732A98: Visualization - Cheatbook

Anubhav Dikshit (anudi287)

02 Nov 2018

```
# Loading required R packages
library(ggplot2)
library(plotly)
library(shiny)
library(gridExtra)
library(xlsx)
library(MASS)
library(sf)
library(akima)
library(scales)
library(seriation)
library(dplyr)
library(crosstalk)
library(GGally)
library(tm)
library(wordcloud)
library(RColorBrewer)
library(htmltools)
library(tourr)
library(reshape)
library(ggraph)
library(igraph)
library(visNetwork)

Sys.setenv('MAPBOX_TOKEN' = 'pk.eyJ1IjoibGFrc2hpZGFhIiwiYSI6ImNqbWIyOHN2NTRlZ3kzam10aTljeGNybWgifQ
.8EG9Y6r024e-TGk79f7GpA')
```

1 Reading Data

2 Data Mugging

2.0.1 Quantile Computation

```
get_outliers <- function(x){
  quantile_values = quantile(x, probs = c(0.25, 0.75))
  q1 = quantile_values["25%"]
  q3 = quantile_values["75%"]

  return(c(which((x > (q3+1.5*(q3-q1)))), which(x < (q1-1.5*(q3-q1)))))
}
```

2.0.2 Scaling the Data

```
baseball_scaled <- scale(baseball_data[,3:length(baseball_data)])
```

2.0.3 Distance Matrix between rows

```
distance_matrix <- dist(baseball_scaled, method = "euclidean")
```

2.0.4 Non-metric MDS

```
mds_result <- isoMDS(distance_matrix, k=2, p=2)
```

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

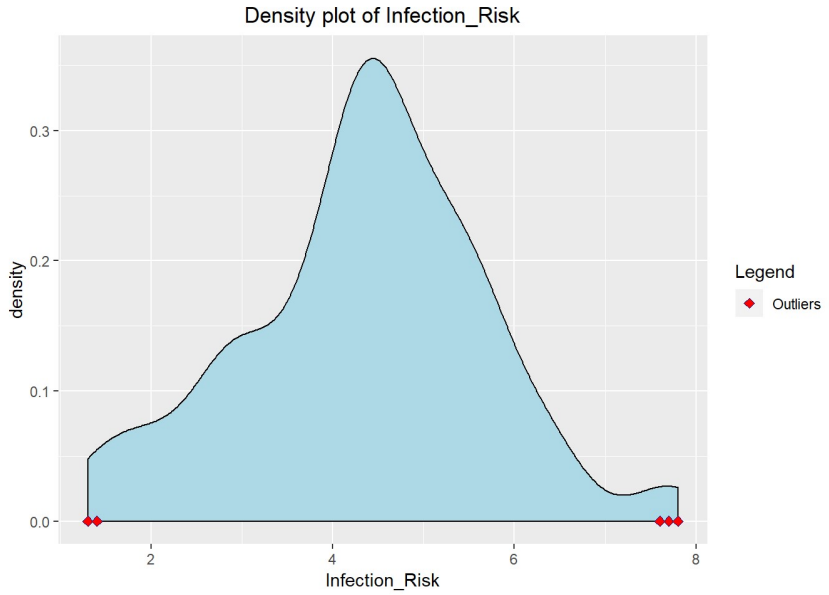
```
coords <- mds_result$points
coords_mds <- as.data.frame(coords)
baseball_data_with_mds <- baseball_data
baseball_data_with_mds$MDS_V1 <- coords_mds$V1
baseball_data_with_mds$MDS_V2 <- coords_mds$V2
```

3 Single Plots

3.1 Density Plot

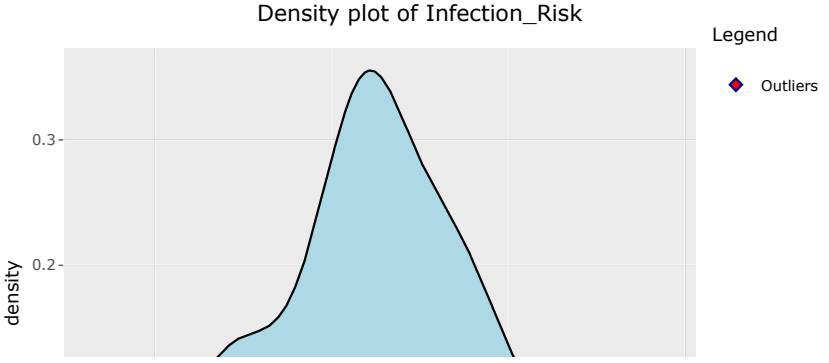
3.1.1 Density Plot with Outlier Highlight using GGplot2

```
density_plot_infection_risk = ggplot(senic_data) +
  ggtitle("Density plot of Infection_Risk")  +
  geom_density(aes(x=Infection_Risk), fill = "lightblue") +
  geom_point(data=senic_data[get_outliers(senic_data$Infection_Risk),],
             aes(x=Infection_Risk, y=0, colour="Outliers"),
             shape=23, size=2, fill="red") +
  scale_color_manual(values = c("darkblue","black")) +
  labs(colour="Legend") +
  theme(plot.title = element_text(hjust = 0.5), legend.position = "right")

density_plot_infection_risk
```



3.1.2 Density Plot with Outlier Highlight using Plotly (converting from ggplot2)

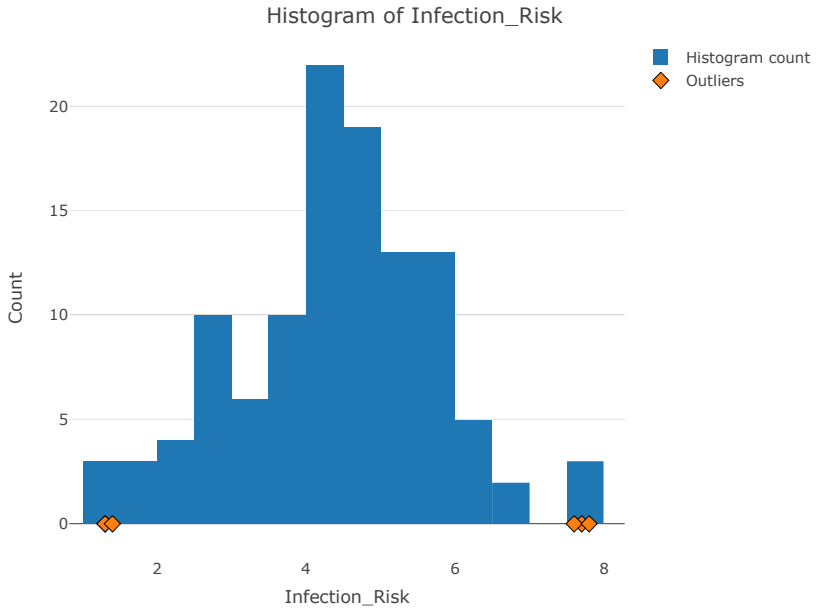
```
x <- ggplotly(p=density_plot_infection_risk)

x
```



