualization can be used for text
- A plenty of other visualizations for text:

http://textvis.lnu.se/

Graph visualization, Animation

Lecture 6

732A98

Graph visualization

- Visualizes relational information
  - Weighted/unweighted, directional/bidirectional
- Applications
  - Social networks
  - Biological networks (genetic)
  - Computer networks
  - Railway/bus networks
  - …

Tree visualization

- Space-filling methods (sunburst chart, treemaps, circle packing)
- Non-space filling methods (trees)
- Aesthetics: node color/ size, link color/size

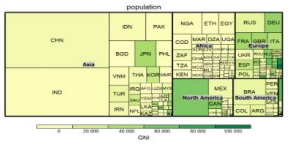
Treemaps, **Circle packing** and **Sunburst diagrams**

- Recursively divides the space into areas according to the hierarchy (e.g. catalogue structure)
- Aesthetics: size and color
- Assuming: $size(parent) = \sum size(kids)$

Treemaps

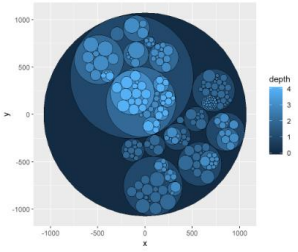
- Analysis: Comparing size/color of hierarchies between/within different levels

Example: World population



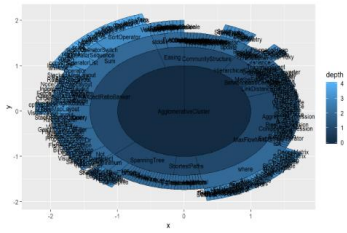
Circle packing

Example: Hierarchy for Flare ActionScript Library



Sunburst

Example: Hierarchy for Flare ActionScript Library



Tree

- Aesthetics: Node/link size, node/link color
- Can be used for classification result analysis
- Analysis
  - Analyse each branch and terminal nodes
  - Compare information between nodes
  - Compare information between branches

Graph Visualization

- Layout has great impact (influences understanding a lot)
- Different layouts proposed
  - Force-directed layout
    - Repulsive and Spring forces
    - Optimization used to find equilibrium
  - Fixed layouts: circular, line,…
  - Minimizing other objectives (e.g. symmetry, edge crossing)

Graph visualization

Example: Network of hyperlinks and mentions among news sources

See Network example 1

- Now adding node_size="audience size", link_size=amount of references

See Network example 2

Graph visualization

Analysis:

- Which nodes are hubs?
- Which links are strongest?
- Which nodes are n steps away from some node?
- Components with strong connections?
  - Relationships to groups?
- How are different groups connected?
- Shortest path between nodes
- Community detection (densely connected nodes)
- Other interesting substructures

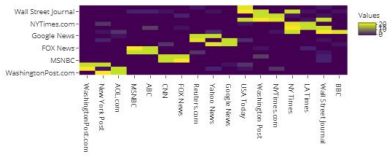
Graph visualization

Example: Community detection based on edge betweeness

See Network example 3

Graph visualization

- Graph can be represented as adjacency matrix
- Seriation methods can be used
- Communities can be detected by heatmaps



Graph Visualization in R

- A plenty of packages available
  - `igraph`: rich functionality, poor graphics
  - `visNetwork`: interactive graph visualization, can use `igraph` objects
  - `ggraph`: static graphs, based on the grammar of graphics, can use `igraph`
  - `ggnet2` in `GGally`: static graphs
  - ...
- Describing graph objects - very package dependent
  - `igraph/visNetwork/ggraph`:
    - Nodes: data frame with "id" column (and other attrs)
    - Edges: data frame with "from" and "to" columns (and other attrs)

Animation

- Eye is drawn to similar motions and outlying motions
- Advantages
  - Effective at attracting attention
  - Time=one extra aesthetics
  - Easily perceived in peripheral vision ->many features can be captured at one time point
- Disadvantages
  - Unappropriate transformation (transition) -> false conclusions
  - Speed of the graphics may hide important details/make boring

Animation- recommendations

- Maintain valid data during transitions
  - Example: using splines
- Be careful when using interpolations: use appropriate models
- Group similar transitions
- Minimize occlusion
- Use simple transitions
- If trajectory is stable, use ease-in, ease-out
- Make transition as long as needed but no longer.

Gapminder

See http://www.gapminder.org

- Gapminder is
  - A database that stores important world statistics (by country)
  - Each dataset can be analysed online with Motion Chart
  - Popularized by Hans Rosling
- Check "Life expectancy" (years) in Gapminder

Gapminder

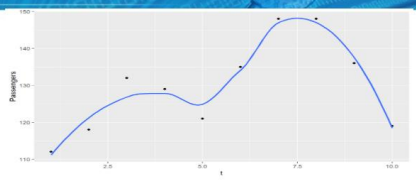
Life expectancy set

- How the minimum and maximum life expectancies changed after 200 years
- Which countries had in general the highest life expectancy? Lowest lifexpectancy? Highest income level? Lowest income level?
- What happened after 1946 and why?
- Compare development of China and USA (mark this countries and use "trace")

How to create an animation?

- Limited amount of timepoints
- Smooth transition is desired
- For each aesthetics x, consider interpolation $x'(t)$
  - How to create?
    - Linear interpolation, splines, kernel smoothers…
    - Compute path length $s(t)$
    - Insert intermediate frames $t_1, \ldots, t_n$ with $x'(t_1), \ldots x'(t_n)$

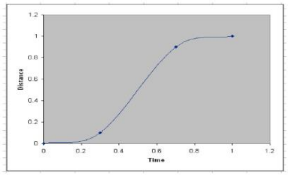
How to create an animation?



How to create an animation?

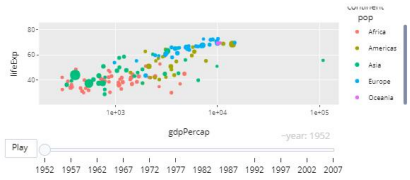
Inserting intermediate values:

- Smooth interpolation
- Ease in, ease out (focuses user on main frames)
- … (https://easings.net/)



Animation in R

- Motion charts : googleVis package
- Plotly: various animation, set time variable to `frame` aesthetics
  - Motion charts are simpler than googleVis



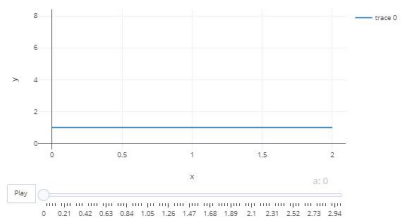
Animation in R

Example: power function

```
library(plotly)

m=matrix(nrow=0,ncol=3)
for (a in seq(0,3,by=0.03)) {
  x<-seq(0,2,0.01)
  y<-x^a
  m<-rbind(m,cbind(x,y,a))
}

df=as.data.frame(m)

plot_ly(df, x=~x, y=~y, frame =~a)%>%add_lines()%>%animation_opts(
  100, easing = "cubic", redraw = F
)
```


Animation in R

Example: power function



2D-tours

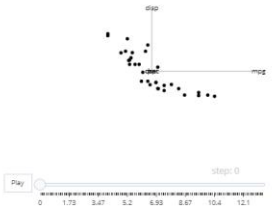
- MDS and PCA projections
  - Is one projection enough to understand data?

2D-tours

- Idea
  - Investigate various projections
  - Connect them into animation
- What to analyze?
  - Same as in scatterplots plus
  - Which variables contribute to projections
    - Patterns in lower dimensional manifold?
- Types of 2d-tours: grand tour and guided tour

2D-tours

- Grand tour for Car data



Projection math

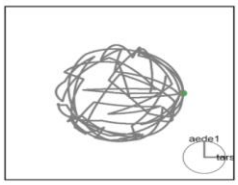
- Data matrix $X = [n \times p]$
- Projection matrix $A = [p \times d]$, $d = 2$
  - Rows in A: contribution of ith coordinate to projection
  - Columns in A: coordinate of projecting vectors in R^p
- Projected data $Z = X \cdot A$
- Example: $X = \begin{pmatrix} 2 & 4 & 3 \\ 6 & 2 & 1 \\ 2 & 9 & 9 \end{pmatrix}$, $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$

Grand tour

- Idea: explore projections randomly

1. Select some projection A_1
2. Select next projection randomly A_2
3. Create shortest path projections $A_1^1, \ldots, A_1^k$ from A_1 to A_2
4. Select next projection randomly A_3
5. …

Grand tour



Guided tour

- Idea: focus on "interesting" projections
  - Interest measure: projection pursuit index $I(XA)$
  - Go through local $\max I(XA)$

- Various PP indices: Holes, central mass,…

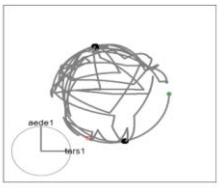
Guided tour

Algorithm:

- Select some projection A_a, calculate I_a
- For i=1,2,…
  - $A_i = A_a + c^i B$, B- random direction, c^i- "cooling parameter"
  - Move to A_i with probability depending on ΔI (and create shortest path intermediate projections)
  - Set $A_a = A_i$ if moved.

Guided tour

- Note: path goes often through same points…



732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Laboratory work 1

Instructions

- Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments.
- **Include all your codes as an appendix into your report; you are also recommended to show parts of the codes in the flowing text of the report.**
- A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes.
- Create a report to the lab solutions in RMarkdown. Make sure that it is can be compiled to HTML and that all paths in RMD file are relative to the current directory where the RMD file is located. **Reports that can not be compiled are returned without revision.**
- Put the RMD file and all supporting files into one ZIP archive when you submit it to LISAM.
- The lab report should be submitted via LISAM before the deadline.

Assignment 1

Sometimes it is necessary to adjust visualizations obtained by complicated R packages and it is difficult to do it programmically. File **tree.pdf** shows a decision tree created by **party** package. Use Inkscape to produce a publication quality graph which is like the one shown in Figure 1 (you may make more improvements if you like!). Note that:

- Terminal nodes are renumbered
- Node numbers and p-values are removed from the non-terminal nodes, title is removed
- Percentages are explicitly added below each terminal node and their positions are adjusted appropriately
- Colors are adjusted

Add the resulting PDF picture to your report.

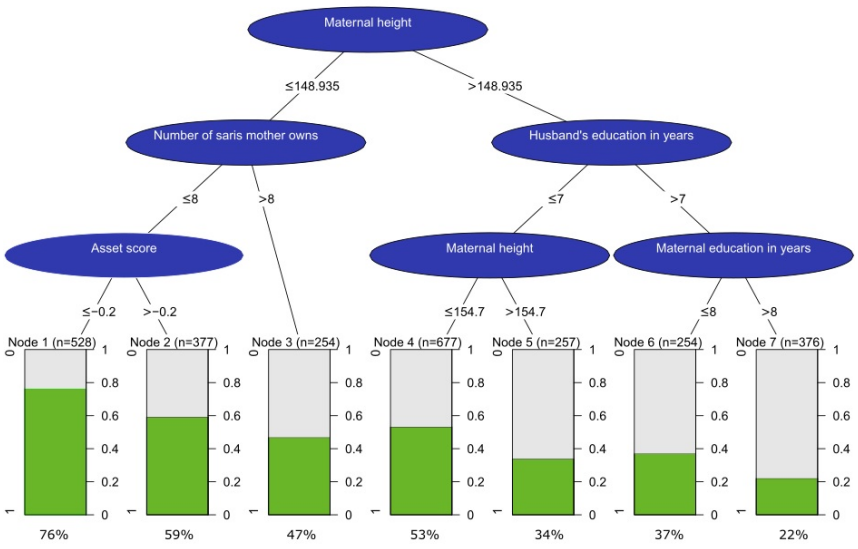


Figure 1. A tree from assignment 1.

732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Assignment 2

Data set SENIC describes the results of measurements taken at different US hospitals. The description of the variables is given in the accompanying document **SENIC.pdf.**

1. Read data from SENIC.txt into R.
2. Create a function that for a given column (vector) X does the following:
    a. It computes first and third quantiles Q1 and Q3 with quantiles()
    b. It returns indices of outlying observations, i.e. observation with X-values greater than Q3+1.5(Q3-Q1) or less than Q1+1.5(Q3-Q1).
3. Use **ggplot2** and the function from step 2 to create a density plot of *Infection risk* in which outliers are plotted as a diamond symbol (◊) . Make some analysis of this graph.
4. Produce graphs of the same kind as in step 3 but for all other quantitative variables in the data (aes_string() can be useful here). Put these graphs into one (hint: arrangeGrob() in gridExtra package can be used) and make some analysis.
5. Create a **ggplot2** scatter plot showing the dependence of *Infection risk* on the *Number of Nurses* where the points are colored by *Number of Beds*. Is there any interesting information in this plot that was not visible in the plots in step 4? What do you think is a possible danger of having such a color scale?
6. Convert graph from step 3 to **Plotly** with *ggplotly* function. What important new functionality have you obtained compared to the graph from step 3? Make some additional analysis of the new graph.
7. Use data plot-pipeline and the pipeline operator to make a histogram of *Infection risk* in which outliers are plotted as a diamond symbol (◊) . Make this plot in the **Plotly** directly (i.e. without using ggplot2 functionality). **Hint**: select(), filter() and is.element() functions might be useful here.
8. Write a **Shiny** app that produces the same kind of plot as in step 4 but in addition include:
    a. Checkboxes indicating for which variables density plots should be produced
    b. A slider changing the bandwidth parameter in the density estimation ('bw' parameter)

Comment how the graphs change with varying bandwidth and which bandwidth value is optimal from your point of view.

Submission procedure

Assume that X is the current lab number, Y is your group number.

If you are neither speaker nor opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline

If you are a speaker for this lab,

- Make sure that you or your group comrade does the following before the deadline:
  - submits the group report using *Lab X* item in the *Submissions* folder before the deadline
  - Goes to Study room *Group Y→Documents* and opens file *Password X.txt*. Then the student should put your group report into ZIP file *Lab X_Group Y.zip* and protect it with a password you found in Password X.txt
  - Uploads the file to *Collaborative workspace →Lab X* folder

If you are opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline
- After the deadline for the lab has passed, go to Collaborative workspace→*Lab X* folder and download the appropriate ZIP file. Open the RMD file in this ZIP file by using the password available in *Course Documents →Password X.txt,* compile it, read it carefully and prepare (in cooperation with your group comrade) **at least three questions/comments/improvement suggestions per lab assignment** in order to put them at the seminar.

732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Laboratory work 2

Instructions

- Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments.
- **Include all your codes as an appendix into your report; you are also recommended to show parts of the codes in the flowing text of the report.**
- A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes.
- Create a report to the lab solutions in RMarkdown. Make sure that it is can be compiled to HTML and that all paths in RMD file are relative to the current directory where the RMD file is located. **Reports that can not be compiled are returned without revision.**
- Put the RMD file and all supporting files into one ZIP archive when you submit it to LISAM.
- The lab report should be submitted via LISAM before the deadline.

Assignment 1. Perception in Visualization

File **olive.csv** contains information about contents of olive oils coming from different regions in Italy. Each observation contains information about

- Region (1=North, 2=South, 3= Sardinia island)
- Area (different Italian regions)

Different acids:

- Palmitic
- …
- Eicosenoic

1. Create a scatterplot in Ggplot2 that shows dependence of Palmitic on Oleic in which observations are colored by Linolenic. Create also a similar scatter plot in which you divide Linolenic variable into fours classes (use cut_interval()) and map the discretized variable to color instead. How easy/difficult is it to analyze each of these plots? What kind of perception problem is demonstrated by this experiment?
2. Create scatterplots of Palmitic vs Oleic in which you map the discretized Linolenic with four classes to:
    a. Color
    b. Size
    c. Orientation angle (use geom_spoke())

    State in which plots it is more difficult to differentiate between the categories and connect your findings to perception metrics (i.e. how many bits can be decoded by a specific aesthetics)
3. Create a scatterplot of Oleic vs Eicosenoic in which color is defined by numeric values of Region. What is wrong with such a plot? Now create a similar kind of plot in which Region is a categorical variable. How quickly can you identify decision boundaries? Does preattentive or attentive mechanism make it possible?

4. Create a scatterplot of Oleic vs Eicosenoic in which color is defined by a discretized Linoleic (3 classes), shape is defined by a discretized Palmitic (3 classes) and size is defined by a discretized Palmitoleic (3 classes). How difficult is it to differentiate between 27=3*3*3 different types of observations? What kind of perception problem is demonstrated by this graph?
5. Create a scatterplot of Oleic vs Eicosenoic in which color is defined by Region, shape is defined by a discretized Palmitic (3 classes) and size is defined by a discretized Palmitoleic (3 classes). Why is it possible to clearly see a decision boundary between Regions despite many aesthetics are used? Explain this phenomenon from the perspective of Treisman's theory.
6. Use Plotly to create a pie chart that shows the proportions of oils coming from different Areas. Hide labels in this plot and keep only hover-on labels. Which problem is demonstrated by this graph?
7. Create a 2d-density contour plot with Ggplot2 in which you show dependence of Linoleic vs Eicosenoic. Compare the graph to the scatterplot using the same variables and comment why this contour plot can be misleading.

Assignment 2. Multidimensional scaling of a high-dimensional dataset

The data set ***baseball-2016.xlsx*** contains information about the scores of baseball teams in USA in 2016, such as:

> Games won, Games Lost, Runs peer game, At bats, Runs, Hits, Doubles, Triples, Home runs, Runs batted in, Bases stolen, Time caught stealing, Bases on Balls, Strikeouts, Hits/At Bats, On Base Percentage, Slugging percentage, On base+Slugging, Total bases, Double plays grounded into, Times hit by pitch, Sacrifice hits, Sacrifice flies, Intentional base on balls, and Runners Left On Base.

1. Load the file to R and answer whether it is reasonable to scale these data in order to perform a multidimensional scaling (MDS).
2. Write an R code that performs a non-metric MDS with Minkowski distance=2 of the data (numerical columns) into two dimensions. Visualize the resulting observations in Plotly as a scatter plot in which observations are colored by League. Does it seem to exist a difference between the leagues according to the plot? Which of the MDS components seem to provide the best differentiation between the Leagues? Which baseball teams seem to be outliers?
3. Use Plotly to create a Shepard plot for the MDS performed and comment about how successful the MDS was. Which observation pairs were hard for the MDS to map successfully?
4. Produce series of scatterplots in which you plot the MDS variable that was the best in the differentiation between the leagues in step 2 against all other numerical variables of the data. Pick up two scatterplots that seem to show the strongest (positive or negative)

connection between the variables and include them into your report. Find some information about these variables in Google – do they appear to be important in scoring the baseball teams? Provide some interpretation for the chosen MDS variable.

Submission procedure

Assume that X is the current lab number, Y is your group number.

If you are neither speaker nor opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline

If you are a speaker for this lab,

- Make sure that you or your group comrade does the following before the deadline:
  - submits the group report using *Lab X* item in the *Submissions* folder before the deadline
  - Goes to Study room *Group Y→Documents* and opens file *Password X.txt*. Then the student should put your group report into ZIP file *Lab X_Group Y.zip* and protect it with a password you found in Password X.txt
  - Uploads the file to *Collaborative workspace →Lab X* folder

If you are opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline
- After the deadline for the lab has passed, go to Collaborative workspace→*Lab X* folder and download the appropriate ZIP file. Open the RMD file in this ZIP file by using the password available in *Course Documents →Password X.txt,* compile it, read it carefully and prepare (in cooperation with your group comrade) **at least three questions/comments/improvement suggestions per lab assignment** in order to put them at the seminar.

732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Laboratory work 3

Instructions

- Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments.
- **Include all your codes as an appendix into your report; you are also recommended to show parts of the codes in the flowing text of the report.**
- A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes.
- Create a report to the lab solutions in RMarkdown. Make sure that it is can be compiled to HTML and that all paths in RMD file are relative to the current directory where the RMD file is located. **Reports that can not be compiled are returned without revision.**
- Put the RMD file and all supporting files into one ZIP archive when you submit it to LISAM.
- The lab report should be submitted via LISAM before the deadline.

Assignment 1. Visualization of mosquito's populations

File **aegypti_albopictus.csv** shows information about the location and detection time of two types of mosquitoes. Both *Aedes aegypti* and *Aedes albopictus* mosquitoes may spread viruses like Zika, dengue, chikungunya and other viruses but *Aedes aegypti* are more likely to spread these viruses (and therefore are more dangerous). The data file contain the following variables:

VECTOR: Identifying the species; Ae. aegypti or Ae. albopictus
LOCATION_TYPE: Whether the record represents a point or a polygon location.
POLYGON_ADMIN: Admin level or polygon size which the record represents when the location type is a polygon. -999 when the location type is a point (5 km x 5 km).
X: The longitudinal coordinate of the point or polygon centroid (WGS1984 Datum).
Y: The latitudinal coordinate of the point or polygon centroid (WGS1984 Datum).
YEAR: The year of the occurrence.
COUNTRY: The name of the country within which the occurrence lies.
COUNTRY_ID: ISO alpha-3 country codes. .
GAUL_AD0: The country-level global administrative unit layer (GAUL) code (see http://www.fao.org/geonetwork) which identifies the Admin-0 polygon within which any smaller polygons and points lie.
STATUS: Established vs. transient populations.