3.2 Histogram Plot

3.2.1 Histogram plot with Outlier Highlight using Plotly

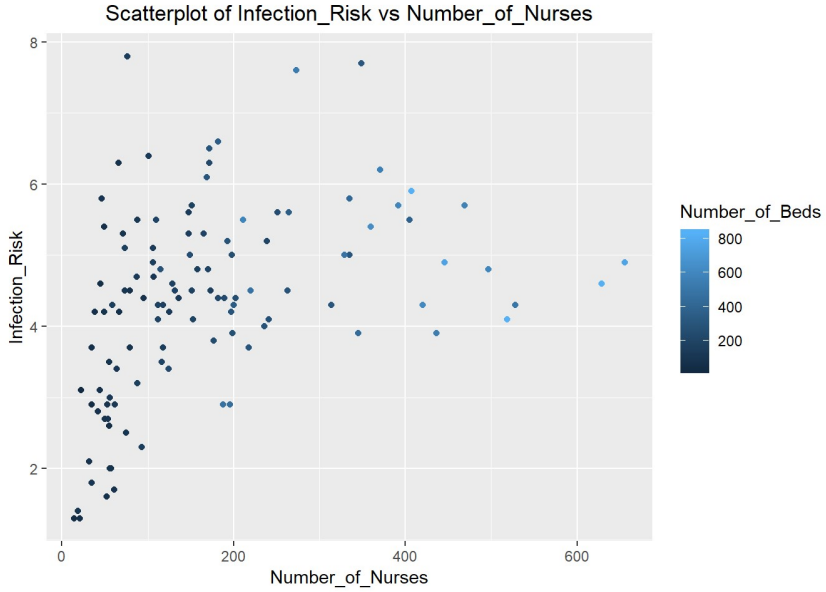
```
outliers = senic_data[get_outliers(senic_data$Infection_Risk),c("Infection_Risk")]
senic_data$zero = 0

plot_ly(senic_data, x=~Infection_Risk) %>%
  add_histogram(name="Histogram count") %>%
  filter(is.element(Infection_Risk, outliers)) %>%
  add_markers(x=~Infection_Risk,y=~zero, name="Outliers",
              marker=list(symbol="diamond", size=10, line = list(color="black", width=1))) %>%
  layout(title="Histogram of Infection_Risk", yaxis=list(title="Count"))
```



3.3 Scatter Plot

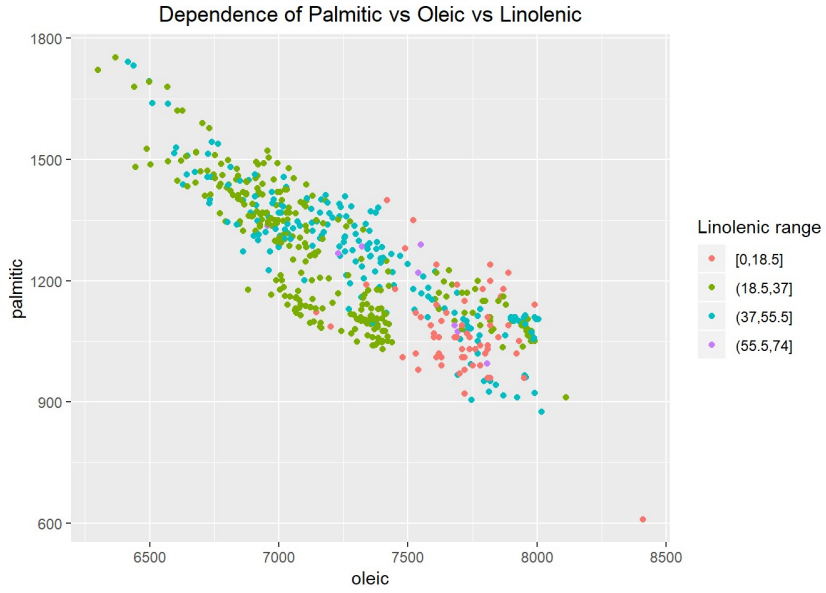
3.3.1 Simple scatter plot with colour

```
ggplot(senic_data) + geom_point(aes(x=Number_of_Nurses, y=Infection_Risk, color=Number_of_Beds)) +
  ggtitle("Scatterplot of Infection_Risk vs Number_of_Nurses") +
  theme(plot.title = element_text(hjust = 0.5))
```



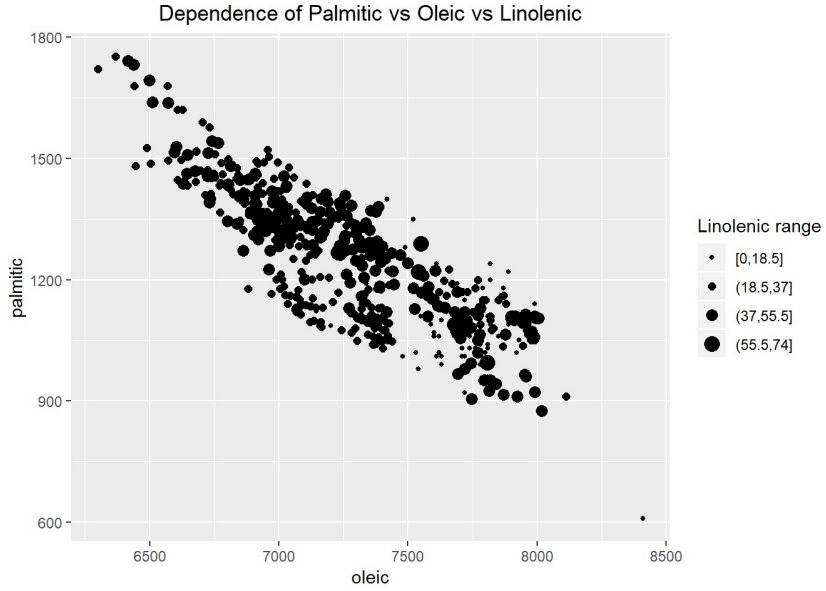
3.3.2 Scatter Plot with Discreetization (split a variable into classes)

```
ggplot(olive_data) +
  geom_point(aes(x = oleic, y = palmitic,
                 color=cut_interval(olive_data$linolenic, n = 4))) +
  ggtitle("Dependence of Palmitic vs Oleic vs Linolenic") +
  theme(plot.title = element_text(hjust = 0.5)) +
  labs(color = 'Linolenic range')
```