1. Use MapBox interface in Plotly to create two dot maps (for years 2004 and 2013) that show the distribution of the two types of mosquitos in the world (use color to distinguish between mosquitos). Analyze which countries and which regions in these countries had high density of each mosquito type and how the situation changed between these time points. What perception problems can be found in these plots?
2. Compute Z as the numbers of mosquitos per country detected during all study period. Use plot_geo() function to create a choropleth map that shows Z values. This map should have an Equirectangular projection. Why do you think there is so little information in the map?

3. Create the same kind of maps as in step 2 but use
    a. Equidistant projection with choropleth color $\log(Z)$
    b. Conic equal area projection with choropleth color $\log(Z)$

    Analyze the map from step 3a and make conclusions. Compare maps from 3a and 3b and comment which advantages and disadvantages you may see with both types of maps.
4. In order to resolve problems detected in step 1, use data from 2013 only for Brazil and
    a. Create variable X1 by cutting X into 100 piecies (use cut_interval())
    b. Create variable Y1 by cutting Y into 100 piecies (use cut_interval())
    c. Compute mean values of X and Y per group (X1,Y1) and the amount of observations N per group (X1,Y1)
    d. Visualize mean X,Y and N by using MapBox

    Identify regions in Brazil that are most infected by mosquitoes. Did such discretization help in analyzing the distribution of mosquitoes?

Assignment 2. Visualization of income in Swedish households

In this assignment, you will analyze the mean incomes of the Swedish households. Go to http://www.scb.se and choose "English" language, and in the "Search" menu type "Disposable income for households by region, type of households and age", click "Search" and then click at "Statistical Database". Select "Mean value, SEK thousands", all counties, age groups 18-29, 30-49 and 50-64, and year 2016. Download the "Comma delimited without heading" file.

1. Download a relevant map of Swedish counties from http://gadm.org/country and load it into R. Read your data into R and process it in such a way that different age groups are shown in different columns. Let's call these groups Young, Adult and Senior.
2. Create a plot containing three violin plots showing mean income distributions per age group. Analyze this plot and interpret your analysis in terms of income.
3. Create a surface plot showing dependence of Senior incomes on Adult and Young incomes in various counties. What kind of trend can you see and how can this be interpreted? Do you think that linear regression would be suitable to model this dependence?
4. Use Plotly and add_sf() function to visualize incomes of Young and Adults in two choropleth maps. Analyze these maps and make conclusions. Is there any new information that you could not discover in previous statistical plots?
5. Use GPVisualizer http://www.gpsvisualizer.com/geocoder/ and extract the coordinates of Linköping. Add a red dot to the choropleth map for Young from step 4 in order to show where we are located :)

732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Submission procedure

Assume that X is the current lab number, Y is your group number.

If you are neither speaker nor opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline

If you are a speaker for this lab,

- Make sure that you or your group comrade does the following before the deadline:
  - submits the group report using *Lab X* item in the *Submissions* folder before the deadline
  - Goes to Study room *Group Y→Documents* and opens file *Password X.txt*. Then the student should put your group report into ZIP file *Lab X_Group Y.zip* and protect it with a password you found in Password X.txt
  - Uploads the file to *Collaborative workspace →Lab X* folder

If you are opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline
- After the deadline for the lab has passed, go to Collaborative workspace→*Lab X* folder and download the appropriate ZIP file. Open the RMD file in this ZIP file by using the password available in *Course Documents →Password X.txt,* compile it, read it carefully and prepare (in cooperation with your group comrade) **at least three questions/comments/improvement suggestions per lab assignment** in order to put them at the seminar.

732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Laboratory work 4

Instructions

- Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments.
- **Include all your codes as an appendix into your report; you are also recommended to show parts of the codes in the flowing text of the report.**
- A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes.
- Create a report to the lab solutions in RMarkdown. Make sure that it is can be compiled to HTML and that all paths in RMD file are relative to the current directory where the RMD file is located. **Reports that can not be compiled are returned without revision.**
- Put the RMD file and all supporting files into one ZIP archive when you submit it to LISAM.
- The lab report should be submitted via LISAM before the deadline.

Assignment 1. High-dimensional visualization of economic data

File **prices-and-earnings.txt** shows a UBS's (one of the largest banks in the world) report comparing prices, wages, and other economic conditions in cities around the world. Some of the variables measured in 73 cities are Cost of Living, Food Costs, Average Hourly Wage, average number of Working Hours per Year, average number of Vacation Days, hours of work (at the average wage) needed to buy an iPhone, minutes of work needed to buy a Big Mac, and Women's Clothing Cost.

1. For further analysis, import data to R and keep only the columns with the following numbers: 1,2,5,6,7,9,10,16,17,18,19. Use the first column as labels in further analysis.
2. Plot a heatmap of the data without doing any reordering. Is it possible to see clusters, outliers?
3. Compute distance matrices by a) using Euclidian distance and b) as one minus correlation. For both cases, compute orders that optimize Hamiltonian Path Length and use Hierarchical Clustering (HC) as the optimization algorithm. Plot two respective heatmaps and state which plot seems to be easier to analyse and why. Make a detailed analysis of the plot based on Euclidian distance.
4. Compute a permutation that optimizes Hamiltonian Path Length but uses Traveling Salesman Problem (TSP) as solver. Compare the heatmap given by this reordering with the heatmap produced by the HC solver in the previous step – which one seems to be better? Compare also objective function values such as Hamiltonian Path length and Gradient measure achieved by row permutations of TSP and HC solvers (Hint: use criterion() function)
5. Use Ploty to create parallel coordinate plots from unsorted data and try to permute the variables in the plot manually to achieve a better clustering picture. After you are ready with

this, brush clusters by different colors and comment about the properties of the clusters: which variables are important to define these clusters and what values of these variables are specific to each cluster. Can these clusters be interpreted? Find the most prominent outlier and interpret it.

6. Use the data obtained by using the HC solver and create a radar chart diagram with juxtaposed radars. Identify two smaller clusters in your data (choose yourself which ones) and the most distinct outlier.
7. Which of the tools you have used in this assignment (heatmaps, parallel coordinates or radar charts) was best in analyzing these data? From which perspective? (e.g. efficiency, simplicity, etc.)

Assignment 2. Trellis plots for population analysis

File **adult.csv** shown data collected in a population census in 1994. The following metrics are available:

1. age: continuous.
2. workclass: Private, Self-emp-not-inc, etc.
3. fnlwgt: a population index.
4. education: Bachelors, Some-college, etc.
5. education-num: ordered Education variable.
6. marital-status: Married-civ-spouse, Divorced, etc.
7. occupation: Tech-support, Craft-repair, etc.
8. relationship: Wife, Own-child, etc.
9. race: White, Asian-Pac-Islander etc.
10. sex: Female, Male.
11. capital-gain: continuous.
12. capital-loss: continuous.
13. hours-per-week: continuous.
14. native-country: United-States, Cambodia etc.
15. Income level

1. Use ggplot2 to make a scatter plot of Hours per Week versus age where observations are colored by Income level. Why it is problematic to analyze this plot? Make a trellis plot of the same kind where you condition on Income Level. What new conclusions can you make here?
2. Use ggplot2 to create a density plot of age grouped by the Income level. Create a trellis plot of the same kind where you condition on Marital Status. Analyze these two plots and make conclusions.
3. Filter out all observations having Capital loss equal to zero. For the remaining data, use Plotly to create a 3D-scatter plot of Education-num vs Age vs Captial Loss. Why is it difficult to analyze this plot? Create a trellis plot with 6 panels in ggplot2 in which each panel shows a raster-type 2d-density plot of Capital Loss versus Education-num conditioned on values of Age (use cut_number()) . Analyze this plot.

4. Make a trellis plot containing 4 panels where each panel should show a scatter plot of Capital Loss versus Education-num conditioned on the values of Age by a) using cut_number() b) using Shingles with 10% overlap. Which advantages and disadvantages you see in using Shingles?

Submission procedure

Assume that X is the current lab number, Y is your group number.

If you are neither speaker nor opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline

If you are a speaker for this lab,

- Make sure that you or your group comrade does the following before the deadline:
  - submits the group report using *Lab X* item in the *Submissions* folder before the deadline
  - Goes to Study room *Group Y→Documents* and opens file *Password X.txt*. Then the student should put your group report into ZIP file *Lab X_Group Y.zip* and protect it with a password you found in Password X.txt
  - Uploads the file to *Collaborative workspace →Lab X* folder

If you are opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline
- After the deadline for the lab has passed, go to Collaborative workspace→*Lab X* folder and download the appropriate ZIP file. Open the RMD file in this ZIP file by using the password available in *Course Documents →Password X.txt,* compile it, read it carefully and prepare (in cooperation with your group comrade) **at least three questions/comments/improvement suggestions per lab assignment** in order to put them at the seminar.

732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Laboratory work 5

Instructions

- Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments.
- **Include all your codes as an appendix into your report; you are also recommended to show parts of the codes in the flowing text of the report.**
- A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes.
- Create a report to the lab solutions in RMarkdown. Make sure that it is can be compiled to HTML and that all paths in RMD file are relative to the current directory where the RMD file is located. **Reports that can not be compiled are returned without revision.**
- Put the RMD file and all supporting files into one ZIP archive when you submit it to LISAM.
- The lab report should be submitted via LISAM before the deadline.

Assignment 1. Text Visualization of Amazon reviews

In this assignment you will analyze feedbacks given by customers for watches Casio AMW320R-1EV bought at www.amazon.com . Files Five.txt and OneTwo.txt contain feedbacks of the customers who were pleased and not pleased with their buy, respectively.

1. Use R tools to create a word cloud corresponding to Five.txt and OneTwo.txt and adjust the colors in the way you like. Analyze the graphs.
2. Run Phrase Nets as follows:
    a. Download the software from https://www.cg.tuwien.ac.at/courses/InfoVis/HallOfFame/2011/Gruppe08/Homepage/
    b. Use the file phrase-net.zip that you have downloaded and unpack it somewhere
        i. If you are using Windows
            1. Launch the command line environment in Windows by opening the start menu in Windows and typing *cmd* in the search field and change the current directory to the "phrase-nets" directory with this kind of commands:
            ```
            z:
            cd lab5\phrase-nets
            ```
            2. In the command line environment, copy and paste the contents of run.txt
        ii. If you are using Linux
            1. Change your working directory to "phrase-nets" directory using Terminal
            2. Run "run.sh"

Note that you need Java to run this software:
https://www.java.com/en/download/

Create the phrase nets for Five.Txt and One.Txt with connector words

- am, is, are, was, were
- a, the
- at
- of

where you choose ”Filter stopwords” option.

3. When you find an interesting connection between some words, use Word Trees https://www.jasondavies.com/wordtree/ to understand the context better. Note that this link might not work properly in Microsoft Edge (if you are using Windows 10) so use other browsers.

Analyze the graphs obtained and comment on the most interesting findings, like:

- Which properties of this watch are mentioned mostly often?
- What are satisfied customers talking about?
- What are unsatisfied customers talking about?
- What are good and bad properties of the watch mentioned by both groups?
- Can you understand watch characteristics (like type of display, features of the watches) by observing these graphs?

Assignment 2. Interactive analysis of Italian olive oils.

In this assignment, you will continue analyzing data **olive.csv** that you started working with in lab 2. These data contain information about contents of olive oils coming from different regions in Italy. Each observation contains information about

- Region (1=North, 2=South, 3= Sardinia island)
- Area (different Italian regions)

Different acids:

- Palmitic
- …
- Eicosenoic

ATTN: All diagrams that support your judgments should be included to the report

In this assignment, you are assumed to use Plotly without Shiny.

1. Create an interactive scatter plot of the eicosenoic against linoleic. You have probably found a group of observations having unusually low values of eicosenoic. Hover on these observations to find out the exact values of eicosenoic for these observations.
2. Link the scatterplot of (eicosenoic, linoleic) to a bar chart showing Region and a slider that allows to filter the data by the values of stearic. Use persistent brushing to identify

the regions that correspond unusually low values of eicosenoic. Use the slider and describe what additional relationships in the data can be found by using it. Report which interaction operators were used in this step.

3. Create linked scatter plots eicosenoic against linoleic and arachidic against linolenic. Which outliers in (arachidic, linolenic) are also outliers in (eicosenoic, linoleic)? Are outliers grouped in some way? Use brushing to demonstrate your findings.
4. Create a parallel coordinate plot for the available eight acids, a linked 3d-scatter plot in which variables are selected by three additional drop boxes and a linked bar chart showing Regions. Use persistent brushing to mark each region by a different color. Observe the parallel coordinate plot and state which three variables (let's call them influential variables) seem to be mostly reasonable to pick up if one wants to differentiate between the regions. Does the parallel coordinate plot demonstrate that there are clusters among the observations that belong to the same Region? Select the three influential variables in the drop boxes and observe in the 3d-plot whether each Region corresponds to one cluster.
5. Think about which interaction operators are available in step 4 and what interaction operands they are be applied to. Which additional interaction operators can be added to the visualization in step 4 to make it even more efficient/flexible? Based on the analysis in the previous steps, try to suggest a strategy (or, maybe, several strategies) that would use information about the level of acids to discover which regions different oils comes from.

Submission procedure

Assume that X is the current lab number, Y is your group number.

If you are neither speaker nor opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline

If you are a speaker for this lab,

- Make sure that you or your group comrade does the following before the deadline:
  - submits the group report using *Lab X* item in the *Submissions* folder before the deadline
  - Goes to Study room *Group Y→Documents* and opens file *Password X.txt*. Then the student should put your group report into ZIP file *Lab X_Group Y.zip* and protect it with a password you found in Password X.txt
  - Uploads the file to *Collaborative workspace →Lab X* folder

If you are opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline

- After the deadline for the lab has passed, go to Collaborative workspace→*Lab X* folder and download the appropriate ZIP file. Open the RMD file in this ZIP file by using the password available in *Course Documents→Password X.txt,* compile it, read it carefully and prepare (in cooperation with your group comrade) **at least three questions/comments/improvement suggestions per lab assignment** in order to put them at the seminar.

732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Laboratory work 6

Instructions

- Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments.
- **Include all your codes as an appendix into your report; you are also recommended to show parts of the codes in the flowing text of the report.**
- A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes.
- Create a report to the lab solutions in RMarkdown. Make sure that it is can be compiled to HTML and that all paths in RMD file are relative to the current directory where the RMD file is located. **Reports that can not be compiled are returned without revision.**
- Put the RMD file and all supporting files into one ZIP archive when you submit it to LISAM.
- The lab report should be submitted via LISAM before the deadline.

Assignment 1. Network visualization of terrorist connections

Files **trainData.dat** and **trainMeta.dat** contain information about a network of the individuals involved in the bombing of commuter trains in Madrid on March 11, 2004. The names included were of those people suspected of having participated and their relatives.

File **trainMeta.dat** contains the names of individuals (first column) and Bombing group (second column) which shows "1" if person participated in placing the explosives and "0" otherwise. According to the order in this file, persons were enumerated 1-70.

File **trainData.dat** contains information about connections between the individuals (first two columns) and strength of ties linking (from one to four):

1. Trust--friendship (contact, kinship, links in the telephone center).
2. Ties to Al Qaeda and to Osama Bin Laden.
3. Co-participation in training camps and/or wars.
4. Co-participation in previous terrorist Attacks (Sept 11, Casablanca).

1. Use visNetwork package to plot the graph in which
    a. you use strength of links variable
    b. nodes are colored by Bombing Group.
    c. size of nodes is proportional to the number of connections (function strength() from IGRAPH might be useful here)
    d. you use a layout that optimizes repulsion forces (visPhysics(solver="repulsion")).
    e. all nodes that are connected to a currently selected node by a path of length one are highlighted

    Analyse the obtained network, in particular describe which clusters you see in the network.

2. Add a functionality to the plot in step 1 that highlights all nodes that are connected to the selected node by a path of length one or two. Check some amount of the largest nodes and comment which individual has the best opportunity to spread the information in the network. Read some information about this person in Google and present your findings.
3. Compute clusters by optimizing edge betweenness and visualize the resulting network. Comment whether the clusters you identified manually in step 1 were also discovered by this clustering method.
4. Use adjacency matrix representation to perform a permutation by Hierarchical Clustering (HC) seriation method and visualize the graph as a heatmap. Find the most pronounced cluster and comment whether this cluster was discovered in steps 1 or 3.

Assignment 2. Animations of time series data.

The data file ***Oilcoal.csv*** provides time series about the consumption of oil (million tonnes) and coal (million tonnes oil equivalents) in China, India, Japan, US, Brazil, UK, Germany and France. Marker size shows how large a country is (1 for China and the US, 0.5 for all other countries).

1. Visualize data in Plotly as an animated bubble chart of Coal versus Oil in which the bubble size corresponds to the country size. List several noteworthy features of the investigated animation.
2. Find two countries that had similar motion patterns and create a motion chart including these countries only. Try to find historical facts that could explain some of the sudden changes in the animation behavior.
3. Compute a new column that shows the proportion of fuel consumption related to oil: $Oil_p = \frac{Oil}{Oil+Coal} * 100$. One could think of visualizing the proportions Oil_p by means of animated bar charts; however smooth transitions between bars are not yet implemented in Plotly. Thus, use the following workaround:
    a. Create a new data frame that for each year and country contains two rows: one that shows Oil_p and another row containing 0 in Oil_p column
    b. Make an animated line plot of Oil_p versus Country where you group lines by Country and make them thicker

    Perform an analysis of this animation. What are the advantages of visualizing data in this way compared to the animated bubble chart? What are the disadvantages?
4. Repeat the previous step but use "elastic" transition (easing). Which advantages and disadvantages can you see with this animation? Use information in https://easings.net/ to support your arguments.
5. Use Plotly to create a guided 2D-tour visualizing Coal consumption in which the index function is given by Central Mass index and in which observations are years and variables are different countries. Find a projection with the most compact and well-separated clusters. Do clusters correspond to different Year ranges? Which variable has the largest contribution to this projection? How can this be interpreted? (Hint: make a time series plot for the Coal consumption of this country)

732A98 Visualization
Division of Statistics and Machine Learning
Department of Computer and Information Science

Submission procedure

Assume that X is the current lab number, Y is your group number.

If you are neither speaker nor opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline

If you are a speaker for this lab,

- Make sure that you or your group comrade does the following before the deadline:
  - submits the group report using *Lab X* item in the *Submissions* folder before the deadline
  - Goes to Study room *Group Y→Documents* and opens file *Password X.txt*. Then the student should put your group report into ZIP file *Lab X_Group Y.zip* and protect it with a password you found in Password X.txt
  - Uploads the file to *Collaborative workspace →Lab X* folder

If you are opponent for this lab,

- Make sure that you or your group comrade submits the group report using *Lab X* item in the *Submissions* folder before the deadline
- After the deadline for the lab has passed, go to Collaborative workspace→*Lab X* folder and download the appropriate ZIP file. Open the RMD file in this ZIP file by using the password available in *Course Documents →Password X.txt,* compile it, read it carefully and prepare (in cooperation with your group comrade) **at least three questions/comments/improvement suggestions per lab assignment** in order to put them at the seminar.

2.1 Types of Data

In its simplest form, each observation or variable of a data record represents a single piece of information. We can categorize this information as being *ordinal* (numeric) or *nominal* (nonnumeric). Subcategories of each can be readily defined.

Ordinal. The data take on numeric values:

- binary—assuming only values of 0 and 1;
- discrete—taking on only integer values or from a specific subset (e.g., (2, 4, 6));
- continuous—representing real values (e.g., in the interval [0, 5]).

Nominal. The data take on nonnumeric values:

- categorical—a value selected from a finite (often short) list of possibilities (e.g., red, blue, green);
- ranked—a categorical variable that has an implied ordering (e.g., small, medium, large);
- arbitrary—a variable with a potentially infinite range of values with no implied ordering (e.g., addresses).

Another method of categorizing variables is by using the mathematical concept of *scale*.

Scale. Three attributes that define a variable's measure are as follows:

- Ordering relation, with which the data can be ordered in some fashion. By definition, ranked nominal variables and all ordinal variables exhibit this relation.
- Distance metric, with which the distances can be computed between different records. This measure is clearly present in all ordinal variables, but is generally not found in nominal variables.