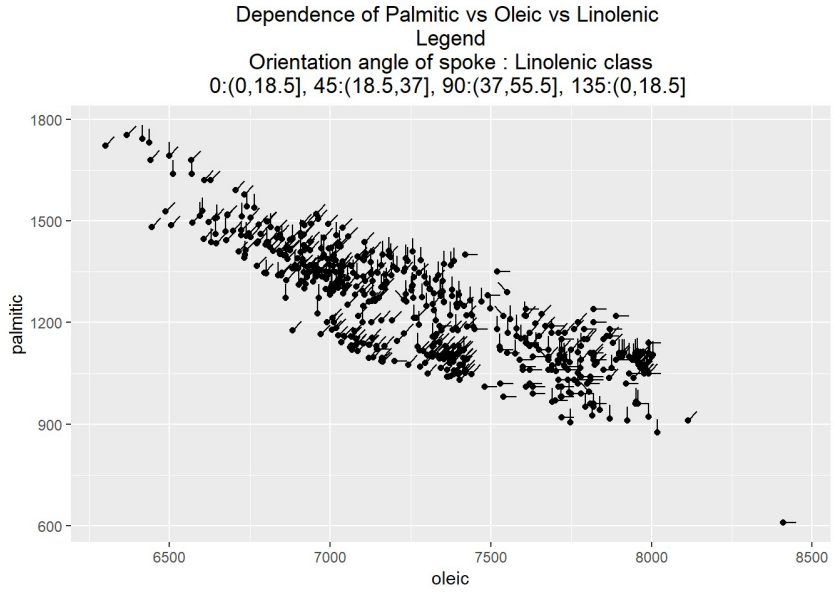
3.3.3 Scatter plot size varied

```
ggplot(olive_data) + geom_point(aes(x = oleic, y = palmitic, size = cut_interval(linolenic, n = 4)
)) +
  ggtitle("Dependence of Palmitic vs Oleic vs Linolenic") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_size_manual(name = "Linolenic range", values = c(1, 2, 3, 4))
```



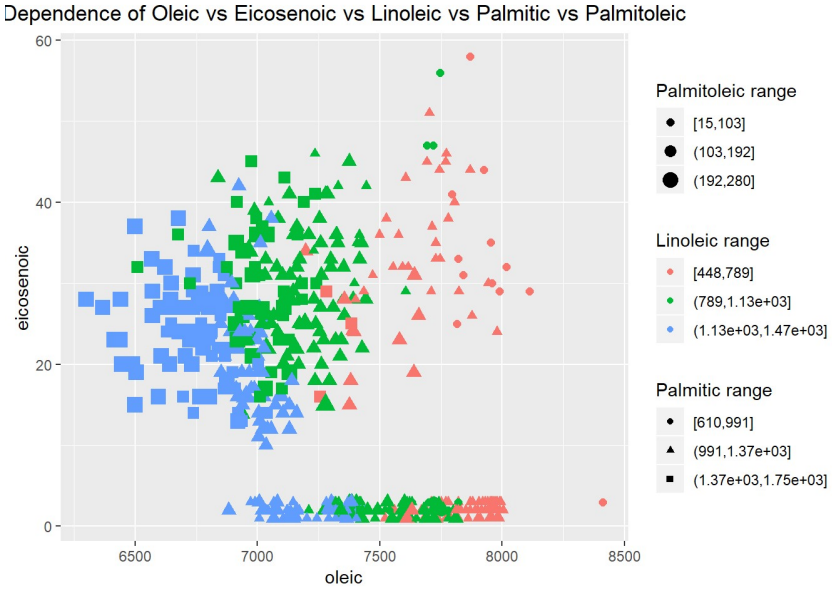
3.3.4 Scatter plot angle varied

```
# Pre-processing - Setting angle values based on category
olive_data$linolenic_class <- cut_interval(olive_data$linolenic, n = 4)
levels(olive_data$linolenic_class) <- (0:3) * (pi/4)
olive_data$linolenic_class <- as.numeric(as.character(olive_data$linolenic_class))

ggplot(olive_data, aes(x=oleic, y=palmitic)) + geom_point() +
  geom_spoke(aes(angle = olive_data$linolenic_class), radius=40) +
  ggtitle("Dependence of Palmitic vs Oleic vs Linolenic
Legend
Orientation angle of spoke : Linolenic class
0:(0,18.5], 45:(18.5,37], 90:(37,55.5], 135:(0,18.5] ") +
  theme(plot.title = element_text(hjust = 0.5))
```



3.3.5 Scatter plot with color, size and shape

```
ggplot(olive_data)+
  geom_point(aes(x = oleic, y = eicosenoic, color = cut_interval(linoleic, n = 3),
                           shape = cut_interval(palmitic, n = 3),
                           size = cut_interval(palmitoleic, n = 3))) +
  scale_size_manual(values = c(2,3,4)) +
  labs(shape = "Palmitic range", color = "Linoleic range", size = "Palmitoleic range") +
  ggtitle("Dependence of Oleic vs Eicosenoic vs Linoleic vs Palmitic vs Palmitoleic") +
  theme(plot.title = element_text(hjust = 0.5))
```

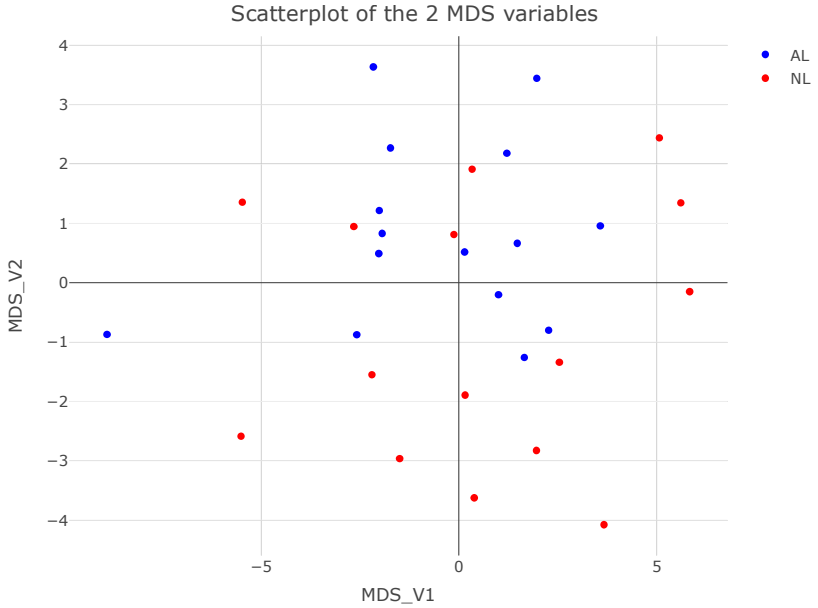


3.3.6 Scatter Plot of MDS

```
baseball_scaled <- scale(baseball_data[,3:length(baseball_data)])

### Distance Matrix between rows
distance_matrix <- dist(baseball_scaled, method = "euclidean")

### Non-metric MDS
mds_result <- isoMDS(distance_matrix, k=2, p=2)
```

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

```
coords <- mds_result$points
coords_mds <- as.data.frame(coords)
baseball_data_with_mds <- baseball_data
baseball_data_with_mds$MDS_V1 <- coords_mds$V1
baseball_data_with_mds$MDS_V2 <- coords_mds$V2



plot_ly(baseball_data_with_mds, x=~MDS_V1, y=~MDS_V2) %>%
  add_markers(color=~League, colors = c("blue", "red"),
              text=baseball_data_with_mds$Team, hoverinfo="text") %>%
  layout(title="Scatterplot of the 2 MDS variables")
```



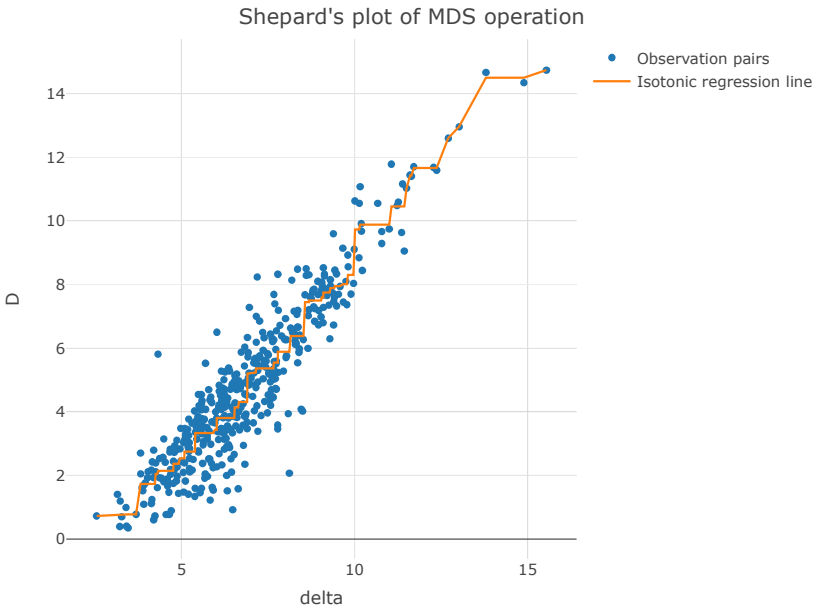
3.4 Shepard Plot

3.4.1 Shepard plot shows the fit of MDS, the distance in original dataset vs. distance in reordered dataset should be similar

```
baseball_scaled <- scale(baseball_data[,3:length(baseball_data)])

### Distance Matrix between rows
distance_matrix <- dist(baseball_scaled, method = "euclidean")
mds_result <- isoMDS(distance_matrix, k=2, p=2)
```

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

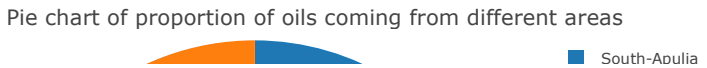
```
coords <- mds_result$points

shp <- Shepard(distance_matrix, coords)
delta <- as.numeric(distance_matrix)
D <- as.numeric(dist(coords, method = "euclidean"))

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, name="Observation pairs", hoverinfo = 'text',
              text = ~paste('Obj 1: ',
                            rownames(baseball_data_with_mds)[index1],
                            '<br> Obj 2: ',
                            rownames(baseball_data_with_mds)[index2])) %>%
  add_lines(x=~shp$x, y=~shp$yf, name="Isotonic regression line") %>%
  layout(title="Shepard's plot of MDS operation")
```



3.5 Pie Charts

3.5.1 Simple pie chart

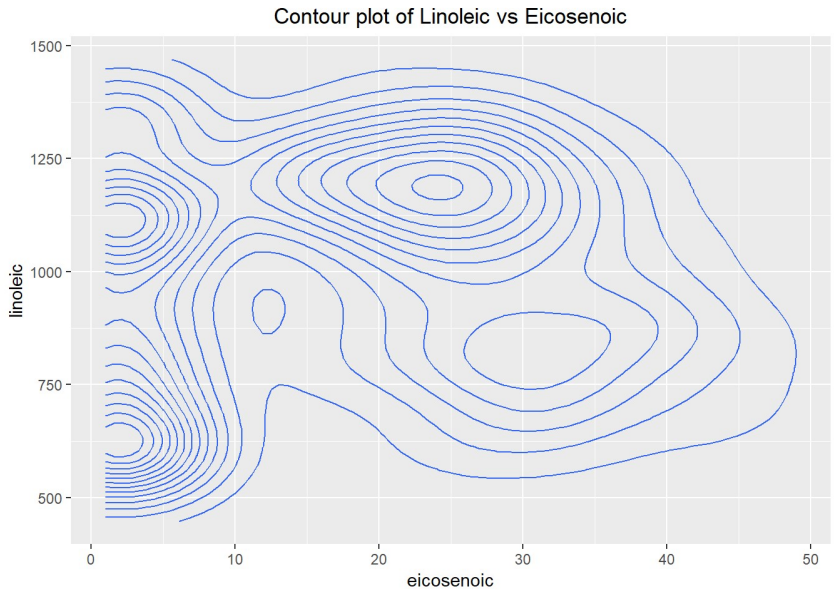
```
plot_ly(olive_data,labels=~Area,type='pie',textinfo = "none") %>%
  layout(title = "Pie chart of proportion of oils coming from different areas")
```