2.3.6 Dimension Reduction

In situations where the dimensionality of the data exceeds the capabilities of the visualization technique, it is necessary to investigate ways to reduce the data dimensionality, while at the same time preserving, as much as possible, the information contained within. This can be done manually by allowing the user to select the dimensions deemed most important, or via computational techniques, such as *principal component analysis* (PCA) [385], *multidimensional scaling* (MDS) [259], *Kohonen self-organizing maps* (SOMs) [248], and *Local Linear Embedding* (LLE) [350]. Each of these can be used to convey, within the dimensionality of the display, a description of the data set that covers the majority of significant features, such as clusters, patterns, and outliers. However, it is important to note that most of these techniques do not produce unique results. Starting configurations, parameters of the

calculations, and variations in the computations can lead to quite different results, as we will encounter in the aggregation techniques. In Figure 2.4 we can see the results of reducing a four-dimensional data set to two dimensions using PCA and plotting the resulting points. In fact, we use a form of glyph, called a *star glyph*, to convey the original data points. Clearly, the PCA algorithm does a good job of separating data into clusters.

Principal component analysis (PCA) is a popular method for dimension reduction [385]. PCA is a data-space transformation technique that computes new dimensions/attributes which are linear combinations of the original data attributes. The advantage of the new dimensions is that they

PCA algorithm does a good job of separating data into clusters.

Principal component analysis (PCA) is a popular method for dimension reduction [385]. PCA is a data-space transformation technique that computes new dimensions/attributes which are linear combinations of the original data attributes. The advantage of the new dimensions is that they can be sorted according to their contribution in explaining the variance of the data. By selecting the most relevant new dimensions, a subspace of variables is obtained that minimizes the average error of lost information. An intuitive description of PCA is as follows:

1. Select a line in space that spreads the projected n-dimensional data the most. This line represents the first principal component.
2. Select a line perpendicular to the first that now spreads the data the most. That line is the second PCA.

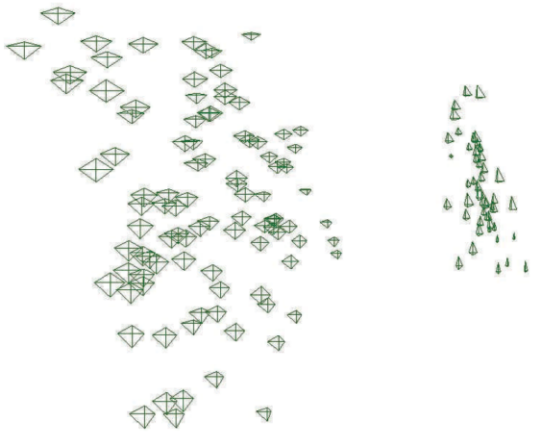


Figure 2.4. The Iris data set in star glyphs, with the position of each point based on the first two principal components. The star glyph represents four variables as the lengths of the each of the lines emanating from the center of a four-pointed star. Reasonable clustering can be seen.

3. Repeat until all PC dimensions are computed, or until the desired number of PCs have been obtained.

More formally, the steps in computing the PCA are as follows (from [385]):

1. Assume the data has m dimensions/attributes. For each member of a record, subtract the mean value for that dimension, resulting in a data set with mean=0.
2. Compute the covariance matrix (see any statistics book).
3. Compute the eigenvectors and eigenvalues of the covariance matrix.
4. Sort the eigenvectors based on their eigenvalues, from largest to smallest.
5. Select the first m_r eigenvectors, where m_r is the number of dimensions you want to reduce your data set to.
6. Create a matrix of these eigenvectors as the rows, with the first one being the top row of the matrix.
7. For each data record, create a vector of its values, transpose it, and premultiply it with the matrix above. This completes the transformation; each data record is now represented in the reduced dimension space.

As another example of a dimension reduction process, we present below the steps for computing a form of multidimensional scaling known as a *gradient descent approach*. Briefly, MDS tries to find a lower dimensional representation for a data set that best preserves the inter-point distances from the original data set. In other words, we wish to minimize, for every pair of points (i, j), the difference between d_{ij} (the distance between the points in N dimensions) and δ_{ij} (the distance between the points in the reduced space). This discrepancy is usually referred to as the *stress* of the configuration. The procedure is as follows:

1. Compute the pairwise distances in n-dimensional space. If there are n data points, this requires the calculation of $n(n-1)/2$ distances.
2. Assign all data point locations (often random) in the lower dimensional space.
3. Calculate the stress of the configuration (a normalized, signed value or a mean-squared error are just two of the possibilities).

4. If the average or accumulated stress is less than a user-prescribed threshold, terminate the process and output the results.
5. If not, for each data point, compute a vector indicating the direction in which it should move to reduce the stress between it and all other data points. This would be a weighted sum of vectors between this point and all its neighbors, pointing either toward or away from the neighbors and weighted proportional to the pairwise stress. Thus, positive stresses would move the points away from each other, and negative stresses
6. Move each data point in the lower dimensional space, according to the vector computed, and return to step 3.

Care must be taken to avoid infinite loops caused by points overshooting their "best" configuration. Likewise, as in all optimization processes, the algorithm can become stuck in a local optimum. This can be caused by *blockages*, where some points that should be close together are caught on opposite sides of a repulsive force (points that are trying to maintain a long distance from the points in question). In this case, repeating the algorithm with different starting conditions, or allowing, as in simulated annealing [261], an occasional *jump* with a magnitude and/or direction different from the vector calculated, to enable a point to clear such blockages if they exist, may improve the solution. There are also a number of non-linear dimension reduction techniques which use a non-linear mapping from high- to low-dimensional space. Two widely used methods are emphSelf-organizing maps (SOMs) [248] and *Local Linear Embeddings* (LLEs) [350].