3.6 2D density contour plot

3.6.1 Simple 2D plot using ggplot2

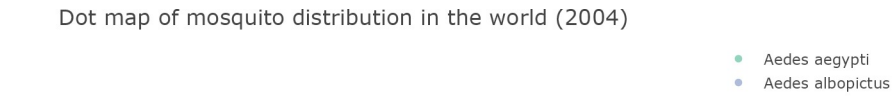
```
ggplot(olive_data)+geom_density_2d(aes(x=eicosenoic, y=linoleic)) +
  ggtitle("Contour plot of Linoleic vs Eicosenoic") +
  theme(plot.title = element_text(hjust = 0.5))
```



3.7 Dot Map Plots (World Map)

3.7.1 Dot Map (Map with scatter plots)

```
plot_mapbox(aegypti_data[aegypti_data$YEAR == 2004,], lat = ~Y, lon = ~X) %>%
  add_markers(color = ~VECTOR, hoverinfo = "text",
              text = ~paste(COUNTRY), alpha = 0.7) %>%
  layout(title = "Dot map of mosquito distribution in the world (2004)")
```

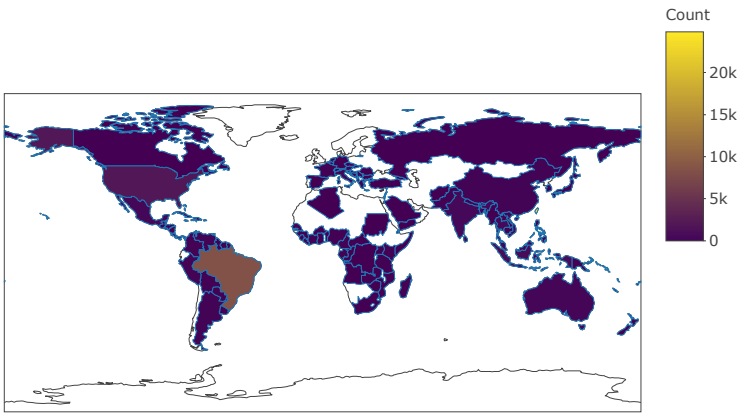


3.8 Choropleth Map

3.8.1 Chorpleth Map with Equirectangular Projection

```
# Data aggregation
country_aggregate = aggregate(aegypti_data[,c("COUNTRY", "COUNTRY_ID")],
                              by = list(aegypti_data$COUNTRY, aegypti_data$COUNTRY_ID), FUN=length
)
country_aggregate$COUNTRY = NULL
colnames(country_aggregate) = c("COUNTRY", "COUNTRY_ID", "Count")

plot_geo(country_aggregate) %>% add_trace(
    z = ~Count,
    text = ~COUNTRY, locations = ~COUNTRY_ID) %>%
  layout(title = "Choropleth plot of number of mosquitoes",
    geo = list(projection = list(type = "equirectangular")))
```

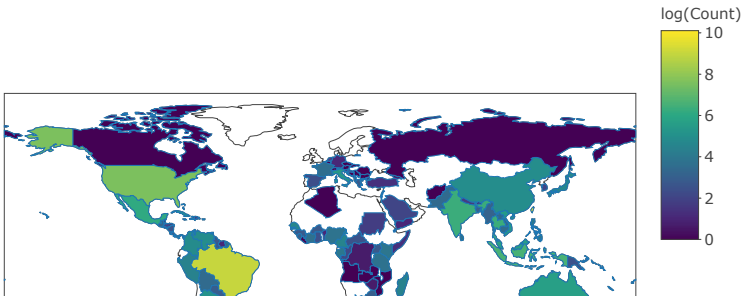
Choropleth plot of number of mosquitoes



3.8.2 Choropleth plot with Equirectangular Projection and log

```
plot_geo(country_aggregate) %>%
  add_trace(
    z = ~log(Count) ,
    text = ~paste(COUNTRY, "\n Count: ", Count), locations = ~COUNTRY_ID,
    hoverinfo = "text"
  ) %>%
  layout(
    title = "Choropleth plot of number of mosquitoes",
    geo = list(projection = list(type = "equirectangular")))
```

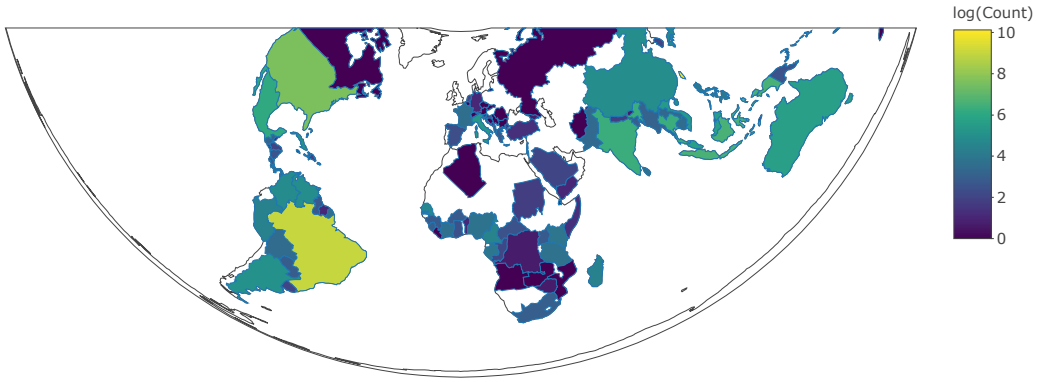
Choropleth plot of number of mosquitoes



3.8.3 Choropleth plot with Conic Area Projection and log

```
plot_geo(country_aggregate) %>%
  add_trace(
    z = ~log(Count) ,
    text = ~paste(COUNTRY, "\n Count: ", Count), locations = ~COUNTRY_ID
  ) %>%
  layout(
    title = "Choropleth plot of number of mosquitoes",
    geo = list(projection = list(type = "conic equal area")))
```