Many visualization and statistical graphics packages include both MDS and PCA implementations. In fact, some will even use PCA to compute initial positions to be used in the MDS process, which can greatly reduce the number of iterations required to converge on a low-stress configuration. It is worthwhile to note that there are examples of higher-dimensional data where the first few principal components do not separate the data well, but the later principal components with small eigenvalues do.

```
###app1
library(shiny)


ui <- fluidPage(
   sliderInput(inputId="ws", label="Choose bandwidth size", value=0.01, min=0.1, max=1),
   plotOutput("densPlot")
)

server <- function(input, output) {

   output$densPlot <- renderPlot({
     ggplot(iris, aes(x=Sepal.Length,fill=Species))+
       stat_density(alpha=0.8, bw=input$ws, position="identity")
   })
}

# Run the application
shinyApp(ui = ui, server = server)


###app2
library(shiny)


ui <- fluidPage(
   actionButton(inputId = "add",label = "Add one class"),
   actionButton(inputId = "rem",label = "Remove one class"),
   plotOutput("densPlot")
)

server <- function(input, output) {
  nclass <- reactiveValues(n=1)
  observeEvent(input$add, {nclass$n <- nclass$n+1})
  observeEvent(input$rem, {nclass$n <- nclass$n-1})

   output$densPlot <- renderPlot({
     iris1=iris[as.numeric(iris$Species)<=nclass$n,]
     ggplot(iris1, aes(x=Sepal.Length,fill=Species))+
       stat_density()
   })
}

# Run the application
shinyApp(ui = ui, server = server)

###barChart
library(dplyr)
library(ggplot2)
library(plotly)

#Bar chart ggplot2
p2<-ggplot(mtcars, aes(cyl))+ geom_bar(fill="orange")
p2

#bar chart ggplot2 another way- any other summary can be computed.
c1<-mtcars %>%   group_by(cyl) %>%   summarise(ncyl = n())
p1<- ggplot(c1, aes(cyl, ncyl))+
  geom_bar(stat = "identity", fill="orange")+
  labs(x="# Cylinders", y="Amount")
print(p1)
```

```
#Pie chart ggplot2
c2<-c1
c2$ncyl=c2$ncyl/sum(c2$ncyl)*100
p3<- ggplot(c2, aes(x="", y=ncyl, fill=factor(cyl)))+
  geom_bar(width=1,stat = "identity" )+
  labs(x=NULL, y=NULL, fill="#Cylinders")+
  coord_polar(theta="y")

print(p3)

#Bar chart with 2 categories
c3<- mtcars%>% group_by(gear, cyl) %>% summarize( count=n())

p4<- ggplot(c3, aes(gear, count,fill=factor(cyl)))+
  geom_bar(stat="identity", position="dodge")+
  labs(fill="#Cylinders")
p4

## NOW in plotly

#Bar chart
c1<-mtcars %>%   group_by(cyl) %>%   summarise(ncyl = n())
p<- plot_ly(data=c1, x=~cyl, y=~ncyl)%>%
  add_bars()%>%layout(xaxis=list(title="Cylinders"), yaxis=list(title="Count"))
p

# Bar chart wth two categories

c3<- mtcars%>% group_by(gear, cyl) %>% summarize( count=n())

p<- plot_ly(data=c3, x=~cyl, y=~count, color=~factor(gear))%>%
  add_bars()%>%
  layout(xaxis=list(title="Cylinders"), yaxis=list(title="Count"), barmode='group')
p

#Pie chart plotly
c2<-c1
c2$ncyl=c2$ncyl/sum(c2$ncyl)*100
p3<- plot_ly(c2, labels=~cyl, values=~ncyl)%>%
  add_pie()

print(p3)


###metricMDS
library(plotly)

music = read.csv("music-sub.csv", row.names=1)
music.numeric= scale(music[,4:7])
d=dist(music.numeric)
coords=cmdscale(d,2)
coordsMDS=as.data.frame(coords)
coordsMDS$name=rownames(coordsMDS)

plot_ly(coordsMDS, x=~V1, y=~V2, type="scatter", hovertext=~name)

###nonMetricMDS
library(plotly)
library(MASS)

music = read.csv("music-sub.csv", row.names=1)
music.numeric= scale(music[,4:7])
d=dist(music.numeric)
```

```
res=isoMDS(d,k=3)
coords=res$points

coordsMDS=as.data.frame(coords)
coordsMDS$name=rownames(coordsMDS)
coordsMDS$artist=music$artist

plot_ly(coordsMDS, x=~V1, y=~V2, z=~V3, type="scatter3d", hovertext=~name, color=
~artist)

###shepard
sh <- Shepard(d, coords)
delta <-as.numeric(d)
D<- as.numeric(dist(coords))

n=nrow(coords)
index=matrix(1:n, nrow=n, ncol=n)
index1=as.numeric(index[lower.tri(index)])

n=nrow(coords)
index=matrix(1:n, nrow=n, ncol=n, byrow = T)
index2=as.numeric(index[lower.tri(index)])



plot_ly()%>%
  add_markers(x=~delta, y=~D, hoverinfo = 'text',
        text = ~paste('Obj1: ', rownames(music)[index1],
                      '<br> Obj 2: ', rownames(music)[index2]))%>%
  #if nonmetric MDS inolved
  add_lines(x=~sh$x, y=~sh$yf)

###densityPlots
library(dplyr)
library(ggplot2)
library(plotly)

#Ggplot2: histogram, density and violin plot

p1<-ggplot(mtcars, aes(mpg, fill=factor(am)))+geom_density(alpha=0.2)
p1

p2<-ggplot(mtcars, aes(mpg, fill=factor(am)))+geom_histogram(bins = 5)
p2

p3<-ggplot(mtcars, aes( y=mpg, x=factor(am)))+geom_violin(fill="orange")
p3

#Plotly: histogram, density and violin plot

p1<-plot_ly(mtcars, x=~mpg, color=~factor(am), alpha=0.6)%>%
  add_histogram()%>%
  layout(barmode = "overlay")

p1


#easiest from ggplot2:
p2<-ggplot(mtcars, aes(mpg, fill=factor(am)))+geom_density(alpha=0.6)
ggplotly(p2)

# Violin plot
```

```
p3<-plot_ly(mtcars, x=~factor(am), y=~mpg, split=~factor(am),
            type="violin", box=list(visible=T))
p3

###splom
library(dplyr)
library(ggplot2)

library(plotly)

#plotly

mtcars %>% plot_ly()%>%
  add_trace(type='splom',     dimensions = list(
    list(label='MPG', values=~mpg),
    list(label='Displacement', values=~disp),
    list(label='#Horse Power', values=~hp),
    list(label='weight', values=~wt)),
    marker = list(
      color = as.integer(mtcars$cyl),
      size = ~qsec

    )

)

#ggplot2

library(GGally)
ggpairs(mtcars, columns=c(1,3,4,6),
        mapping =aes(color=factor(cyl), size=qsec))

###surfaceContour
library(dplyr)
library(ggplot2)
library(MASS)

library(plotly)

mtcars %>%plot_ly(x=~mpg, y=~disp, z=~wt, type="scatter3d")
mtcars %>%plot_ly(x=~mpg, y=~disp, z=~wt, type="contour")

library(akima)
attach(mtcars)
s=interp(mpg,disp,wt, duplicate = "mean")
detach(mtcars)

plot_ly(x=~s$x, y=~s$y, z=~s$z, type="surface")

###choropleth
library(plotly)

df <-
read.csv('https://raw.githubusercontent.com/plotly/datasets/master/2014_world_gdp_with_co
des.csv')



# WORLD MAP
g <- list(
  projection = list(type = 'Mercator')
)
```

```
p <- plot_geo(df) %>%
  add_trace(
    z = ~GDP..BILLIONS., color = ~GDP..BILLIONS., colors = 'Blues',
    text = ~COUNTRY, locations = ~CODE
  ) %>%
    layout(
    geo = g
  )

p

####Other country maps

rds<-readRDS("gadm36_GBR_2_sf.rds")
df<-read.csv("GBcities.csv")

rownames(df)=df$name
rds$Price=df[rds$NAME_2, "Price"]
#Data for some regions absent, setting to 0
rds$Price[is.na(rds$Price)]=0

#plotly
p<-plot_ly()%>%add_sf(data=rds, split=~NAME_2, color=~Price, showlegend=F, alpha=1)
p
#ggplot2
p<-ggplot(rds) +  geom_sf(aes(fill = Price))
ggplotly(p)

#mapbox

p<-plot_mapbox(rds, split=~NAME_2, color=~Price, showlegend=F, alpha=0.5)
p

###lineMaps
library(plotly)
library(dplyr)
# airport locations
air <-
read.csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_february_us_airpo
rt_traffic.csv')
# flights between airports
flights <-
read.csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_february_aa_fligh
t_paths.csv')
flights$id <- seq_len(nrow(flights))

# map projection
geo <- list(
  scope = 'north america'
)

p <- plot_geo(locationmode = 'USA-states', color = I("red")) %>%
  add_markers(
    data = air, x = ~long, y = ~lat, text = ~airport,
    size = ~cnt, hoverinfo = "text", alpha = 0.5
  ) %>%
  add_segments(
    data = group_by(flights, id),
    x = ~start_lon, xend = ~end_lon,
    y = ~start_lat, yend = ~end_lat,
    alpha = 0.3, size = I(1) , hoverinfo = "none"
  ) %>%
  layout(
```

```
    geo = geo
  )

p

###symbol_dotmap
library(plotly)
df <-
read.csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_february_us_airpo
rt_traffic.csv')


g <- list(
  scope = 'usa'
)

p <- plot_geo(df, lat = ~lat, lon = ~long) %>%
  add_markers(
    text = ~paste(airport, city, state, paste("Arrivals:", cnt), sep = "<br />"),
    color = ~cnt, symbol = I("diamond"), hoverinfo = "text",
    colors=colorRamp(c("lightblue","darkblue"))
  )  %>%
  layout(geo = g)
p


#size adjustment

p <- plot_geo(df, lat = ~lat, lon = ~long) %>%
  add_markers(
    text = ~paste(airport, city, state, paste("Arrivals:", cnt), sep = "<br />"),
    size = ~cnt, symbol = I("diamond"), hoverinfo = "text",
    colors=colorRamp(c("lightblue","darkblue"))
  )  %>%
  layout(geo = g)
p

## Other countries than USA- ggplot2

##Example -UK

rds<-readRDS("gadm36_GBR_0_sf.rds")
df<-read.csv("GBcities.csv")
p<-ggplot()+geom_sf(data=rds)+
  geom_point(data=df, mapping = aes(longitude, latitude, color=Price))



p
##Use google chrome- does not work in explorer.

ggplotly(p)


##Now with mapbox

p <- df %>%
  plot_mapbox(lat = ~latitude, lon = ~longitude,
              size = ~Price,
              mode = 'scattermapbox')
p

###seriation
library(plotly)
```

```
library(seriation)

mtscaled=scale(mtcars[1:7])
rowdist<-dist(mtscaled)
coldist<-dist(t(mtscaled))

order1<-seriate(rowdist, "BBURCG")
order2<-seriate(coldist, "BBURCG")
ord1<-get_order(order1)
ord2<-get_order(order2)

reordmatr<-mtscaled[rev(ord1),ord2]

dims=list()
for( i in 1:ncol(reordmatr)){
  dims[[i]]=list( label=colnames(reordmatr)[i],
    values=as.formula(paste("~",colnames(reordmatr)[i])))
}

dims0=list()
for( i in 1:ncol(mtscaled)){
  dims0[[i]]=list( label=colnames(mtscaled)[i],
                  values=as.formula(paste("~",colnames(mtscaled)[i])))
}

p <- as.data.frame(mtscaled) %>%
  plot_ly(type = 'parcoords',
          #line = list(color = ~as.numeric(Species)),

          dimensions = dims0
  )

p

p1 <- as.data.frame(reordmatr) %>%
  plot_ly(type = 'parcoords',
          #line = list(color = ~as.numeric(Species)),

          dimensions = dims
  )

p1


plot_ly(x=colnames(reordmatr), y=rownames(reordmatr),
        z=reordmatr, type="heatmap", colors =colorRamp(c("yellow", "red")))

###radarChart
library(plotly)

mtscaled=scale(mtcars)

#Superimposed


p<- plot_ly(type='scatterpolar', fill = 'toself')

for (i in 1:3){
  p<-p%>% add_trace(r=c(mtscaled[i,], mtscaled[i,1]),
                    theta=c(colnames(mtscaled), colnames(mtscaled)[1]),
                    name=rownames(mtscaled)[i])
}
```

```
p <- p%>% layout(
  polar = list(
    radialaxis = list(
      visible = T
    )
  )
)

p

#Juxtaposed

#Ugly graphics
stars(mtcars,key.loc=c(15,2), draw.segments=F, col.stars =rep("Yellow", nrow(mtcars)))

##ggplot2
library(scales)
mtcars1<-mtcars%>% mutate_all(funs(rescale))
mtcars1$name=rownames(mtcars)
mtcars2 <- mtcars1%>%tidyr::gather(variable, value, -name, factor_key=T)%>%arrange(name)
p<-mtcars2 %>%
  ggplot(aes(x=variable, y=value, group=name)) +
  geom_polygon(fill="blue") +
  coord_polar() + theme_bw() + facet_wrap(~ name) +
  theme(axis.text.x = element_text(size = 5))
p

## Plotly

library(plotly)
library(dplyr)
library(scales)

Ps=list()
nPlot=10#nrow(mtcars)

mtcars %>%
  add_rownames( var = "group" ) %>%
  mutate_each(funs(rescale), -group) -> mtcars_radar

for (i in 1:nPlot){
  Ps[[i]] <- htmltools::tags$div(
    plot_ly(type = 'scatterpolar',
            r=as.numeric(mtcars_radar[i,-1]),
            theta= colnames(mtcars_radar)[-1],
            fill="toself")%>%
      layout(title=mtcars_radar$group[i]), style="width: 25%;")
}

h <-htmltools::tags$div(style = "display: flex; flex-wrap: wrap", Ps)

htmltools::browsable(h)

###parallelCoord
library(ggplot2)
library(plotly)


#Plotly - can not see observation ID
data=iris
p <- data %>%
  plot_ly(type = 'parcoords',
          #line = list(color = ~as.numeric(Species)),
```

```
          dimensions = list(
            list(label = 'Sepal Width', values = ~Sepal.Width),
            list(label = 'Sepal Length', values = ~Sepal.Length),
            list(label = 'Petal Length', values = ~Petal.Length)
          )
  )

p


## GGplot2 - can see observation ID, no interactivity
library(GGally)

obj<- ggparcoord(iris, columns=1:3)
ggplotly(obj)

#coloring lines
data=iris
data$Col=as.numeric(scale(data$Petal.Length)< -0.5)
obj<- ggparcoord(data, columns=1:3, scale="uniminmax", groupColumn = 6)

ggplotly(obj)

#coloring by another variable
obj<- ggparcoord(data, columns=1:3, scale="uniminmax", groupColumn = 5)

ggplotly(obj)

###heatMap
library(plotly)
mtscaled=scale(mtcars)

plot_ly(x=colnames(mtscaled), y=rownames(mtscaled),
        z=mtscaled, type="heatmap", colors =colorRamp(c("yellow", "red")))


###facetsTrellis
library(plotly)

df=lattice::barley

p<-ggplot(df, aes(y=variety, x=yield, color=year))+geom_point()+
  facet_grid(site~.)
p

ggplotly(p)

df1<-mpg

p1<-ggplot(df1, aes(cty, displ))+geom_point()+geom_smooth()+
  facet_wrap(~class, labeller = "label_both")


df3<-MASS::Aids2
Agerange<-lattice::equal.count(df3$age, number=6, overlap=0.03) #overlap is 3%

L<-matrix(unlist(levels(Agerange)), ncol=2, byrow = T)
L1<-data.frame(Lower=L[,1],Upper=L[,2], Interval=factor(1:nrow(L)))
ggplot(L1)+geom_linerange(aes(ymin = Lower, ymax = Upper, x=Interval))

index=c()
Class=c()
```

```
for(i in 1:nrow(L)){
  Cl=paste("[", L1$Lower[i], ",", L1$Upper[i], "]", sep="")
  ind=which(df3$age>=L1$Lower[i] &df3$age<=L1$Upper[i])
  index=c(index,ind)
  Class=c(Class, rep(Cl, length(ind)))
}

df4<-df3[index,]
df4$Class<-as.factor(Class)

ggplot(df4, aes(x=death-diag, fill="orange"))+ geom_histogram()+
  facet_wrap(~Class, labeller = "label_both")

###interaction
library(plotly)
library(crosstalk)
library(tidyr)

crabs<- read.csv("australian-crabs.csv")

d <- SharedData$new(crabs)

#LINKING EXAMPLES

#Bar-scatter
scatterCrab <- plot_ly(d, x = ~CL, y = ~RW) %>%
  add_markers(color = I("black"))


barCrab <-plot_ly(d, x=~sex)%>%add_histogram()%>%layout(barmode="overlay")

subplot(scatterCrab,barCrab)%>%
  highlight(on="plotly_select", dynamic=T, persistent = T, opacityDim =
I(1))%>%hide_legend()


#Scatter-scatter

scatterCrab2 <- plot_ly(d, x = ~CL, y = ~BD) %>%
  add_markers(color = I("black"))
subplot(scatterCrab,scatterCrab2)%>%
  highlight(on="plotly_select", dynamic=T, persistent=T, opacityDim =
I(1))%>%hide_legend()

#Parallel coords

library(GGally)
p<-ggparcoord(crabs, columns = c(4:6))

d<-plotly_data(ggplotly(p))%>%group_by(.ID)
d1<-SharedData$new(d, ~.ID, group="crab")
p1<-plot_ly(d1, x=~variable, y=~value)%>%
  add_lines(line=list(width=0.3))%>%
  add_markers(marker=list(size=0.3),
              text=~.ID, hoverinfo="text")


crabs2=crabs
crabs2$.ID=1:nrow(crabs)
d2<-SharedData$new(crabs2, ~.ID, group="crab")
p2<-plot_ly(d2, x=~factor(sex) )%>%add_histogram()%>%layout(barmode="stack")

ps<-htmltools::tagList(p1%>%
```

```
  highlight(on="plotly_select", dynamic=T, persistent = T, opacityDim = I(1))%>%
  hide_legend(),
  p2%>%
    highlight(on="plotly_select", dynamic=T, persistent = T, opacityDim = I(1))%>%
    hide_legend()
  )
htmltools::browsable(ps)

#3D-plot and parcoord
df=read.csv("flea.csv")
d2<-SharedData$new(df)

p<-ggparcoord(flea, columns = c(6,7,2))

d<-plotly_data(ggplotly(p))%>%group_by(.ID)
d1<-SharedData$new(d, ~.ID, group="flea")
p1<-plot_ly(d1, x=~variable, y=~value)%>%
  add_lines(line=list(width=0.3))%>%
  add_markers(marker=list(size=0.3),
              text=~.ID, hoverinfo="text")

flea2=flea[, c("tars1", "aede2", "aede3")]
flea2$.ID=1:nrow(flea)
d2<-SharedData$new(flea2, ~.ID, group="flea")

p3<-plot_ly(d2,x=~tars1,y=~aede2,z=~aede3)%>%add_markers()
bscols(p1%>%highlight(on="plotly_select", dynamic=T, persistent = T, opacityDim =
I(1))%>%
         hide_legend(),
       p3%>%highlight(on="plotly_click", dynamic=T, persistent = T)%>%hide_legend())

#Filter - check crosstalk package

d <- SharedData$new(crabs)
scatterCrab <- plot_ly(d, x = ~CL, y = ~RW) %>%
  add_markers(color = I("black"))


barCrab <-plot_ly(d, x=~sex)%>%add_histogram()%>%layout(barmode="overlay")

bscols(widths=c(2, NA),filter_slider("FL", "Frontal Lobe", d, ~FL)
        ,subplot(scatterCrab,barCrab)%>%
  highlight(on="plotly_select", dynamic=T, persistent = T, opacityDim =
I(1))%>%hide_legend())


## Variable selection

ButtonsX=list()
for (i in 4:7){
  ButtonsX[[i-3]]= list(method = "restyle",
                        args = list( "x", list(crabs[[i]])),
                        label = colnames(crabs)[i])
}

ButtonsY=list()
for (i in 4:7){
  ButtonsY[[i-3]]= list(method = "restyle",
                        args = list( "y", list(crabs[[i]])),
                        label = colnames(crabs)[i])
}

p <- plot_ly(d, x = ~CL, y = ~CW, alpha = 0.8) %>%
```

```
  add_markers() %>%
  layout(xaxis=list(title=""), yaxis=list(title=""),
    title = "Select variable:",
    updatemenus = list(
      list(y=0.9, buttons = ButtonsX),
      list(y=0.6, buttons = ButtonsY)
    )  )

p

###interactionWithShiny
#Borrowed from PLOTLYs website

library(plotly)
library(shiny)

# compute a correlation matrix
correlation <- round(cor(mtcars), 3)
nms <- names(mtcars)

ui <- fluidPage(
  mainPanel(
    plotlyOutput("heat"),
    plotlyOutput("scatterplot")
  ),
  verbatimTextOutput("selection")
)

server <- function(input, output, session) {
  output$heat <- renderPlotly({
    plot_ly(x = nms, y = nms, z = correlation,
            key = correlation, type = "heatmap", source = "heatplot") %>%
      layout(xaxis = list(title = ""),
             yaxis = list(title = ""))
  })

  output$selection <- renderPrint({
    s <- event_data("plotly_click", source = "heatplot")
    if (length(s) == 0) {
      "Click on a cell in the heatmap to display a scatterplot"
    } else {
      cat("You selected: \n\n")
      as.list(s)
    }
  })

  output$scatterplot <- renderPlotly({
    s <- event_data("plotly_click", source = "heatplot")
    if (length(s)) {
      vars <- c(s[["x"]], s[["y"]])
      d <- setNames(mtcars[vars], c("x", "y"))
      yhat <- fitted(lm(y ~ x, data = d))
      plot_ly(d, x = ~x) %>%
        add_markers(y = ~y) %>%
        add_lines(y = ~yhat) %>%
        layout(xaxis = list(title = s[["x"]]),
               yaxis = list(title = s[["y"]]),
               showlegend = FALSE)
    } else {
      plotly_empty()
    }
  })
```

```
}

shinyApp(ui, server)

###steamGraph
#devtools::install_github("hrbrmstr/streamgraph")

library(dplyr)
library(ggplot2movies)
library(streamgraph)

ggplot2movies::movies %>%
  select(year, Action, Animation, Comedy, Drama, Documentary, Romance, Short) %>%
  tidyr::gather(genre, value, -year) %>%
  group_by(year, genre) %>%
  tally(wt=value) %>%
  ungroup %>%
  streamgraph("genre", "n", "year") %>%
  sg_axis_x(20) %>%
  sg_fill_brewer("PuOr") %>%
  sg_legend(show=TRUE, label="Genres: ")

###wordCloud
data<-read.table("romeo.txt",header=F, sep='\n') #Read file
library(tm)
library(wordcloud)
library(RColorBrewer)
data$doc_id=1:nrow(data)
colnames(data)[1]<-"text"

#Here we interpret each line in the document as separate document
mycorpus <- Corpus(DataframeSource(data)) #Creating corpus (collection of text data)
mycorpus <- tm_map(mycorpus, removePunctuation)
mycorpus <- tm_map(mycorpus, function(x) removeWords(x, stopwords("english")))
tdm <- TermDocumentMatrix(mycorpus) #Creating term-document matrix
m <- as.matrix(tdm)

#here we merge all rows
v <- sort(rowSums(m),decreasing=TRUE) #Sum up the frequencies of each word
d <- data.frame(word = names(v),freq=v) #Create one column=names, second=frequences
pal <- brewer.pal(6,"Dark2")
pal <- pal[-(1:2)] #Create palette of colors
wordcloud(d$word,d$freq, scale=c(8,.3),min.freq=2,max.words=100, random.order=F,
rot.per=.15, colors=pal, vfont=c("sans serif","plain"))

###network
library(ggraph)
library(igraph)
library(visNetwork)

nodes <- read.csv("Dataset1-Media-Example-NODES.csv", header=T, as.is=T)
links <- read.csv("Dataset1-Media-Example-EDGES.csv", header=T, as.is=T)

## Collapsing multiple links into one

links <- aggregate(links[,3], links[,-3], sum)
links <- links[order(links$from, links$to),]
colnames(links)[4] <- "weight"
rownames(links) <- NULL

nodes$label=nodes$media
#net <- graph_from_data_frame(d=links, vertices=nodes, directed=T)
#visIgraph(net)
```

```
visNetwork(nodes, links)

nodes$group=nodes$type.label
nodes$value=nodes$audience.size
links$width=links$weight
visNetwork(nodes, links)%>%visLegend()

visNetwork(nodes,links)%>%visIgraphLayout(layout="layout_in_circle")

# Community identification
nodes1<-nodes
net <- graph_from_data_frame(d=links, vertices=nodes, directed=F)
ceb <- cluster_edge_betweenness(net)
nodes1$group=ceb$membership
visNetwork(nodes1,links)%>%visIgraphLayout()


#adjacency representation

netm <- get.adjacency(net, attr="weight", sparse=F)
colnames(netm) <- V(net)$media
rownames(netm) <- V(net)$media

rowdist<-dist(netm)

library(seriation)
order1<-seriate(rowdist, "HC")
ord1<-get_order(order1)

reordmatr<-netm[ord1,ord1]

library(plotly)

plot_ly(z=~reordmatr, x=~colnames(reordmatr),
        y=~rownames(reordmatr), type="heatmap")

###motion
library(plotly)

m=matrix(nrow=0,ncol=3)
for (a in seq(0,3,by=0.03)) {
  x<-seq(0,2,0.01)
  y<-x^a
  m<-rbind(m,cbind(x,y,a))
}

df=as.data.frame(m)

plot_ly(df, x=~x, y=~y, frame =~a)%>%add_lines()%>%animation_opts(
  100, easing = "cubic", redraw = F
)


###tours
#A modified code from plotly's website

library(tourr)
library(plotly)

mat <- rescale(mtcars[,c(3:5)])
set.seed(12345)
tour <- new_tour(mat, grand_tour(), NULL)
#tour<- new_tour(mat, guided_tour(cmass), NULL)
```

```
steps <- c(0, rep(1/15, 200))
Projs<-lapply(steps, function(step_size){
  step <- tour(step_size)
  if(is.null(step)) {
    .GlobalEnv$tour<- new_tour(mat, guided_tour(cmass), NULL)
    step <- tour(step_size)
  }
  step
}
)

# projection of each observation
tour_dat <- function(i) {
  step <- Projs[[i]]
  proj <- center(mat %*% step$proj)
  data.frame(x = proj[,1], y = proj[,2], state = rownames(mat))
}

# projection of each variable's axis
proj_dat <- function(i) {
  step <- Projs[[i]]
  data.frame(
    x = step$proj[,1], y = step$proj[,2], variable = colnames(mat)
  )
}

stepz <- cumsum(steps)

# tidy version of tour data

tour_dats <- lapply(1:length(steps), tour_dat)
tour_datz <- Map(function(x, y) cbind(x, step = y), tour_dats, stepz)
tour_dat <- dplyr::bind_rows(tour_datz)

# tidy version of tour projection data
proj_dats <- lapply(1:length(steps), proj_dat)
proj_datz <- Map(function(x, y) cbind(x, step = y), proj_dats, stepz)
proj_dat <- dplyr::bind_rows(proj_datz)

ax <- list(
  title = "", showticklabels = FALSE,
  zeroline = FALSE, showgrid = FALSE,
  range = c(-1.1, 1.1)
)

# for nicely formatted slider labels
options(digits = 3)
tour_dat <- highlight_key(tour_dat, ~state, group = "A")
tour <- proj_dat %>%
  plot_ly(x = ~x, y = ~y, frame = ~step, color = I("black")) %>%
  add_segments(xend = 0, yend = 0, color = I("gray80")) %>%
  add_text(text = ~variable) %>%
  add_markers(data = tour_dat, text = ~state, ids = ~state, hoverinfo = "text") %>%
  layout(xaxis = ax, yaxis = ax)#%>%animation_opts(frame=0, transition=0, redraw = F)
tour


#linking
mtcars1<-mtcars
mtcars1$state<-rownames(mtcars)
mtcars2<-highlight_key(mtcars1, ~state, group="A")
barChart<-plot_ly(mtcars2,x=~factor(cyl))%>%add_histogram()
```

```
subplot(tour, barChart%>%layout(barmode="overlay"))%>%
  highlight(persistent = TRUE, dynamic = TRUE)%>%hide_legend()

###treeMap
library(ggraph)
library(igraph)

graph <- graph_from_data_frame(flare$edges, vertices = flare$vertices)

ggraph(graph, 'treemap', weight = 'size') +
  geom_node_tile(aes(fill = depth), size = 0.25)+
  geom_node_text(label=flare$vertices$shortName, size=3)

devtools::install_github("d3TreeR/d3TreeR")

library(treemap)
library(d3treeR)
data(GNI2014)
d3tree2(treemap(GNI2014,
        index=c("continent", "iso3"),
        vSize="population",
        vColor="GNI",
        type="value",
        format.legend = list(scientific = FALSE, big.mark = " ")),
rootname = "World")

###treeVis
library(ggraph)
library(igraph)
graph <- graph_from_data_frame(flare$edges, vertices = flare$vertices)

ggraph(graph, 'circlepack', weight = 'size') +
  geom_edge_link() +
  geom_node_point(aes(colour = depth)) +
  geom_node_text(label=flare$vertices$shortName, size=1)+
  coord_fixed()

library(visNetwork)
library(rpart)

# Basic classification tree
crabs=read.csv("australian-crabs.csv")
res <- rpart(as.factor(sex)~., data=crabs)
visTree(res, main = "Iris classification Tree", width = "100%")

###sunBurst
library(ggraph)
library(igraph)

graph <- graph_from_data_frame(flare$edges, vertices = flare$vertices)

ggraph(graph, 'partition', circular = TRUE) +
  geom_node_arc_bar(aes(fill = depth), size = 0.25)+
  geom_node_text(label=flare$vertices$shortName, size=3)

###cyclePacking
library(ggraph)

graph <- graph_from_data_frame(flare$edges, vertices = flare$vertices)
```

```
ggraph(graph, 'circlepack', weight = 'size') +
  geom_node_circle(aes(fill = depth), size = 0.25, n = 50) +
  coord_fixed()
```

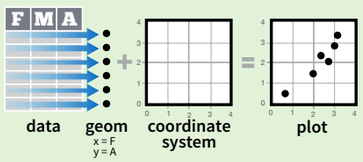
Data Visualization with ggplot2

Cheat Sheet

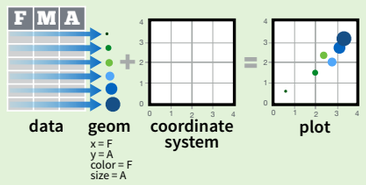
RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

Required

Not required, sensible defaults supplied

ggplot(data = mpg, **aes(**x = cty, y = hwy**))**

Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, data = mpg, geom = "point"**)**

Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot()

Returns the last plot

ggsave("plot.png", width = 5, height = 5)

Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms - Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

Graphical Primitives

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)** - x, ymax, ymin alpha, color, fill, group, linetype, size

Line Segments

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**
b + geom_segment(aes(yend=lat+1, xend=long+1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

One Variable

Continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

Two Variables

Continuous X, Continuous Y

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point()
x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, group, linetype, size, weight

Discrete X, Discrete Y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count()
x, y, alpha, color, fill, shape, size, stroke

Continuous Bivariate Distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

Continuous Function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv"**)**
x, y, alpha, color, group, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar()
x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map**) + expand_limits(**x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, group, linetype, size, weight

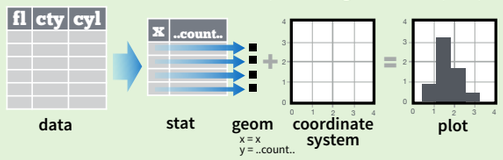
l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

l + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size, width

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

Stats - An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function).

Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geom mappings

i + **stat_density2d**(aes(fill = ..level..), geom = "polygon") — variable created by stat

1D distributions

c + **stat_bin**(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
c + **stat_count**(width = 1) x, y, | ..count.., ..prop..
c + **stat_density**(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

2D distributions

e + **stat_bin_2d**(bins = 30, drop = T)
x, y, fill | ..count.., ..density..
e + **stat_bin_hex**(bins=30) x, y, fill | ..count.., ..density..
e + **stat_density_2d**(contour = TRUE, n = 100)
x, y, color, size | ..level..
e + **stat_ellipse**(level = 0.95, segments = 51, type = "t")

l + **stat_contour**(aes(z = z)) x, y, z, order | ..level..
l + **stat_summary_hex**(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..
l + **stat_summary_2d**(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

3 Variables

Comparisons

f + **stat_boxplot**(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..
f + **stat_ydensity**(kernel = "gaussian", scale = "area")
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

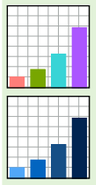
e + **stat_ecdf**(n = 40) x, y | ..x.., ..y..
e + **stat_quantile**(quantiles = c(0.1, 0.9),
formula = y ~ log(x), method = "rq") x, y | ..quantile..
e + **stat_smooth**(method = "lm", formula = y ~ x,
se=T, level=0.95) x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + **stat_function**(aes(x = -3:3), n = 99,
fun = dnorm, args = list(sd=0.5)) x | ..x.., ..y..
e + **stat_identity**(na.rm = TRUE)
ggplot() + **stat_qq**(aes(sample=1:100), dist = qt,
dparam=list(df=5)) sample, x, y | ..sample.., ..theoretical..
e + **stat_sum**() x, y, size | ..n.., ..prop..
e + **stat_summary**(fun.data = "mean_cl_boot")
h + **stat_summary_bin**(fun.y = "mean", geom = "bar")
e + **stat_unique**()

General Purpose

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.



(n <- d + geom_bar(aes(fill = fl)))

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

n + **scale_fill_manual**(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

General Purpose scales

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times.
Use same arguments as scale_x_date(). See ?strptime for label formats.