Choropleth plot of number of mosquitoes



3.8.4 Choropleth plot using custom shape files(sf file)

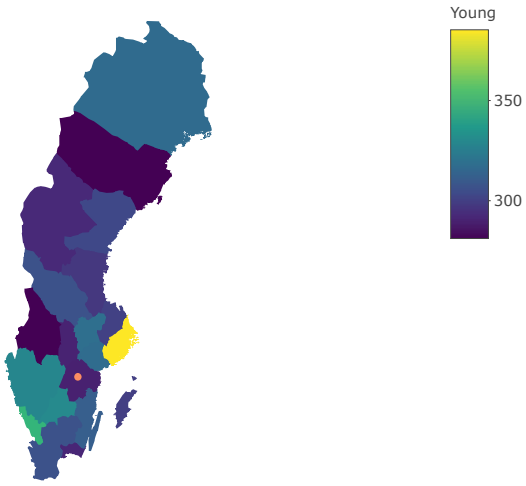
```
swe_data = read.csv("000000KD.csv")

swe_data_processed = data.frame(region = unique(swe_data$region))
swe_data_split = split(swe_data, swe_data$age)
for (i in seq_along(swe_data_split)) {
  swe_data_processed[[names(swe_data_split)[i]]] = merge(swe_data_split[[i]],
                                                         swe_data_processed$region,
                                                         by.x = 'region',
                                                         by.y = 1, all = T)$X2016
}

colnames(swe_data_processed) = c("region", "Young", "Adult", "Senior")
swe_data_processed$region = gsub(" county", "", swe_data_processed$region)
swe_data_processed$region = gsub("\\d{2} ", "", swe_data_processed$region)
swe_data_processed$region = gsub("Örebro", "Orebro", swe_data_processed$region)
rownames(swe_data_processed) = swe_data_processed$region
rds = readRDS('gadm36_SWE_1_sf.rds')

rds$Young = swe_data_processed[rds$NAME_1, "Young"]
plot_ly() %>% add_sf(data = rds, split = ~NAME_1,
                     color = ~Young, showlegend = F, alpha = 1) %>%
  layout(title = "Choropleth plot of mean income of Young age group")
```

Choropleth plot of mean income of Young age group

3.8.5 Choropleth plot with a custom maker

```
rds$Young = swe_data_processed[rds$NAME_1, "Young"]
plot_ly() %>% add_sf(data = rds, split = ~NAME_1, color = ~Young,
                     showlegend = F, alpha = 1) %>%
  add_markers(x = 15.621373, y = 58.410809, color = "red",
              hoverinfo = "text", text = "We are here!") %>%
  layout(title = "Choropleth plot of mean income of Young age group")
```

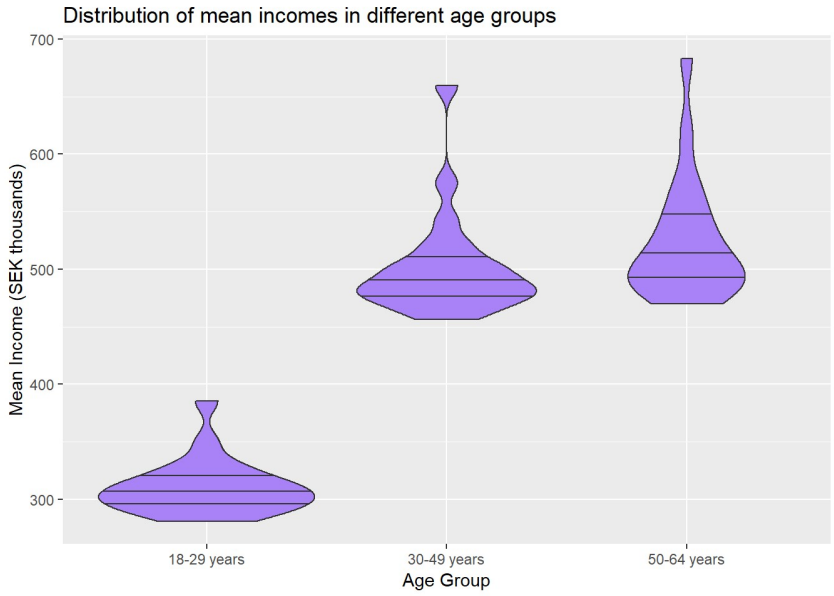
Choropleth plot of mean income of Young age group



3.9 Violin Plots

3.9.1 Violin Plot Simple 2 variable (one cateogrical and one numeric)

```
swe_data_processed = data.frame(region = unique(swe_data$region))

swe_data_split = split(swe_data, swe_data$age)
for (i in seq_along(swe_data_split)) {
  swe_data_processed[[names(swe_data_split)[i]]] = merge(swe_data_split[[i]],
                                                         swe_data_processed$region,
                                                         by.x = 'region',
                                                         by.y = 1, all = T)$X2016
}

colnames(swe_data_processed) = c("region", "Young", "Adult", "Senior")
swe_data_processed$region = gsub(" county", "", swe_data_processed$region)
swe_data_processed$region = gsub("\\d{2} ", "", swe_data_processed$region)
swe_data_processed$region = gsub("Örebro", "Orebro", swe_data_processed$region)
rownames(swe_data_processed) = swe_data_processed$region


ggplot(swe_data) +
  geom_violin(aes(x = age, y = X2016),
              draw_quantiles = c(0.25, 0.5, 0.75), fill = "#A982F7")  +
  scale_x_discrete(labels=c("18_29" = "Young",
                            "30_49" = "Adult",
                            "50_69" = "Senior"),
                   name = "Age Group") +
  ylab("Mean Income (SEK thousands)") +
  ggtitle("Distribution of mean incomes in different age groups")
```

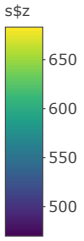


3.10 3D surface plots

3.10.1 3D surface plots using plotly

```
s = interp(swe_data_processed$Young, swe_data_processed$Adult,
           swe_data_processed$Senior, duplicate = "mean")
plot_ly(x=~s$x, y=~s$y, z=~s$z, type="surface") %>% layout(
  scene=list(
  xaxis = list(title = "Young"),
  yaxis = list(title = "Adult"),
  zaxis = list(title = "Senior")),
  title = "3D surface plot of income distribution")
```