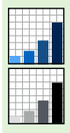
X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales (Discrete)

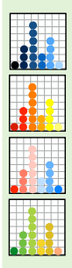
n <- d + geom_bar(aes(fill = fl))



n + **scale_fill_brewer**(palette = "Blues")
For palette choices: RColorBrewer::display.brewer.all()
n + **scale_fill_grey**(start = 0.2, end = 0.8, na.value = "red")

Color and fill scales (Continuous)

o <- c + geom_dotplot(aes(fill = ..x..))



o + **scale_fill_distiller**(palette = "Blues")
o + **scale_fill_gradient**(low="red", high="yellow")
o + **scale_fill_gradient2**(low="red", high="blue", mid = "white", midpoint = 25)
o + **scale_fill_gradientn**(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape and size scales

p <- e + geom_point(aes(shape = fl, size = cyl))

p + **scale_shape()** + **scale_size()**
p + **scale_shape_manual**(values = c(3:7))

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + **scale_radius**(range = c(1,6)) — Maps to radius of circle, or area
p + **scale_size_area**(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + **coord_cartesian**(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

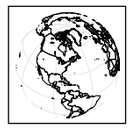
r + **coord_fixed**(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + **coord_flip**()
xlim, ylim
Flipped Cartesian coordinates

r + **coord_polar**(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + **coord_trans**(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

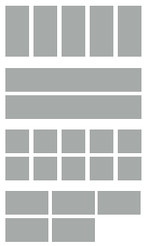
π + **coord_quickmap**()
π + **coord_map**(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)



Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()



t + **facet_grid(. ~ fl)**
facet into columns based on fl
t + **facet_grid(year ~ .)**
facet into rows based on year
t + **facet_grid(year ~ fl)**
facet into both rows and columns
t + **facet_wrap(~ fl)**
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + **facet_grid(drv ~ fl, scales = "free")**
x and y axis limits adjust to individual facets
- **"free_x"** - x axis limits adjust
- **"free_y"** - y axis limits adjust

Set **labeller** to adjust facet labels

t + **facet_grid(. ~ fl, labeller = label_both)**

fl: c	fl: d	fl: e	fl: p	fl: r

t + **facet_grid(fl ~ ., labeller = label_bquote(**alpha ^ .(fl)**))**

α^c	α^d	α^e	α^p	α^r

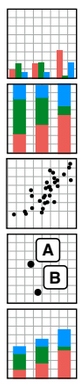
t + **facet_grid(. ~ fl, labeller = label_parsed)**

c	d	e	p	r

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

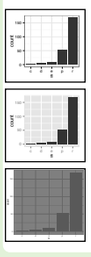


s + **geom_bar(position = "dodge")**
Arrange elements side by side
s + **geom_bar(position = "fill")**
Stack elements on top of one another, normalize height
e + **geom_point(position = "jitter")**
Add random noise to X and Y position of each element to avoid overplotting
e + **geom_label(position = "nudge")**
Nudge labels away from points
s + **geom_bar(position = "stack")**
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments

s + **geom_bar(position = position_dodge(width = 1))**

Themes



r + **theme_bw()**
White background with grid lines
r + **theme_gray()**
Grey background (default theme)
r + **theme_dark()**
dark for contrast
r + **theme_classic()**
r + **theme_light()**
r + **theme_linedraw()**
r + **theme_minimal()**
Minimal themes
r + **theme_void()**
Empty theme

Labels

t + **labs(** x = "New x axis label", y = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

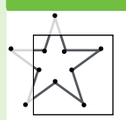
t + **annotate**(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

n + **theme**(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"
n + **guides**(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
n + **scale_fill_discrete**(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming



Without clipping (preferred)
t + **coord_cartesian**(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)

t + **xlim**(0, 100) + **ylim**(10, 20)
t + **scale_x_continuous**(limits = c(0, 100)) + **scale_y_continuous**(limits = c(0, 100))

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

plotly

GETTING STARTED

1. Install

In the console:

```
install.packages ( 'plotly' )
```

2. Sign Up & Configure

plot.ly/r/getting-started

3. A Hello World Figure

```
library( plotly )
p <- plot_ly (
  x = rnorm( 1000 ),
  y = rnorm( 1000 ),
  mode = 'markers' )
```

4. Plot the Figure!

In the console, either:

Plot Offline by printing the figure:

```
p OR  print ( p )
```

Plot and Save in Cloud:

```
plotly_POST ( p )
```

BASIC CHARTS

Line Plots

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7),
  type = 'scatter' ,
  mode = 'lines' )
```

Bubble Charts

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7 ),
  type = 'scatter' ,
  mode = 'markers' ,
  size = c( 1, 5, 10 ),
  marker = list(
      color = c( 'red', 'blue' ,
      'green' )))
```

Scatter Plots

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7 ),
  type = 'scatter' ,
  mode = 'markers' )
```

Heatmaps

```
plot_ly (
   z = volcano ,
   type = 'heatmap' )
```

Bar Charts

```
plot_ly (
  x = c( 1, 2, 3),
  y = c( 5, 6, 7),
  type = 'bar' ,
  mode = 'markers' )
```

Area Plots

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7 ),
  type = 'scatter' ,
  mode = 'lines' ,
  fill = 'tozeroy' )
```

LAYOUT

Legends

```
set.seed( 123 )
  x = 1 : 100
  y1 = 2*x + rnorm ( 100 )
  y2 = -2*x + rnorm ( 100 )

plot_ly (
  x = x ,
  y = y1 ,
  type = 'scatter' ) %>%

  add_trace(
      x = x ,
      y = y2 ) %>%

  layout(
      legend =
          list( x = 0.5 ,
          y = 1 ,
          bgcolor = '#F3F3F3' ))
```

Axes

```
set.seed( 123 )
  x = 1 : 100
  y1 = 2*x + rnorm( 100 )
  y2 = -2*x + rnorm( 100 )

axis_template <- list(
  showgrid = F ,
  zeroline = F ,
  nticks = 20 ,
  showline = T ,
  title = 'AXIS' ,
  mirror = 'all' )

plot_ly (
  x = x ,
  y = y1 ,
  type = 'scatter' ) %>%

  layout(
      xaxis = axis_template ,
      yaxis = axis_template )
```

R CLIENT BASIC CHART PLOT.LY/R

ALL LAYOUTS PLOT.LY/REFERENCE/#LAYOUT

STATISTICAL CHARTS

Histograms

```
x <- rchisq ( 100, 5, 0 )
  plot_ly (
  x = x ,
  type = 'histogram' )
```

Box Plots

```
plot_ly (
  y = rnorm( 50 ) ,
  type = 'box' ) %>%

add_trace( y = rnorm( 50, 1 ))
```

2D Histogram

```
plot_ly (
  x = rnorm( 1000, sd = 10 ) ,
  y = rnorm( 1000, sd = 5 ) ,
  type = 'histogram2d' )
```

MAPS

Bubble Map

```
plot_ly (
  type = 'scattergeo' ,
  lon = c( -73.5, 151.2 ) ,
  lat = c( 45.5, -33.8 ) ,
  marker = list (
    color = c( 'red' , 'blue' ) ,
    size = c( 30, 50 ) ,
    mode = 'markers' ))
```

Choropleth Map

```
plot_ly (
  type = 'choropleth' ,
  locations = c( 'AZ', 'CA', 'VT' ) ,
  locationmode = 'USA-states' ,
  colorscale = 'Viridis' ,
  z = c( 10, 20, 40 )) %>%
  layout ( geo = list ( scope = 'usa' ))
```

Scatter Map

```
plot_ly (
  type = 'scattergeo' ,
  lon = c( 42, 39 ) ,
  lat = c( 12, 22 ) ,
  text = c( 'Rome' , 'Greece' ) ,
  mode = 'markers' )
```

3D CHARTS

3D Surface Plots

```
# Using a dataframe:
plot_ly (
  type = 'surface' ,
  z = ~volcano )
```

3D Line Plots

```
plot_ly (
  type = 'scatter3d' ,
  x = c( 9, 8, 5, 1 ) ,
  y = c( 1, 2, 4, 8 ) ,
  z = c( 11, 8, 15, 3 ) ,
  mode = 'lines' )
```

3D Scatter Plots

```
plot_ly (
  type = 'scatter3d' ,
  x = c( 9, 8, 5, 1 ) ,
  y = c( 1, 2, 4, 8 ) ,
  z = c( 11, 8, 15, 3 ) ,
  mode = 'markers' )
```

FIGURE HIERARCHY

Figure { }

```
plot_ly ( )
  data data.frame
    add _trace list ( )
    x, y, z, c ( )
    color, text, size c ( )
    colorscale 'string' or c ( )
      marker list ( )
        color 'string'
        symbol list ( )
      line list ( )
        color 'string'
        width 123

layout ( )
  title 'string'
  xaxis, yaxis list ( )
  scenelist ( )
    xaxis, yaxis, zaxis list ( )
  geo list ( )
  legend list ( )
  annotations list ( )

  c ( ) = array
  list ( ) = list
  'string' = string
  123 = number
```

Data Import : : CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file
write_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

File with arbitrary delimiter
write_delim(x, path, delim = " ", na = "NA", append = FALSE, col_names = !append**)**

CSV for excel
write_excel_csv(x, path, na = "NA", append = FALSE, col_names = !append**)**

String to file
write_file(x, path, append = FALSE**)**

String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE**)**

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...**)**

Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append**)**

Read Tabular Data - These functions share the common arguments:

read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"), quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000, n_max), progress = interactive()**)**

a,b,c
1,2,3
4,5,NA

A	B	C
1	2	3
4	5	NA

Comma Delimited Files
read_csv("file.csv"**)**
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")

a;b;c
1;2;3
4;5;NA

A	B	C
1	2	3
4	5	NA

Semi-colon Delimited Files
read_csv2("file2.csv"**)**
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")

a|b|c
1|2|3
4|5|NA

A	B	C
1	2	3
4	5	NA

Files with Any Delimiter
read_delim("file.txt", delim = "|"**)**
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5)**)**
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

a b c
1 2 3
4 5 NA

A	B	C
1	2	3
4	5	NA

Tab Delimited Files
read_tsv("file.tsv"**)** Also **read_table().**
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")

USEFUL ARGUMENTS

a,b,c
1,2,3
4,5,NA

Example file
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")
f <- "file.csv"

A	B	C
1	2	3
4	5	NA

No header
read_csv(f, **col_names = FALSE**)

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header
read_csv(f, **col_names = c("x", "y", "z")**)

1	2	3
4	5	NA

Skip lines
read_csv(f, **skip = 1**)

A	B	C
1	2	3

Read in a subset
read_csv(f, **n_max = 1**)

A	B	C
NA	2	3
4	5	NA

Missing Values
read_csv(f, **na = c("1", ".")**)

Read Non-Tabular Data

Read a file into a single string
read_file(file, locale = default_locale()**)**

Read each line into its own string
read_lines(file, skip = 0, n_max = -1L, na = character(), locale = default_locale(), progress = interactive()**)**

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive()**)**

Read a file into a raw vector
read_file_raw(file**)**

Read each line into a raw vector
read_lines_raw(file, skip = 0, n_max = -1L, progress = interactive()**)**

Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:
## cols(
##   age = col_integer(),
##   sex = col_character(),
##   earn = col_double()
## )
```

age is an integer
sex is a character
earn is a double (numeric)

1. Use **problems()** to diagnose problems.
x <- read_csv("file.csv"); problems(x)

2. Use a col_ function to guide parsing.
- **col_guess()** - the default
- **col_character()**
- **col_double()**, **col_euro_double()**
- **col_datetime(**format = ""**)** Also **col_date(**format = ""**)**, **col_time(**format = ""**)**
- **col_factor(**levels, ordered = FALSE**)**
- **col_integer()**
- **col_logical()**
- **col_number()**, **col_numeric()**
- **col_skip()**

```
x <- read_csv("file.csv", col_types = cols(
  A = col_double(),
  B = col_logical(),
  C = col_factor()))
```

3. Else, read in as character vectors then parse with a parse_ function.
- **parse_guess()**
- **parse_character()**
- **parse_datetime()** Also **parse_date()** and **parse_time()**
- **parse_double()**
- **parse_factor()**
- **parse_integer()**
- **parse_logical()**
- **parse_number()**

```
x$A <- parse_number(x$A)
```

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the tibble. Tibbles inherit the data frame class, but improve three behaviors:

- **Subsetting** - [always returns a new tibble, [[and $ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen

```
# A tibble: 234 × 6
   manufacturer      model displ
          <chr>      <chr> <dbl>
1          audi         a4   1.8
2          audi         a4   1.8
3          audi         a4   2.0
4          audi         a4   2.0
5          audi         a4   2.8
6          audi         a4   2.8
7          audi         a4   3.1
8          audi a4 quattro   1.8
9          audi a4 quattro   1.8
10         audi a4 quattro   2.0
# ... with 224 more rows, and 3
#   more variables: year <int>,
#   cyl <int>, trans <chr>
```

tibble display

A large table to display

```
156 1999   6   auto(l4)
157 1999   6   auto(l4)
158 2008   6   auto(l4)
159 2008   8   auto(s4)
160 1999   4 manual(m5)
161 1999   4   auto(l4)
162 2008   4 manual(m5)
163 2008   4 manual(m5)
164 2008   4   auto(l4)
165 2008   4   auto(l4)
166 1999   4   auto(l4)
 [ reached getOption("max.print")
-- omitted 68 rows ]
```

data frame display

- Control the default appearance with options:

  options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

CONSTRUCT A TIBBLE IN TWO WAYS

tibble(…)
Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…)
Construct by rows.
tribble(~x, ~y, 1, "a", 2, "b", 3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …) Convert data frame to tibble.

enframe(x, name = "name", value = "value")
Convert named vector to a tibble

is_tibble(x) Test whether x is a tibble.

Tidy Data with tidyr

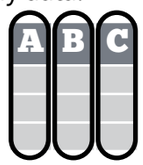
Tidy data is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

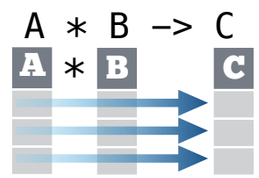
Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

Tidy data:



Makes variables easy to access as vectors



Preserves cases during vectorized operations

Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

gather(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

gather(table4a, `1999`, `2000`, key = "year", value = "cases")

spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

key value

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

spread(table2, type, count)

Handle Missing Values

drop_na(data, ...**)**
Drop rows containing NA's in … columns.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
D	3

drop_na(x, x2)

fill(data, ..., .direction = c("down", "up")**)**
Fill in NA's in … columns with most recent non-NA values.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	1
C	1
D	3
E	3

fill(x, x2)

replace_na(data, replace = list(), ...**)**
Replace NA's by column.

x

x1	x2
A	1
B	NA
C	NA
D	3
E	NA

→

x1	x2
A	1
B	2
C	2
D	3
E	2

replace_na(x, list(x2 = 2))

Expand Tables - quickly create tables with combinations of values

complete(data, ..., fill = list()**)**
Adds to the data missing combinations of the values of the variables listed in …
complete(mtcars, cyl, gear, carb)

expand(data, ...**)**
Create new tibble with all possible combinations of the values of the variables listed in …
expand(mtcars, cyl, gear, carb)

Split Cells

Use these functions to split or combine cells into individual, isolated values.

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...**)**

Separate each cell in a column to make several columns.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174
C	1999	212K	1T
C	2000	213K	1T

separate(table3, rate, into = c("cases", "pop"))

separate_rows(data, ..., sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows. Also **separate_rows_()**.

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M
C	1999	212K/1T
C	2000	213K/1T

→

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M
C	1999	212K
C	1999	1T
C	2000	213K
C	2000	1T

separate_rows(table3, rate)

unite(data, col, ..., sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

table5

country	century	year
Afghan	19	99
Afghan	20	0
Brazil	19	99
Brazil	20	0
China	19	99
China	20	0

→

country	year
Afghan	1999
Afghan	2000
Brazil	1999
Brazil	2000
China	1999
China	2000

unite(table5, century, year, col = "year", sep = "")

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01

Data Transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column**

&

Each **observation**, or **case**, is in its own **row**

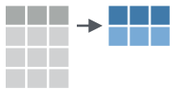
pipes

x %>% f(y) becomes **f(x, y)**

Summarise Cases

These apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function



summarise(.data, …)
Compute table of summaries.
summarise(mtcars, avg = mean(mpg))

count(x, ..., wt = NULL, sort = FALSE)
Count number of rows in each group defined by the variables in … Also **tally**().
count(iris, Species)

VARIATIONS

summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

```
mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))
```

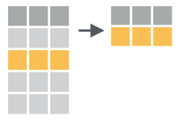
group_by(.data, ..., add = FALSE)
Returns copy of table grouped by …
g_iris <- group_by(iris, Species)

ungroup(x, …)
Returns ungrouped copy of table.
ungroup(g_iris)

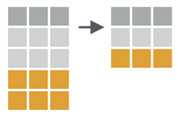
Manipulate Cases

EXTRACT CASES

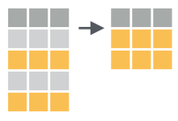
Row functions return a subset of rows as a new table.



filter(.data, …**)** Extract rows that meet logical criteria. *filter(iris, Sepal.Length > 7)*

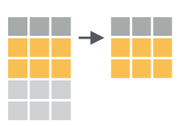


distinct(.data, ..., .keep_all = FALSE**)** Remove rows with duplicate values.
distinct(iris, Species)



sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select fraction of rows.
sample_frac(iris, 0.5, replace = TRUE)

sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()**)** Randomly select size rows. *sample_n(iris, 10, replace = TRUE)*



slice(.data, …**)** Select rows by position.
slice(iris, 10:15)

top_n(x, n, wt**)** Select and order top n entries (by group if grouped data). *top_n(iris, 5, Sepal.Width)*

Logical and boolean operators to use with filter()

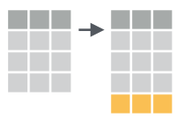
<	<=	is.na()	%in%	|	xor()
>	>=	!is.na()	!	&	

See **?base::logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …**)** Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
arrange(mtcars, mpg)
arrange(mtcars, desc(mpg))

ADD CASES



add_row(.data, ..., .before = NULL, .after = NULL**)**
Add one or more rows to a table.
add_row(faithful, eruptions = 1, waiting = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1**)** Extract column values as a vector. Choose by name or index.
pull(iris, Sepal.Length)

select(.data, …**)**
Extract columns as a table. Also **select_if()**.
select(iris, Sepal.Length, Species)

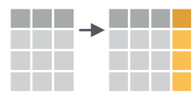
Use these helpers with select (),
e.g. select(iris, starts_with("Sepal"))

contains(match**)**	**num_range(**prefix, range**)**	**:**, e.g. mpg:cyl
ends_with(match**)**	**one_of(**…**)**	**-**, e.g, -Species
matches(match**)**	**starts_with(**match**)**	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function

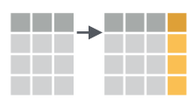


mutate(.data, …**)**
Compute new column(s).
mutate(mtcars, gpm = 1/mpg)

transmute(.data, …**)**
Compute new column(s), drop others.
transmute(mtcars, gpm = 1/mpg)

mutate_all(.tbl, .funs, …**)** Apply funs to every column. Use with **funs()**. Also **mutate_if()**.
mutate_all(faithful, funs(log(.), log2(.)))
mutate_if(iris, is.numeric, funs(log(.)))

mutate_at(.tbl, .cols, .funs, …**)** Apply funs to specific columns. Use with **funs()**, **vars()** and the helper functions for select().
mutate_at(iris, vars(-Species), funs(log(.)))



add_column(.data, ..., .before = NULL, .after = NULL**)** Add new column(s). Also **add_count()**, **add_tally()**. *add_column(mtcars, new = 1:32)*

rename(.data, …**)** Rename columns.
rename(iris, Length = Sepal.Length)

Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function →

OFFSETS

dplyr::**lag()** - Offset elements by 1
dplyr::**lead()** - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::**cumall()** - Cumulative all()
dplyr::**cumany()** - Cumulative any()
cummax() - Cumulative max()
dplyr::**cummean()** - Cumulative mean()
cummin() - Cumulative min()
cumprod() - Cumulative prod()
cumsum() - Cumulative sum()

RANKINGS

dplyr::**cume_dist()** - Proportion of all values <=
dplyr::**dense_rank()** - rank with ties = min, no gaps
dplyr::**min_rank()** - rank with ties = min
dplyr::**ntile()** - bins into n bins
dplyr::**percent_rank()** - min_rank scaled to [0,1]
dplyr::**row_number()** - rank with ties = "first"

MATH

+, - , *, /, ^, %/%, %% - arithmetic ops
log(), log2(), log10() - logs
<, <=, >, >=, !=, == - logical comparisons
dplyr::**between()** - x >= left & x <= right
dplyr::**near()** - safe == for floating point numbers

MISC

dplyr::**case_when()** - multi-case if_else()
dplyr::**coalesce()** - first non-NA values by element across a set of vectors
dplyr::**if_else()** - element-wise if() + else()
dplyr::**na_if()** - replace specific values with NA
pmax() - element-wise max()
pmin() - element-wise min()
dplyr::**recode()** - Vectorized switch()
dplyr::**recode_factor()** - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function →

COUNTS

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::**first()** - first value
dplyr::**last()** - last value
dplyr::**nth()** - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

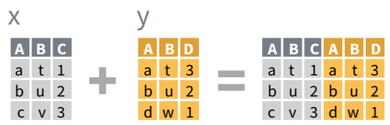
rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES



Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(…) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

left_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join matching values from y to x.

right_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join matching values from x to y.

inner_join(x, y, by = NULL, copy = FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain only rows with matches.

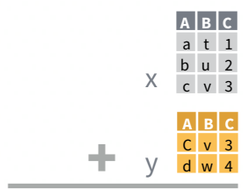
full_join(x, y, by = NULL, copy=FALSE, suffix=c(".x",".y"),…**)**
Join data. Retain all values, all rows.

Use **by = c("col1", "col2", …)** to specify one or more common columns to match on.
left_join(x, y, by = "A")

Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.
left_join(x, y, by = c("C" = "D"))

Use **suffix** to specify the suffix to give to unmatched columns that have the same name in both tables.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES



Use **bind_rows()** to paste tables below each other as they are.

bind_rows(…, .id = NULL**)**
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

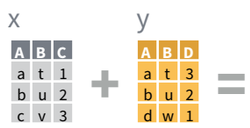
intersect(x, y, …)
Rows that appear in both x and y.

setdiff(x, y, …)
Rows that appear in x but not y.

union(x, y, …)
Rows that appear in x or y. (Duplicates removed). union_all() retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS



Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, …**)**
Return rows of x that have a match in y. USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, …**)**
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more with browseVignettes(package = c("dplyr", "tibble")) • dplyr 0.7.0 • tibble 1.2.0 • Updated: 2017-03

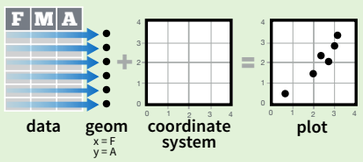
Data Visualization with ggplot2

Cheat Sheet

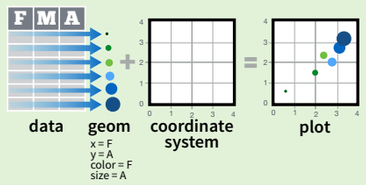
RStudio

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and **geoms**—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(
    mapping = aes(<MAPPINGS>),
    stat = <STAT>,
    position = <POSITION>
  ) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

Required

Not required, sensible defaults supplied

ggplot(data = mpg, **aes(**x = cty, y = hwy**))**

Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, data = mpg, geom = "point"**)**

Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot()

Returns the last plot

ggsave("plot.png", width = 5, height = 5)

Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms - Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

Graphical Primitives

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round', linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group)**)**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)**)** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)**)** - x, ymax, ymin alpha, color, fill, group, linetype, size

Line Segments

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1)**)**
b + geom_hline(aes(yintercept = lat)**)**
b + geom_vline(aes(xintercept = long)**)**
b + geom_segment(aes(yend=lat+1, xend=long+1)**)**
b + geom_spoke(aes(angle = 1:1155, radius = 1)**)**

One Variable

Continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly()
x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

Two Variables

Continuous X, Continuous Y

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point()
x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile()
x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**
x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**
x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

f <- ggplot(mpg, aes(class, hwy))

f + geom_col()
x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot()
x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, group, linetype, size, weight

Discrete X, Discrete Y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count()
x, y, alpha, color, fill, shape, size, stroke

Continuous Bivariate Distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

Continuous Function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv"**)**
x, y, alpha, color, group, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar()
x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map**) + expand_limits(**x = map$long, y = map$lat**)**
map_id, alpha, color, fill, linetype, size

Three Variables

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, group, linetype, size, weight

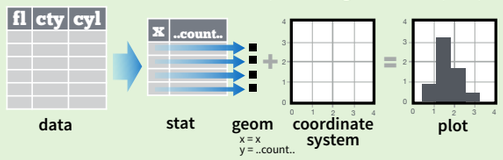
l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

l + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size, width

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

Stats - An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function).

Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geom mappings

i + **stat_density2d**(aes(fill = ..level..), geom = "polygon") — variable created by stat

1D distributions

c + **stat_bin**(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
c + **stat_count**(width = 1) x, y, | ..count.., ..prop..
c + **stat_density**(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

2D distributions

e + **stat_bin_2d**(bins = 30, drop = T)
x, y, fill | ..count.., ..density..
e + **stat_bin_hex**(bins=30) x, y, fill | ..count.., ..density..
e + **stat_density_2d**(contour = TRUE, n = 100)
x, y, color, size | ..level..
e + **stat_ellipse**(level = 0.95, segments = 51, type = "t")

l + **stat_contour**(aes(z = z)) x, y, z, order | ..level..
l + **stat_summary_hex**(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..
l + **stat_summary_2d**(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

3 Variables

Comparisons

f + **stat_boxplot**(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..
f + **stat_ydensity**(kernel = "gaussian", scale = "area")
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

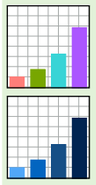
e + **stat_ecdf**(n = 40) x, y | ..x.., ..y..
e + **stat_quantile**(quantiles = c(0.1, 0.9),
formula = y ~ log(x), method = "rq") x, y | ..quantile..
e + **stat_smooth**(method = "lm", formula = y ~ x,
se=T, level=0.95) x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + **stat_function**(aes(x = -3:3), n = 99,
fun = dnorm, args = list(sd=0.5)) x | ..x.., ..y..
e + **stat_identity**(na.rm = TRUE)
ggplot() + **stat_qq**(aes(sample=1:100), dist = qt,
dparam=list(df=5)) sample, x, y | ..sample.., ..theoretical..
e + **stat_sum**() x, y, size | ..n.., ..prop..
e + **stat_summary**(fun.data = "mean_cl_boot")
h + **stat_summary_bin**(fun.y = "mean", geom = "bar")
e + **stat_unique**()

General Purpose

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.



(n <- d + geom_bar(aes(fill = fl)))

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

n + **scale_fill_manual**(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

General Purpose scales

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times.
Use same arguments as scale_x_date(). See ?strptime for label formats.

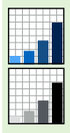
X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales (Discrete)

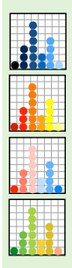
n <- d + geom_bar(aes(fill = fl))



n + **scale_fill_brewer**(palette = "Blues")
For palette choices: RColorBrewer::display.brewer.all()
n + **scale_fill_grey**(start = 0.2, end = 0.8, na.value = "red")

Color and fill scales (Continuous)

o <- c + geom_dotplot(aes(fill = ..x..))



o + **scale_fill_distiller**(palette = "Blues")
o + **scale_fill_gradient**(low="red", high="yellow")
o + **scale_fill_gradient2**(low="red", high="blue", mid = "white", midpoint = 25)
o + **scale_fill_gradientn**(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape and size scales

p <- e + geom_point(aes(shape = fl, size = cyl))

p + **scale_shape()** + **scale_size()**
p + **scale_shape_manual**(values = c(3:7))

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

p + **scale_radius**(range = c(1,6)) — Maps to radius of circle, or area
p + **scale_size_area**(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + **coord_cartesian**(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

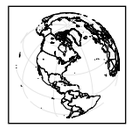
r + **coord_fixed**(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + **coord_flip**()
xlim, ylim
Flipped Cartesian coordinates

r + **coord_polar**(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + **coord_trans**(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

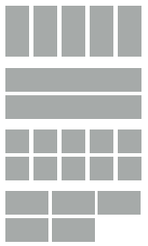
π + **coord_quickmap**()
π + **coord_map**(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)



Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()



t + **facet_grid(. ~ fl)**
facet into columns based on fl
t + **facet_grid(year ~ .)**
facet into rows based on year
t + **facet_grid(year ~ fl)**
facet into both rows and columns
t + **facet_wrap(~ fl)**
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + **facet_grid(drv ~ fl, scales = "free")**
x and y axis limits adjust to individual facets
- **"free_x"** - x axis limits adjust
- **"free_y"** - y axis limits adjust

Set **labeller** to adjust facet labels

t + **facet_grid(. ~ fl, labeller = label_both)**

fl: c	fl: d	fl: e	fl: p	fl: r

t + **facet_grid(fl ~ ., labeller = label_bquote(**alpha ^ .(fl)**))**

α^c	α^d	α^e	α^p	α^r

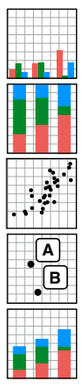
t + **facet_grid(. ~ fl, labeller = label_parsed)**

c	d	e	p	r

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

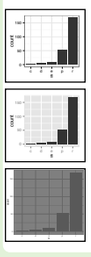


s + **geom_bar(position = "dodge")**
Arrange elements side by side
s + **geom_bar(position = "fill")**
Stack elements on top of one another, normalize height
e + **geom_point(position = "jitter")**
Add random noise to X and Y position of each element to avoid overplotting
e + **geom_label(position = "nudge")**
Nudge labels away from points
s + **geom_bar(position = "stack")**
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments

s + **geom_bar(position = position_dodge(width = 1))**

Themes



r + **theme_bw()**
White background with grid lines
r + **theme_gray()**
Grey background (default theme)
r + **theme_dark()**
dark for contrast
r + **theme_classic()**
r + **theme_light()**
r + **theme_linedraw()**
r + **theme_minimal()**
Minimal themes
r + **theme_void()**
Empty theme

Labels

t + **labs(** x = "New x axis label", y = "New y axis label",
title ="Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
<AES> = "New <AES> legend title")

Use scale functions to update legend labels

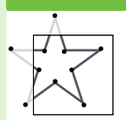
t + **annotate**(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

n + **theme**(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"
n + **guides**(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
n + **scale_fill_discrete**(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming



Without clipping (preferred)
t + **coord_cartesian**(
xlim = c(0, 100), ylim = c(10, 20))

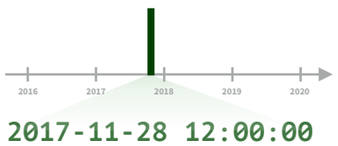
With clipping (removes unseen data points)

t + **xlim**(0, 100) + **ylim**(10, 20)
t + **scale_x_continuous**(limits = c(0, 100)) + **scale_y_continuous**(limits = c(0, 100))

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

Dates and times with lubridate : : CHEAT SHEET

Date-times



`2017-11-28 12:00:00`

`2017-11-28 12:00:00`
A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

dt <- ***as_datetime****(1511870400)*
"2017-11-28 12:00:00 UTC"

`2017-11-28`
A **date** is a day stored as the number of days since 1970-01-01

d <- ***as_date****(17498)*
"2017-11-28"

`12:00:00`
An hms is a **time** stored as the number of seconds since 00:00:00

*t <- hms::****as.hms****(85)*
00:01:25

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

`2017-11-28T14:02:00` **ymd_hms**(), **ymd_hm**(), **ymd_h**(). *ymd_hms("2017-11-28T14:02:00")*

`2017-22-12 10:00:00` **ydm_hms**(), **ydm_hm**(), **ydm_h**(). *ydm_hms("2017-22-12 10:00:00")*

`11/28/2017 1:02:03` **mdy_hms**(), **mdy_hm**(), **mdy_h**(). *mdy_hms("11/28/2017 1:02:03")*

`1 Jan 2017 23:59:59` **dmy_hms**(), **dmy_hm**(), **dmy_h**(). *dmy_hms("1 Jan 2017 23:59:59")*

`20170131` **ymd**(), **ydm**(). *ymd(20170131)*

`July 4th, 2000` **mdy**(), **myd**(). *mdy("July 4th, 2000")*

`4th of July '99` **dmy**(), **dym**(). *dmy("4th of July '99")*

`2001: Q3` **yq**() Q for quarter. *yq("2001: Q3")*

`2:01` hms::**hms**() Also lubridate::**hms**(), **hm**() and **ms**(), which return periods.* *hms::hms(sec = 0, min= 1, hours = 2)*

`2017.5` **date_decimal**(decimal, tz = "UTC") *date_decimal(2017.5)*

now(tzone = "") Current time in tz (defaults to system tz). *now()*

today(tzone = "") Current date in a tz (defaults to system tz). *today()*

fast_strptime() Faster strptime. *fast_strptime('9/1/01', '%y/%m/%d')*

parse_date_time() Easier strptime. *parse_date_time("9/1/01", "ymd")*

GET AND SET COMPONENTS

Use an accessor function to get a component.

d ## "2017-11-28"
day(d) ## 28

Assign into an accessor function to change a component in place.

day(d) <- 1
d ## "2017-11-01"

`2018-01-31 11:59:59` **date**(x) Date component. *date(dt)*

`2018-01-31 11:59:59` **year**(x) Year. *year(dt)*
isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.

`2018-01-31 11:59:59` **month**(x, label, abbr) Month. *month(dt)*

`2018-01-31 11:59:59` **day**(x) Day of month. *day(dt)*
wday(x,label,abbr) Day of week.
qday(x) Day of quarter.

`2018-01-31 11:59:59` **hour**(x) Hour. *hour(dt)*

`2018-01-31 11:59:59` **minute**(x) Minutes. *minute(dt)*

`2018-01-31 11:59:59` **second**(x) Seconds. *second(dt)*

week(x) Week of the year. *week(dt)*
isoweek() ISO 8601 week.
epiweek() Epidemiological week.

quarter(x, with_year = FALSE) Quarter. *quarter(dt)*

semester(x, with_year = FALSE) Semester. *semester(dt)*

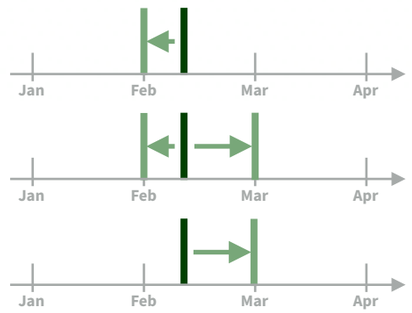
am(x) Is it in the am? *am(dt)*
pm(x) Is it in the pm? *pm(dt)*

dst(x) Is it daylight savings? *dst(d)*

leap_year(x) Is it a leap year? *leap_year(d)*

update(object, ..., simple = FALSE) *update(dt, mday = 2, hour = 1)*

Round Date-times



floor_date(x, unit = "second") Round down to nearest unit. *floor_date(dt, unit = "month")*

round_date(x, unit = "second") Round to nearest unit. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit. *ceiling_date(dt, unit = "month")*

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. *rollback(dt)*

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date**() and **stamp_time**().

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

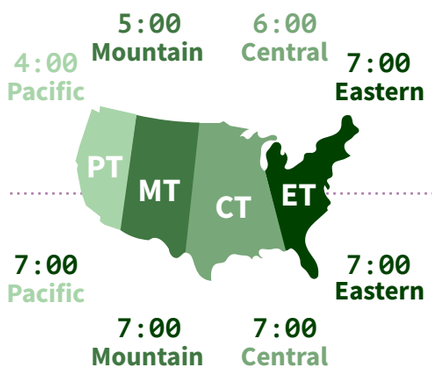
Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. *OlsonNames()*



with_tz(time, tzone = "") Get the **same date-time** in a new time zone (a new clock time). *with_tz(dt, "US/Pacific")*

force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). *force_tz(dt, "US/Pacific")*

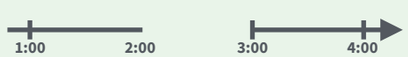
RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **lubridate.tidyverse.org** • lubridate 1.6.0 • Updated: 2017-12

Math with Date-times — Lubridate provides three classes of timespans to facilitate math with dates and date-times

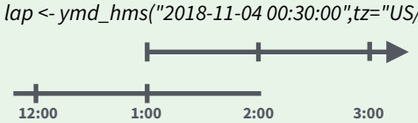
Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

A normal day
nor <- ymd_hms("2018-01-01 01:30:00",tz="US/Eastern")

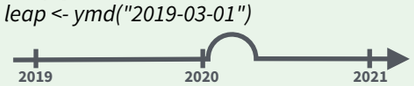
The start of daylight savings (spring forward)
gap <- ymd_hms("2018-03-11 01:30:00",tz="US/Eastern")



The end of daylight savings (fall back)
lap <- ymd_hms("2018-11-04 00:30:00",tz="US/Eastern")

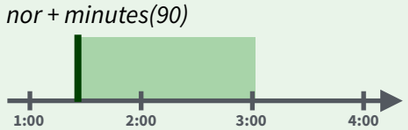


Leap years and leap seconds
leap <- ymd("2019-03-01")

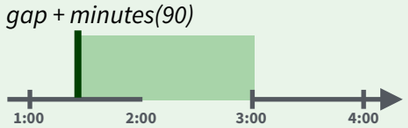


Periods track changes in clock times, which ignore time line irregularities.

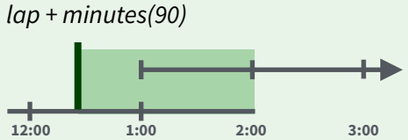
nor + minutes(90)



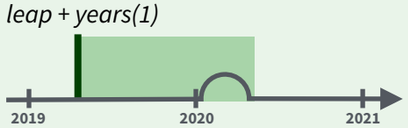
gap + minutes(90)



lap + minutes(90)

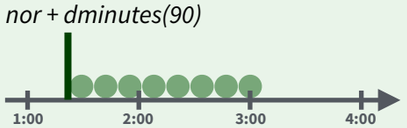


leap + years(1)

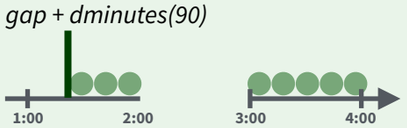


Durations track the passage of physical time, which deviates from clock time when irregularities occur.

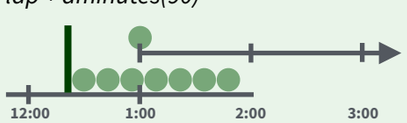
nor + dminutes(90)



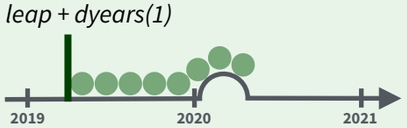
gap + dminutes(90)



lap + dminutes(90)

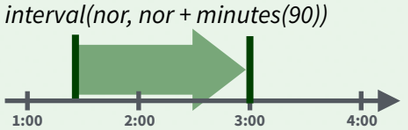


leap + dyears(1)

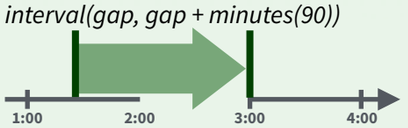


Intervals represent specific intervals of the timeline, bounded by start and end date-times.

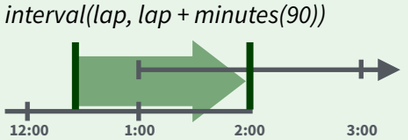
interval(nor, nor + minutes(90))



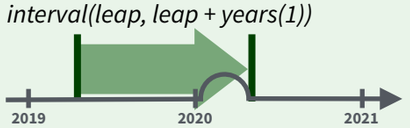
interval(gap, gap + minutes(90))



interval(lap, lap + minutes(90))



interval(leap, leap + years(1))



Not all years are 365 days due to **leap days**.

Not all minutes are 60 seconds due to **leap seconds**.

It is possible to create an imaginary date by adding **months**, e.g. February 31st

```
jan31 <- ymd(20180131)
jan31 + months(1)
## NA
```

%m+% and **%m-%** will roll imaginary dates to the last day of the previous month.

```
jan31 %m+% months(1)
## "2018-02-28"
```

add_with_rollback(e1, e2, roll_to_first = TRUE) will roll imaginary dates to the first day of the new month.

```
add_with_rollback(jan31, months(1),
roll_to_first = TRUE)
## "2018-03-01"
```

PERIODS

Add or subtract periods to model events that happen at specific clock times, like the NYSE opening bell.

Make a period with the name of a time unit ***pluralized***, e.g.

```
p <- months(3) + days(12)
p
"3m 12d 0H 0M 0S"
```

Number of months · Number of days · etc.

years(x = 1) x years.
months(x) x months.
weeks(x = 1) x weeks.
days(x = 1) x days.
hours(x = 1) x hours.
minutes(x = 1) x minutes.
seconds(x = 1) x seconds.
milliseconds(x = 1) x milliseconds.
microseconds(x = 1) x microseconds
nanoseconds(x = 1) x nanoseconds.
picoseconds(x = 1) x picoseconds.

period(num = NULL, units = "second", ...) An automation friendly period constructor. *period(5, unit = "years")*

as.period(x, unit) Coerce a timespan to a period, optionally in the specified units. Also **is.period**(). *as.period(i)*

period_to_seconds(x) Convert a period to the "standard" number of seconds implied by the period. Also **seconds_to_period**(). *period_to_seconds(p)*

DURATIONS

Add or subtract durations to model physical processes, like battery life. Durations are stored as seconds, the only time unit with a consistent length. **Difftimes** are a class of durations found in base R.

Make a duration with the name of a period prefixed with a ***d***, e.g.