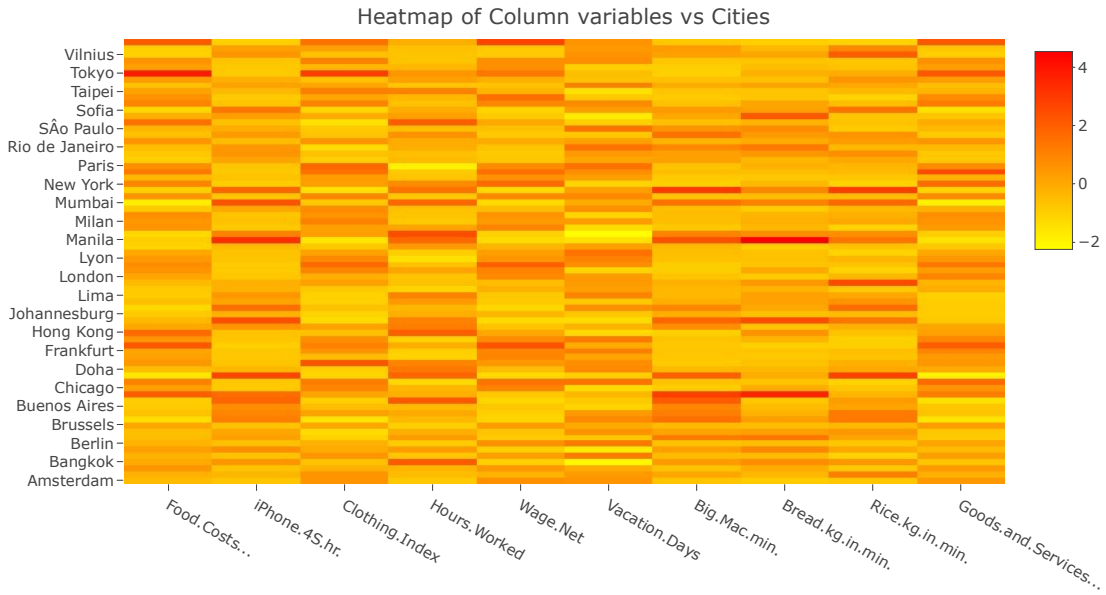
3D surface plot of income distribution



3.11 Heat Map

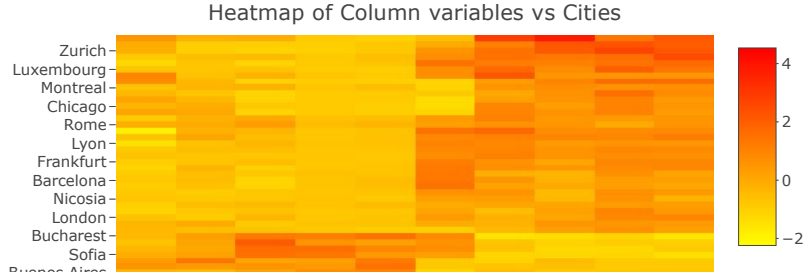
3.11.1 Heat Map without reordering

```
ubs_scaled = scale(ubs_data)
plot_ly(x = colnames(ubs_scaled), y = rownames(ubs_scaled),
        z = ubs_scaled, type = "heatmap",
        colors = colorRamp(c("yellow", "red"))) %>%
  layout(title = "Heatmap of Column variables vs Cities")
```



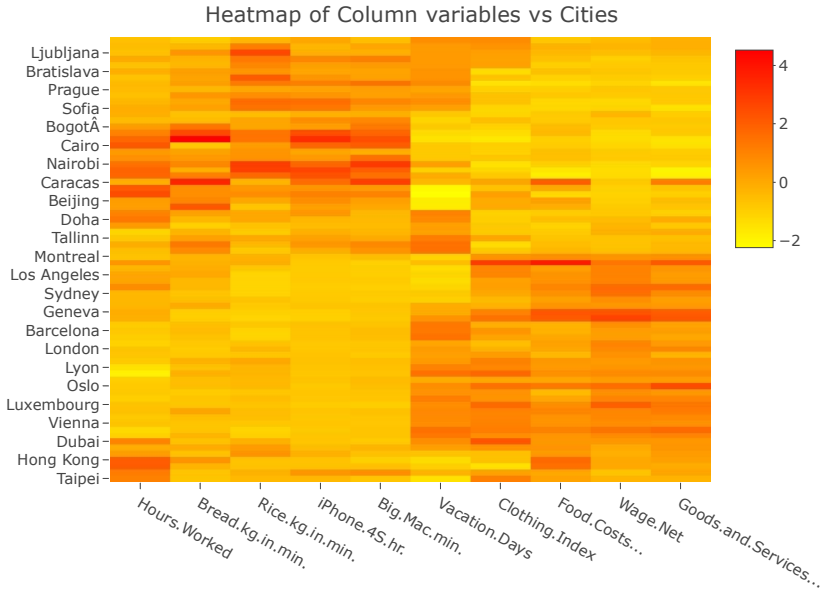
3.11.2 Heat Map with ordering using Hiearchical Clustering (HC) using Euclidean distance

```
ubs_scaled = scale(ubs_data)
row_dist_euc = dist(ubs_scaled, method = "euclidean")
col_dist_euc = dist(t(ubs_scaled), method = "euclidean")

seriate_row_euc = seriate(row_dist_euc, "HC")
seriate_col_euc = seriate(col_dist_euc, "HC")
ord_row_euc = get_order(seriate_row_euc)
ord_col_euc = get_order(seriate_col_euc)

ubs_reordered_euc = ubs_scaled[rev(ord_row_euc),ord_col_euc]

plot_ly(x = colnames(ubs_reordered_euc),
        y = rownames(ubs_reordered_euc),
        z = ubs_reordered_euc, type = "heatmap",
        colors = colorRamp(c("yellow", "red"))) %>%
  layout(title = "Heatmap of Column variables vs Cities")
```