```
dd <- ddays(14)
dd
"1209600s (~2 weeks)"
```

Exact length in seconds · Equivalent in common units

dyears(x = 1) 31536000x seconds.
dweeks(x = 1) 604800x seconds.
ddays(x = 1) 86400x seconds.
dhours(x = 1) 3600x seconds.
dminutes(x = 1) 60x seconds.
dseconds(x = 1) x seconds.
dmilliseconds(x = 1) x $\times 10^{-3}$ seconds.
dmicroseconds(x = 1) x $\times 10^{-6}$ seconds.
dnanoseconds(x = 1) x $\times 10^{-9}$ seconds.
dpicoseconds(x = 1) x $\times 10^{-12}$ seconds.

duration(num = NULL, units = "second", ...) An automation friendly duration constructor. *duration(5, unit = "years")*

as.duration(x, …) Coerce a timespan to a duration. Also **is.duration**(), **is.difftime**(). *as.duration(i)*

make_difftime(x) Make difftime with the specified number of units. *make_difftime(99999)*

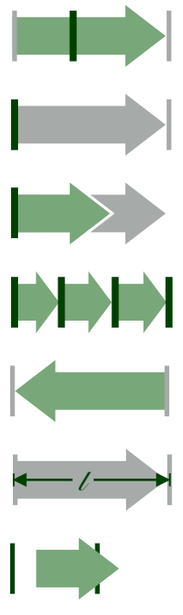
INTERVALS

Divide an interval by a duration to determine its physical length, divide an interval by a period to determine its implied length in clock time.

Make an interval with **interval**() or **%--%**, e.g.

Start Date · End Date

```
i <- interval(ymd("2017-01-01"), d)      ## 2017-01-01 UTC--2017-11-28 UTC
j <- d %--% ymd("2017-12-31")            ## 2017-11-28 UTC--2017-12-31 UTC
```



a **%within%** b Does interval or date-time *a* fall within interval *b*? *now() %within% i*

int_start(int) Access/set the start date-time of an interval. Also **int_end**(). *int_start(i) <- now(); int_start(i)*

int_aligns(int1, int2) Do two intervals share a boundary? Also **int_overlaps**(). *int_aligns(i, j)*

int_diff(times) Make the intervals that occur between the date-times in a vector. *v <-c(dt, dt + 100, dt + 1000); int_diff(v)*

int_flip(int) Reverse the direction of an interval. Also **int_standardize**(). *int_flip(i)*

int_length(int) Length in seconds. *int_length(i)*

int_shift(int, by) Shifts an interval up or down the timeline by a timespan. *int_shift(i, days(-1))*

as.interval(x, start, …) Coerce a timespans to an interval with the start date-time. Also **is.interval**(). *as.interval(days(1), start = now())*

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **lubridate.tidyverse.org** • lubridate 1.6.0 • Updated: 2017-12

plotly

GETTING STARTED

1. Install

In the console:

```
install.packages ( 'plotly' )
```

2. Sign Up & Configure

plot.ly/r/getting-started

3. A Hello World Figure

```
library( plotly )
p <- plot_ly (
  x = rnorm( 1000 ),
  y = rnorm( 1000 ),
  mode = 'markers' )
```

4. Plot the Figure!

In the console, either:

Plot Offline by printing the figure:

```
p OR  print ( p )
```

Plot and Save in Cloud:

```
plotly_POST ( p )
```

BASIC CHARTS

Line Plots

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7),
  type = 'scatter' ,
  mode = 'lines' )
```

Bubble Charts

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7 ),
  type = 'scatter' ,
  mode = 'markers' ,
  size = c( 1, 5, 10 ),
  marker = list(
      color = c( 'red', 'blue' ,
      'green' )))
```

Scatter Plots

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7 ),
  type = 'scatter' ,
  mode = 'markers' )
```

Heatmaps

```
plot_ly (
   z = volcano ,
   type = 'heatmap' )
```

Bar Charts

```
plot_ly (
  x = c( 1, 2, 3),
  y = c( 5, 6, 7),
  type = 'bar' ,
  mode = 'markers' )
```

Area Plots

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7 ),
  type = 'scatter' ,
  mode = 'lines' ,
  fill = 'tozeroy' )
```

LAYOUT

Legends

```
set.seed( 123 )
  x = 1 : 100
  y1 = 2*x + rnorm ( 100 )
  y2 = -2*x + rnorm ( 100 )

plot_ly (
  x = x ,
  y = y1 ,
  type = 'scatter' ) %>%

  add_trace(
      x = x ,
      y = y2 ) %>%

  layout(
      legend =
          list( x = 0.5 ,
          y = 1 ,
          bgcolor = '#F3F3F3' ))
```

Axes

```
set.seed( 123 )
  x = 1 : 100
  y1 = 2*x + rnorm( 100 )
  y2 = -2*x + rnorm( 100 )

axis_template <- list(
  showgrid = F ,
  zeroline = F ,
  nticks = 20 ,
  showline = T ,
  title = 'AXIS' ,
  mirror = 'all' )

plot_ly (
  x = x ,
  y = y1 ,
  type = 'scatter' ) %>%

  layout(
      xaxis = axis_template ,
      yaxis = axis_template )
```

R CLIENT BASIC CHART PLOT.LY/R

ALL LAYOUTS PLOT.LY/REFERENCE/#LAYOUT

STATISTICAL CHARTS

Histograms

```
x <- rchisq ( 100, 5, 0 )
  plot_ly (
  x = x ,
  type = 'histogram' )
```

Box Plots

```
plot_ly (
  y = rnorm( 50 ) ,
  type = 'box' ) %>%

add_trace( y = rnorm( 50, 1 ))
```

2D Histogram

```
plot_ly (
  x = rnorm( 1000, sd = 10 ) ,
  y = rnorm( 1000, sd = 5 ) ,
  type = 'histogram2d' )
```

MAPS

Bubble Map

```
plot_ly (
  type = 'scattergeo' ,
  lon = c( -73.5, 151.2 ) ,
  lat = c( 45.5, -33.8 ) ,
  marker = list (
    color = c( 'red' , 'blue' ) ,
    size = c( 30, 50 ) ,
    mode = 'markers' ))
```

Choropleth Map

```
plot_ly (
  type = 'choropleth' ,
  locations = c( 'AZ', 'CA', 'VT' ) ,
  locationmode = 'USA-states' ,
  colorscale = 'Viridis' ,
  z = c( 10, 20, 40 )) %>%
  layout ( geo = list ( scope = 'usa' ))
```

Scatter Map

```
plot_ly (
  type = 'scattergeo' ,
  lon = c( 42, 39 ) ,
  lat = c( 12, 22 ) ,
  text = c( 'Rome' , 'Greece' ) ,
  mode = 'markers' )
```

3D CHARTS

3D Surface Plots

```
# Using a dataframe:
plot_ly (
  type = 'surface' ,
  z = ~volcano )
```

3D Line Plots

```
plot_ly (
  type = 'scatter3d' ,
  x = c( 9, 8, 5, 1 ) ,
  y = c( 1, 2, 4, 8 ) ,
  z = c( 11, 8, 15, 3 ) ,
  mode = 'lines' )
```

3D Scatter Plots

```
plot_ly (
  type = 'scatter3d' ,
  x = c( 9, 8, 5, 1 ) ,
  y = c( 1, 2, 4, 8 ) ,
  z = c( 11, 8, 15, 3 ) ,
  mode = 'markers' )
```

FIGURE HIERARCHY

Figure { }

```
plot_ly ( )
  data data.frame
    add _trace list ( )
    x, y, z, c ( )
    color, text, size c ( )
    colorscale 'string' or c ( )
      marker list ( )
        color 'string'
        symbol list ( )
      line list ( )
        color 'string'
        width 123

layout ( )
  title 'string'
  xaxis, yaxis list ( )
  scenelist ( )
    xaxis, yaxis, zaxis list ( )
  geo list ( )
  legend list ( )
  annotations list ( )

  c ( ) = array
  list ( ) = list
  'string' = string
  123 = number
```

Shiny examples:

1)

```
library(shiny)

# Define UI for app that draws a histogram ----
ui <- fluidPage(

  # App title ----
  titlePanel("Hello Shiny!"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Slider for the number of bins ----
      sliderInput(inputId = "bins",
                  label = "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Histogram ----
      plotOutput(outputId = "distPlot")

    )
  )
)

# Define server logic required to draw a histogram ----
server <- function(input, output) {

  # Histogram of the Old Faithful Geyser Data ----
  # with requested number of bins
  # This expression that generates a histogram is wrapped in a call
  # to renderPlot to indicate that:
  #
  # 1. It is "reactive" and therefore should be automatically
  #    re-executed when inputs (input$bins) change
```

```
  # 2. Its output type is a plot
  output$distPlot <- renderPlot({

    x    <- faithful$waiting
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    hist(x, breaks = bins, col = "#75AADB", border = "white",
         xlab = "Waiting time to next eruption (in mins)",
         main = "Histogram of waiting times")

    })

}

# Create Shiny app ----
shinyApp(ui = ui, server = server)
```

2)

```
library(shiny)

# Define UI for dataset viewer app ----
ui <- fluidPage(

  # App title ----
  titlePanel("Reactivity"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Text for providing a caption ----
      # Note: Changes made to the caption in the textInput control
      # are updated in the output area immediately as you type
      textInput(inputId = "caption",
                label = "Caption:",
                value = "Data Summary"),

      # Input: Selector for choosing dataset ----
      selectInput(inputId = "dataset",
                  label = "Choose a dataset:",
                  choices = c("rock", "pressure", "cars")),

      # Input: Numeric entry for number of obs to view ----
```

```
      numericInput(inputId = "obs",
                   label = "Number of observations to view:",
                   value = 10)

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Formatted text for caption ----
      h3(textOutput("caption", container = span)),

      # Output: Verbatim text for data summary ----
      verbatimTextOutput("summary"),

      # Output: HTML table with requested number of observations ----
      tableOutput("view")

    )
  )
)

# Define server logic to summarize and view selected dataset ----
server <- function(input, output) {

  # Return the requested dataset ----
  # By declaring datasetInput as a reactive expression we ensure
  # that:
  #
  # 1. It is only called when the inputs it depends on changes
  # 2. The computation and result are shared by all the callers,
  #    i.e. it only executes a single time
  datasetInput <- reactive({
    switch(input$dataset,
           "rock" = rock,
           "pressure" = pressure,
           "cars" = cars)
  })

  # Create caption ----
  # The output$caption is computed based on a reactive expression
  # that returns input$caption. When the user changes the
  # "caption" field:
  #
  # 1. This function is automatically called to recompute the output
  # 2. New caption is pushed back to the browser for re-display
```

```
  #
  # Note that because the data-oriented reactive expressions
  # below don't depend on input$caption, those expressions are
  # NOT called when input$caption changes
  output$caption <- renderText({
    input$caption
  })

  # Generate a summary of the dataset ----
  # The output$summary depends on the datasetInput reactive
  # expression, so will be re-executed whenever datasetInput is
  # invalidated, i.e. whenever the input$dataset changes
  output$summary <- renderPrint({
    dataset <- datasetInput()
    summary(dataset)
  })

  # Show the first "n" observations ----
  # The output$view depends on both the databaseInput reactive
  # expression and input$obs, so it will be re-executed whenever
  # input$dataset or input$obs is changed
  output$view <- renderTable({
    head(datasetInput(), n = input$obs)
  })

}

# Create Shiny app ----
shinyApp(ui, server)
```

3)

```
rm(list = ls())

# load packages
library(treemap)
library(readr) # for read_csv

# read data in
the_url <-
"https://raw.githubusercontent.com/LarryZhang2016/Data/master/NZ_cities.csv"
NZ_cities <- read_csv(the_url, skip =1)

# make a tree map
treemap(dtf = NZ_cities,
        index=c("City_name"),
        vSize="Population",
        vColor="Population_density",
        palette="Spectral",
        type="value",
        border.col=c("grey70", "grey90"),
```

```
        fontsize.title = 18,
        algorithm="pivotSize",
        title ="Treemap of the top 15 NZ's most populous cities",
        title.legend="Population density (people/km^2)")
```

4)

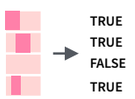
```
library(maps)
library(mapproj)
source("helpers.R")
counties <- readRDS("data/counties.rds")

# User interface ----
ui <- fluidPage(
  titlePanel("censusVis"),

  sidebarLayout(
    sidebarPanel(
      helpText("Create demographic maps with
        information from the 2010 US Census."),

      selectInput("var",
                  label = "Choose a variable to display",
                  choices = c("Percent White", "Percent Black",
                              "Percent Hispanic", "Percent Asian"),
                  selected = "Percent White"),

      sliderInput("range",
                  label = "Range of interest:",
                  min = 0, max = 100, value = c(0, 100))
    ),

    mainPanel(plotOutput("map"))
  )
)

# Server logic ----
server <- function(input, output) {
  output$map <- renderPlot({
    percent_map( # some arguments )
  })
}

# Run app ----

shinyApp(ui, server)
```

```
server <- function(input, output) {
  output$map <- renderPlot({
    data <- switch(input$var,
                   "Percent White" = counties$white,
                   "Percent Black" = counties$black,
                   "Percent Hispanic" = counties$hispanic,
                   "Percent Asian" = counties$asian)

    percent_map(var = data, color = ?, legend.title = ?, max = ?, min = ?)
  })

}
```

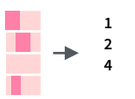
String manipulation with stringr : : **CHEAT SHEET**

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

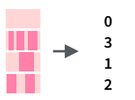
Detect Matches



str_detect(string, **pattern**) Detect the presence of a pattern match in a string. *str_detect(fruit, "a")*



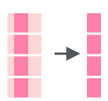
str_which(string, **pattern**) Find the indexes of strings that contain a pattern match. *str_which(fruit, "a")*



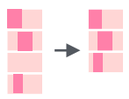
str_count(string, **pattern**) Count the number of matches in a string. *str_count(fruit, "a")*

str_locate(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

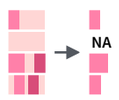
Subset Strings



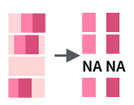
str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. *str_sub(fruit, 1, 3); str_sub(fruit, -2)*



str_subset(string, **pattern**) Return only the strings that contain a pattern match. *str_subset(fruit, "b")*

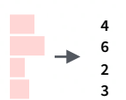


str_extract(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*

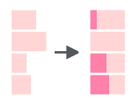


str_match(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**. *str_match(sentences, "(a|the) ([^]+)")*

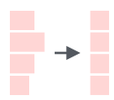
Manage Lengths



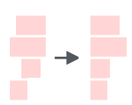
str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*



str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

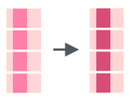


str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*

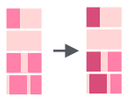


str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

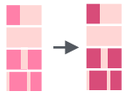
Mutate Strings



str_sub() **<-** value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*



str_replace(string, **pattern**, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*



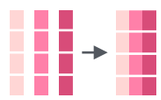
str_replace_all(string, **pattern**, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

str_to_lower(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*

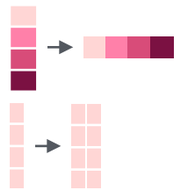
str_to_upper(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*

str_to_title(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

Join and Split



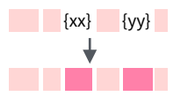
str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*



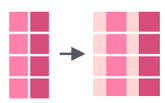
str_c(..., sep = "", **collapse = NULL**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*

str_dup(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

str_split_fixed(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*

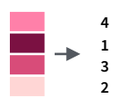


str_glue(…, .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*



str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

Order Strings



str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*

str_view(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

str_view_all(string, **pattern**, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

[1] See **bit.ly/ISO639-1** for a complete list of locales.

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at **stringr.tidyverse.org** • Diagrams from **@LVaudor** • stringr 1.2.0 • Updated: 2017-10

Need to Know

Pattern arguments in stringr are interpreted as regular expressions *after any special characters have been parsed.*

In R, you write regular expressions as *strings*, sequences of characters surrounded by quotes (**""**) or single quotes(**''**).

Some characters cannot be represented directly in an R string . These must be represented as **special characters**, sequences of characters that have a specific meaning., e.g.

Special Character	Represents
\\	\
\"	"
\n	new line

Run **?"'"** to see a complete list

Because of this, whenever a \ appears in a regular expression, you must write it as \\ in the string that represents the regular expression.

Use **writeLines**() to see how R views your string after all special characters have been parsed.

writeLines("\\.")
 # \.

writeLines("\\ is a backslash")
 # \ is a backslash

INTERPRETATION

Patterns in stringr are interpreted as regexs To change this default, wrap the pattern in one of:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...)
Modifies a regex to ignore cases, match end of lines as well of end of strings, allow R comments within regex's , and/or to have . match everything including \n.
str_detect("I", regex("i", TRUE))

fixed() Matches raw bytes but will miss some characters that can be represented in multiple ways (fast). *str_detect("\u0130", fixed("i"))*

coll() Matches raw bytes and will use locale specific collation rules to recognize characters that can be represented in multiple ways (slow).
str_detect("\u0130", coll("i", TRUE, locale = "tr"))

boundary() Matches boundaries between characters, line_breaks, sentences, or words.
str_split(sentences, boundary("word"))

Regular Expressions -

Regular expressions, or *regexps*, are a concise language for describing patterns in strings.

MATCH CHARACTERS

see <- function(rx) str_view_all("abc ABC 123\t.!?\\(){}\n", rx)

string (type this)	**regexp** (to mean this)	**matches** (which matches this)	**example**	
	a (etc.)	a (etc.)	see("a")	abc ABC 123 .!?\(){}
\\.	**\.**	.	see("\\.")	abc ABC 123 .!?\(){}
\\!	**\!**	!	see("\\!")	abc ABC 123 .!?\(){}
\\?	**\?**	?	see("\\?")	abc ABC 123 .!?\(){}
\\\\	****	\	see("\\\\")	abc ABC 123 .!?\(){}
\\(	**\(**	(	see("\\(")	abc ABC 123 .!?\(){}
\\)	**\)**	)	see("\\)")	abc ABC 123 .!?\(){}
\\{	**\{**	{	see("\\{")	abc ABC 123 .!?\(){}
\\}	**\}**	}	see("\\}")	abc ABC 123 .!?\(){}
\\n	**\n**	new line (return)	see("\\n")	abc ABC 123 .!?\(){}
\\t	**\t**	tab	see("\\t")	abc ABC 123 .!?\(){}
\\s	**\s**	any whitespace (**\S** *for non-whitespaces*)	see("\\s")	abc ABC 123 .!?\(){}
\\d	**\d**	any digit (**\D** *for non-digits*)	see("\\d")	abc ABC 123 .!?\(){}
\\w	**\w**	any word character (**\W** *for non-word chars*)	see("\\w")	abc ABC 123 .!?\(){}
\\b	**\b**	word boundaries	see("\\b")	abc ABC 123 .!?\(){}
	[:digit:] [1]	digits	see("[:digit:]")	abc ABC 123 .!?\(){}
	[:alpha:] [1]	letters	see("[:alpha:]")	abc ABC 123 .!?\(){}
	[:lower:] [1]	lowercase letters	see("[:lower:]")	abc ABC 123 .!?\(){}
	[:upper:] [1]	uppercase letters	see("[:upper:]")	abc ABC 123 .!?\(){}
	[:alnum:] [1]	letters and numbers	see("[:alnum:]")	abc ABC 123 .!?\(){}
	[:punct:] [1]	punctuation	see("[:punct:]")	abc ABC 123 .!?\(){}
	[:graph:] [1]	letters, numbers, and punctuation	see("[:graph:]")	abc ABC 123 .!?\(){}
	[:space:] [1]	space characters (i.e. \s)	see("[:space:]")	abc ABC 123 .!?\(){}
	[:blank:] [1]	space and tab (but not new line)	see("[:blank:]")	abc ABC 123 .!?\(){}
	.	every character except a new line	see(".")	abc ABC 123 .!?\(){}

[1] Many base R functions require classes to be wrapped in a second set of **[ ]**, e.g. **[[:digit:]]**

[:space:]
↵ new line

[:blank:]
space
tab

.

[:graph:]

[:punct:]

. , : ; ? ! \ | / ` = * + - ^
_ ~ " ' [ ] { } ( ) < > @ # $

[:alnum:]

[:digit:]

0 1 2 3 4 5 6 7 8 9

[:alpha:]

[:lower:]

a b c d e f
g h i j k l
m n o p q r
s t u v w x
z

[:upper:]

A B C D E F
G H I J K L
M N O P Q R
S T U V W X
Z

ALTERNATES

alt <- function(rx) str_view_all("abcde", rx)

regexp	**matches**	**example**	
ab|d	or	alt("ab|d")	abcde
[abe]	one of	alt("[abe]")	abcde
[^abe]	anything but	alt("[^abe]")	abcde
[a-c]	range	alt("[a-c]")	abcde

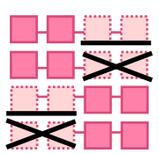
ANCHORS

anchor <- function(rx) str_view_all("aaa", rx)

regexp	**matches**	**example**	
^a	start of string	anchor("^a")	aaa
a$	end of string	anchor("a$")	aaa

LOOK AROUNDS

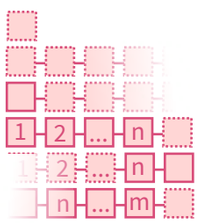
look <- function(rx) str_view_all("bacad", rx)



regexp	**matches**	**example**	
a(?=c)	followed by	look("a(?=c)")	bacad
a(?!c)	not followed by	look("a(?!c)")	bacad
(?<=b)a	preceded by	look("(?<=b)a")	bacad
(?<!b)a	not preceded by	look("(?<!b)a")	bacad

QUANTIFIERS

quant <- function(rx) str_view_all(".a.aa.aaa", rx)



regexp	**matches**	**example**	
a?	zero or one	quant("a?")	.a.aa.aaa
a*	zero or more	quant("a*")	.a.aa.aaa
a+	one or more	quant("a+")	.a.aa.aaa
a{n}	exactly **n**	quant("a{2}")	.a.aa.aaa
a{n,}	**n** or more	quant("a{2,}")	.a.aa.aaa
a{n, m}	between **n** and **m**	quant("a{2,4}")	.a.aa.aaa

GROUPS

ref <- function(rx) str_view_all("abbaab", rx)

Use parentheses to set precedent (order of evaluation) and create groups

regexp	**matches**	**example**	
(ab|d)e	sets precedence	alt("(ab|d)e")	abcde

Use an escaped number to refer to and duplicate parentheses groups that occur earlier in a pattern. Refer to each group by its order of appearance

string (type this)	**regexp** (to mean this)	**matches** (which matches this)	**example** (the result is the same as ref("abba"))	
\\1	**\1** (etc.)	first () group, etc.	ref("(a)(b)\\2\\1")	abbaab

Examination

Linköping University, Department of Computer and Information Science, Statistics

Course code and name	732A98 Visualization
Date and time	2016-10-21, 08.00-12.00
Assisting teacher	Oleg Sysoev
Allowed aids	Book “Visualize this” by N. Yau
Grades:	A=19-20 points
	B=16-18 points
	C=11-15 points
	D=9-10 points
	E=7-8 points
	F=0-6 points

Provide a detailed report that includes necessary code, plots, conclusions and interpretations.

All R plots in your report should contain necessary graphical elements (title, axis,…) and some elements of design (color). If necessary or if it is stated in the assignment, use Inkscape for improving the plot (don’t spend much time on this).

In GGobi plots, background color should be changed to white and the default accent color should be black (see Color&Glyph settings), and the resulting plot can be copied into your report.

When plotting maps, no Inkscape processing is required.

Assignment 1 (11p)

A psychological experiment was performed to investigate the influence of scents on the ability of solving a task. File ***scents.csv*** contains information about time subjects required to complete a pencil and paper labyrinth when they were smelling a floral scent and when they were not. The variables are:

- ID
- Sex: M=male, F=female

- Smoker: Y if subject smoked, N if did not
- Opinion: "pos" if subject found the odor inherently positive, "indiff" if indifferent, "neg" if inherently negative
- Order: 1 if did unscented trials first, 2 if did scented trials first
- U-Trial 1: length of time required for first unscented trial
- U-Trial 2 : length of time required for second unscented trial
- U-Trial 3: length of time required for third unscented trial
- S-Trial 1 : length of time required for first scented trial
- S-Trial 2 : length of time required for second scented trial
- S-Trial 3: length of time required for third scented trial

1. Use ggplot to create a publication quality scatterplot that shows connection between U-Trial 2 and U-Trial 3. Perform a detailed analysis of the plot and interpret your findings. **(2p)**
2. Researchers have also got access to the file **pie.pdf** showing the distribution of the ages for the participants of the experiment. Improve this plot further in Inkscape by
    a. Changing the white pie slice color to green and edge colors to white
    b. Shortening the categories and placing them inside of the pie slices
        i. You can shorten "[31.7-48.3]" as "32-48"
    c. Adding necessary text elements
    d. Rotating the pie so that the edge between the blue and the green pie slices is vertical

    Save the resulting file as PDF and put it in the solutions folder together with your report.

    Report the main reason of why the original picture **pie.pdf** can be misleading. **(3p)**

3. Present all time variables by means of a heatmap in which the rows and columns are permuted by employing the single-link hierarchical clustering. Describe the clusters, report which variables are important in defining these clusters and make interpretations of the clusters. Find and report outliers. Create a variable Cluster_ID in your data that contains the cluster number (according to your visual findings) or "-1" if the observation is an outlier. **(3p)**
4. Visualize these new data in GGobi, create a 2D-tour involving all time variables and find a projection that shows clusters (at least two, do not spend much time on this). Brush the observations by Cluster_ID and check how this coloring is related to 2D-tour clusters and to the histograms of the variables Sex, Smoking and Opinion. Interpret your findings. Which clustering strategy (heatmap or 2D-tour) do you think is more reliable and why? **(3p)**

Assignment 2 (9p)

File **rates.csv** contains information about the currency exchange rates (standardized to [0,1]) versus Euro at various time points.

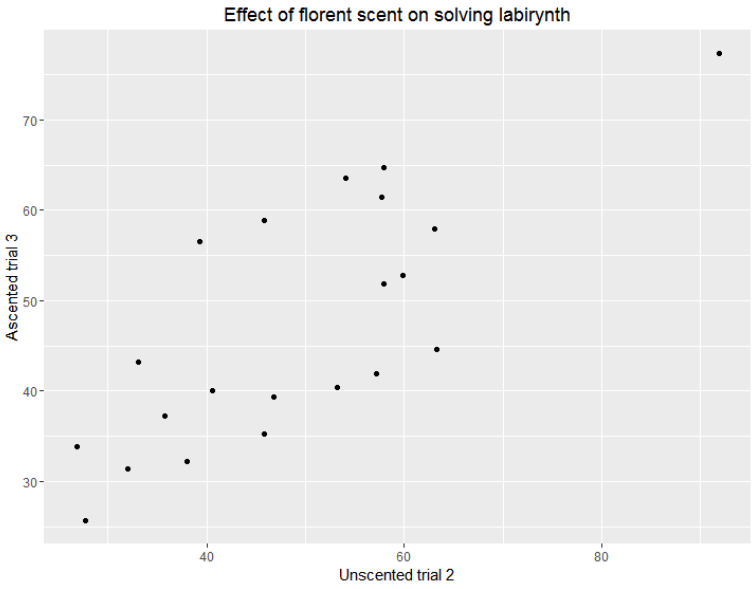
1. Compute a new variable MonthC as Month+Day/32 and subset the data to first four months. Make a shingle with 5% overlap based on MonthC and the reduced data. Use

this shingle and the reduced data in order to produce a trellis plot with 4 panels where each panel shows a boxplot of the DKK exchange rate. Analyze this plot and interpret your findings. **(3p)**

2. Produce a similar plot to step 1 where you condition on the Month instead of the shingle variable. Compare the results with step 1 and comment what advantages there are in using shingles. **(1p)**
3. File **rates2.csv** contains interpolated values of the exchange rates. Make a plot containing the original values of SEK rates (from rates.csv) and the interpolated values of SEK versus time. By looking at the plot, provide some positive and negative arguments of using interpolated values instead of the original ones for doing an animation. For which time period can the interpolation be misleading? **(2p)**
4. Use all available exchange rate values from rate2.csv to produce an animation where the exchange rates for a specific time point are visualized as a bar chart. Watch the animation then and reorder the bars so that the final video corresponds to the standards of the good animation. Motivate your choice. List also at least three noteworthy features from the resulting animation. **(3p)**

Put the resulting mp4 file in the solutions folder together with your report.

U1



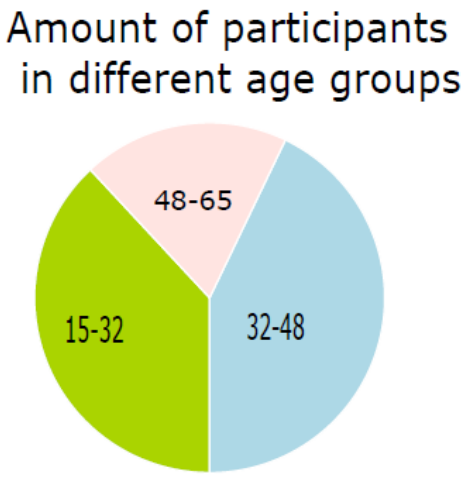
Shape is quite linear. Interpret: reasonable since repeating a trial should lead to similar results

Strength of connection between variables is moderate. Interpret: some other factors affected the quality of experiment

There is a dense area with low Unscented trials 2 and 3 (cluster?) and an area having high values of ascented trials 2 and 3 (cluster?) It seems that the individuals are split into two groups depending on how fast they solve the test.

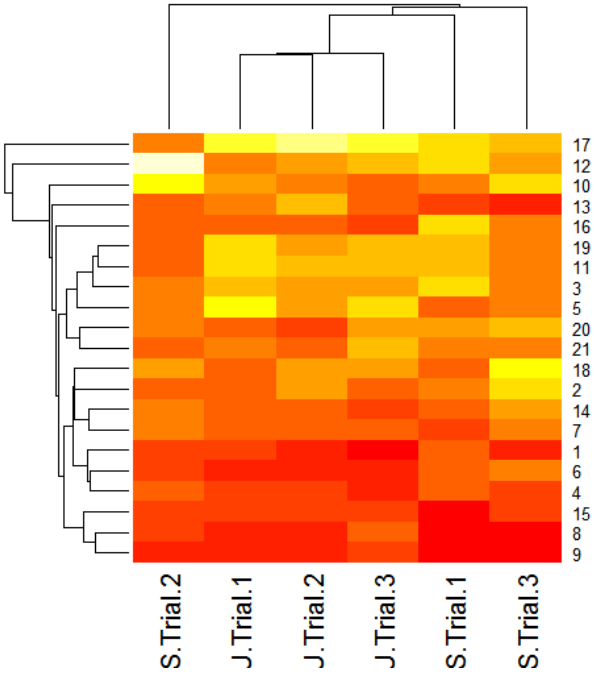
There is a clear outlier with respect to both variables. Interp: a person that solved both times very slowly.

2



In the original plot it is difficult to see whether the green or the blue slice is larger

3



Students may see different clusters. Motivation is most important. Minimalistic example:

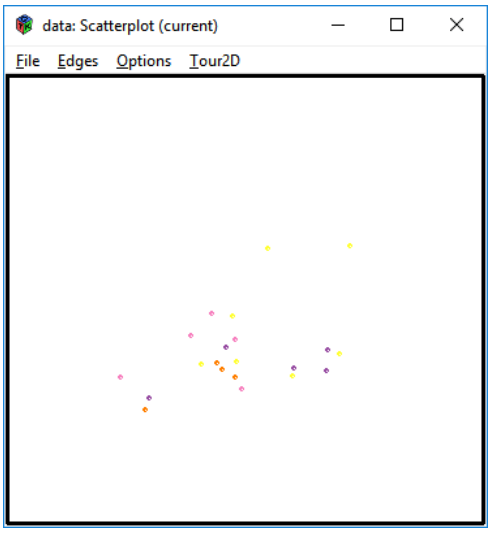
Cluster 1: observations 9,8,15,4,6,1. Variables: S.Trial 2, U.trial 1,2,3. This cluster are persons that have low values of time for unscented tests mostly

Cluster 2: observations 7,14,2,18,21. Moderate values of the unscented variables and high third scented trial

Cluster 3: observations 5,3,11,9,16. Large values of U tests , moderate levels of third scented trial

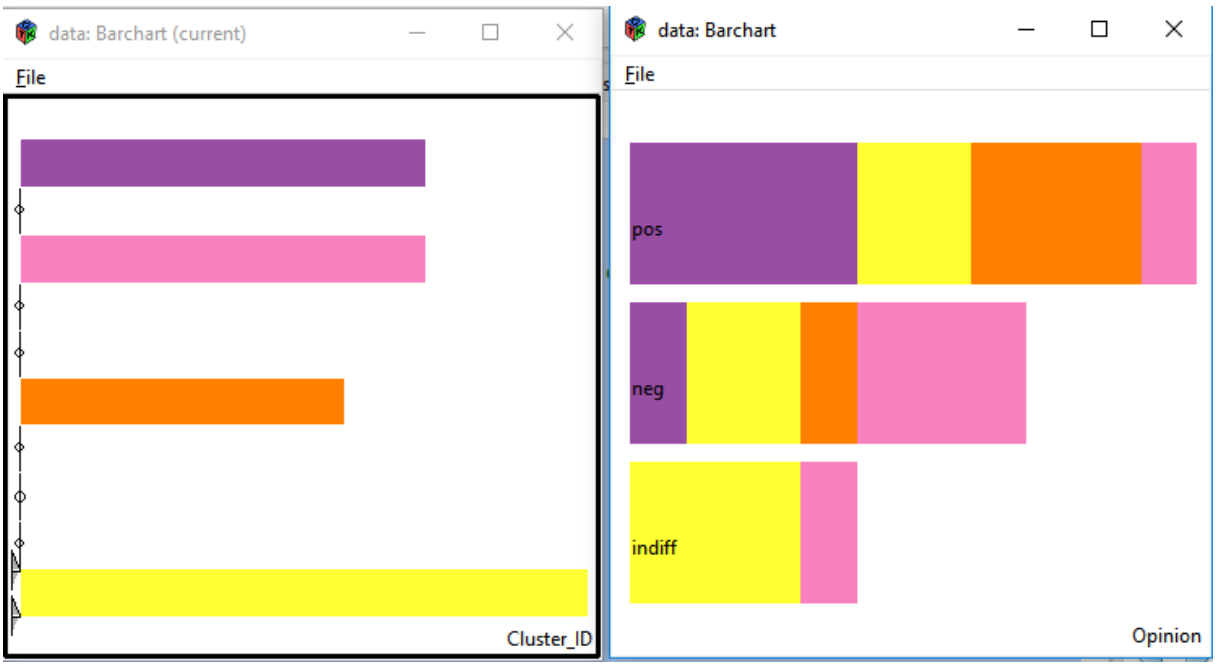
Other observations – outliers.

4



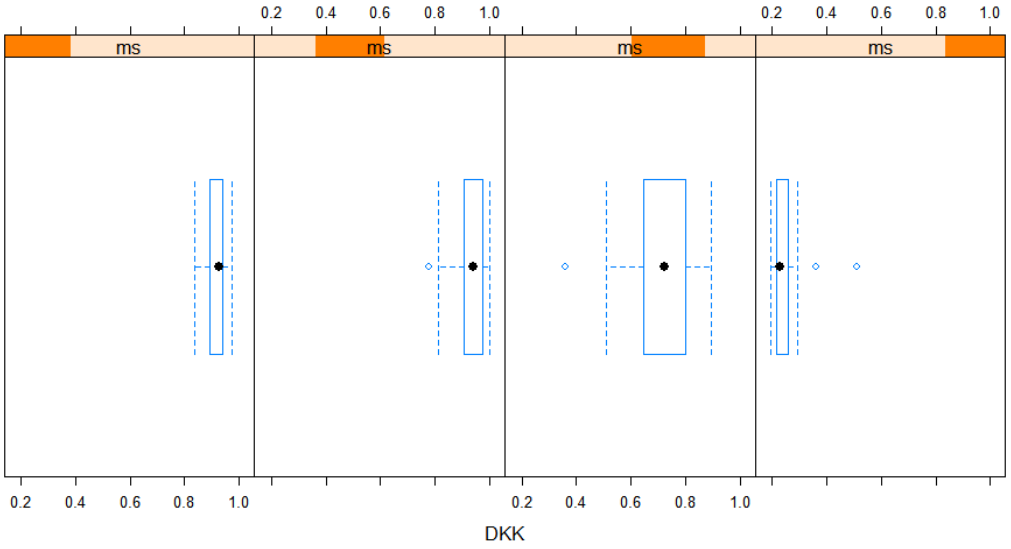
No clear connection between clusters and Cluster ID.

Reliability: different answers possible, motivation is important.



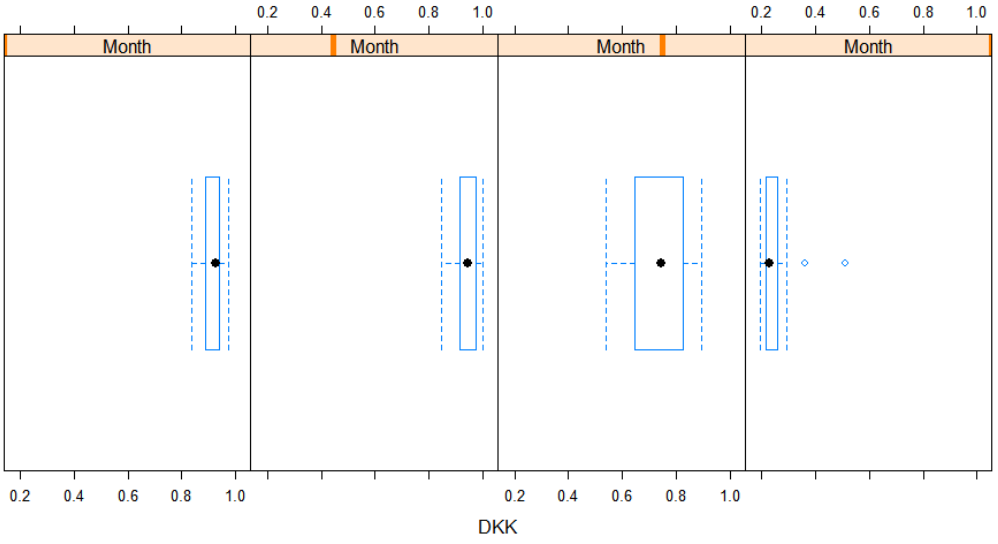
Cluster Id 1 and 3 are not in the indiff group. They had effect of smell. Those are the clusters that had extreme values in the timing variables (either large or low).

Assignment 2

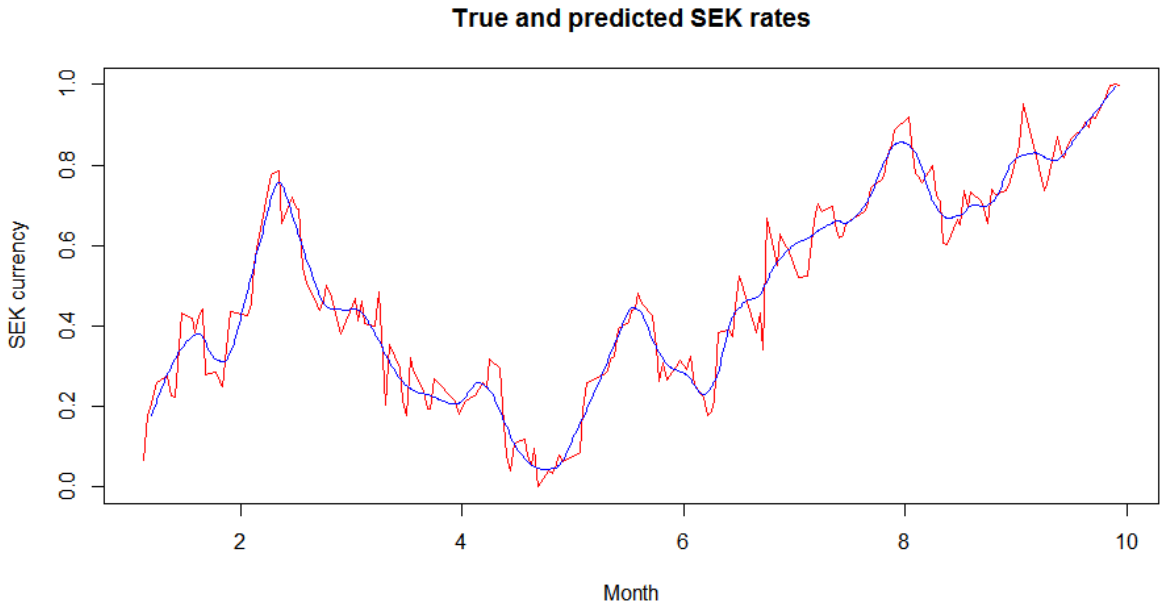


Minimalistic interpretation: One can see that during the first month the currency rates are pretty high, whether during the second month there are still a lot of high rates but the rates are going down. There is one outlier. During the third month the rates continue go down, and there is one more outlier.

During the fourth month, the rates are lowest, but there are two outliers (high values)



Compared to the previous plot, no outliers are seen in month 2 and 3, they are thus on the boundary of each month. Advantage of shingles – we detected an new outlier which was on the boundary of the regions.



Positive: the transitions will be smooth, no quick small changes that hamper the perception.

Disadvantages: some interesting details are hidden, only trend is seen but some smaller variations in trend are smoothed out. Around month 9, a peak is smoothed out.