3.11.3 Heat Map with ordering using Corellation using Euclidean distance

```
row_dist_cor = as.dist(1 - cor(t(ubs_scaled)))
col_dist_cor = as.dist(1 - cor(ubs_scaled))

seriate_row_cor = seriate(row_dist_cor, method = "HC")
seriate_col_cor = seriate(col_dist_cor, method = "HC")
ord_row_cor = get_order(seriate_row_cor)
ord_col_cor = get_order(seriate_col_cor)

ubs_reordered_cor = ubs_scaled[rev(ord_row_cor), ord_col_cor]

plot_ly(x = colnames(ubs_reordered_cor),
        y = rownames(ubs_reordered_cor),
        z = ubs_reordered_cor, type = "heatmap",
        colors = colorRamp(c("yellow", "red"))) %>%
  layout(title = "Heatmap of Column variables vs Cities")
```



3.11.4 Heat Map with ordering using Hamiltonian Path Length using Traveling Salesman Problem (TSP)

```
seriate_row_tsp = seriate(row_dist_euc, "TSP")
seriate_col_tsp = seriate(col_dist_euc, "TSP")
ord_row_tsp = get_order(seriate_row_tsp)
ord_col_tsp = get_order(seriate_col_tsp)

ubs_reordered_tsp = ubs_scaled[rev(ord_row_tsp), ord_col_tsp]

plot_ly(x = colnames(ubs_reordered_tsp),
        y = rownames(ubs_reordered_tsp),
        z = ubs_reordered_tsp, type="heatmap",
        colors = colorRamp(c("yellow", "red"))) %>%
  layout(title = "Heatmap of Column variables vs Cities
         (reordered by TSP/Hamiltonian Path Length/Euclidean Distance)")
```

3.11.5 Comparison of two solvers for heatmap

```
#Hamiltonian Path Length
ord_tsp = seriate(row_dist_euc, "TSP")
ham_tsp = criterion(row_dist_euc, order = ord_row_tsp, "Path_length")
paste("Hamiltonian Path Length : ", ham_tsp)
```

```
## [1] "Hamiltonian Path Length :  122.825971195561"
```

```
#Gradient Measure
gm_tsp = criterion(row_dist_euc, order=ord_tsp, "Gradient_raw")
paste("Gradient Measure : ", gm_tsp)
```

```
## [1] "Gradient Measure :  56956"
```

```
#Hamiltonian Path Length
ord_hc = seriate(row_dist_euc, "HC")
ham_hc = criterion(row_dist_euc, order = ord_hc, "Path_length")
paste("Hamiltonian Path Length : ", ham_hc)
```

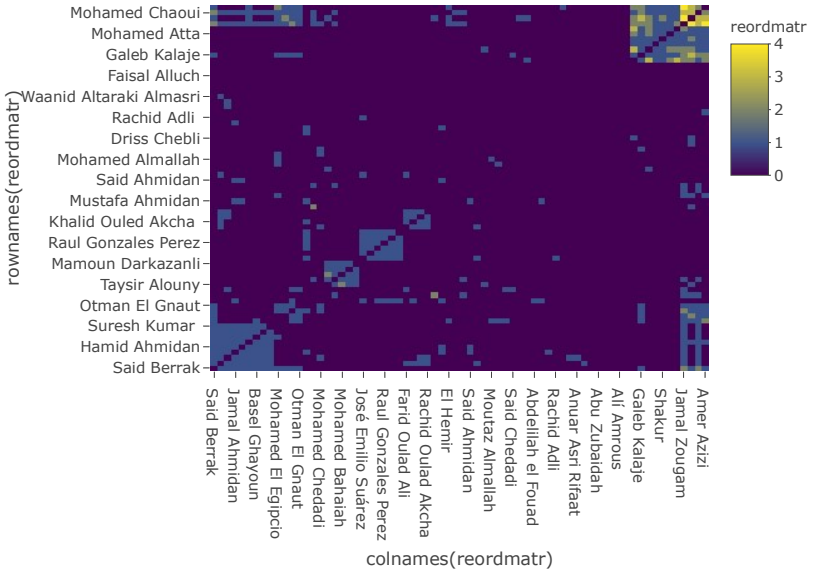
```
## [1] "Hamiltonian Path Length :  144.492476381981"
```

```
#Gradient Measure
gm_hc = criterion(row_dist_euc, order = ord_hc, "Gradient_raw")
paste("Gradient Measure : ", gm_hc)
```

```
## [1] "Gradient Measure :  58432"
```

3.11.6 Heat Map using Adjacency

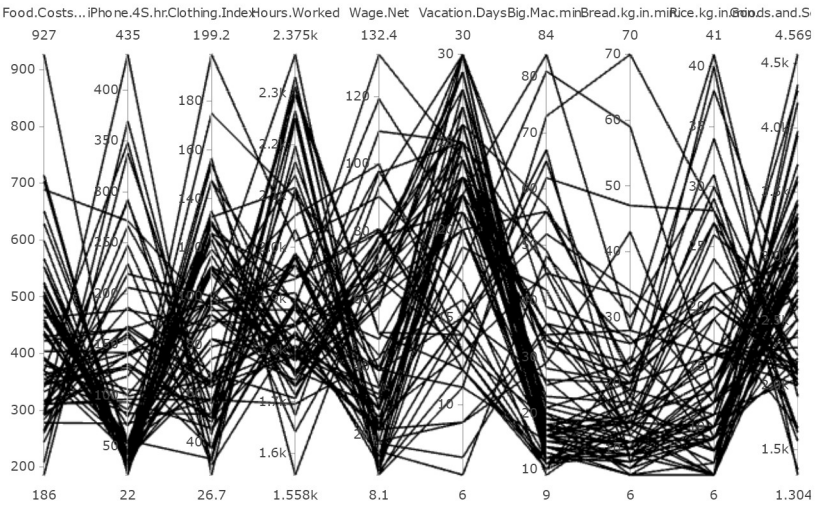
```
colnames(links) <- c("from","to","strength")
links <- links[order(links$from, links$to),]
nodes$id <- 1:70
rownames(links) <- NULL

colnames(nodes)[2] <- "BombingGrp"

#Size of links based on "strength of links"
links$width <- links$strength*3

#Nodes colored based on Bombing Group
nodes$label <- nodes$V1
nodes$group <- nodes$BombingGrp

#Size of nodes proportional to the number of connections
graph <- graph.data.frame(links, directed = F)
strength_value <- strength(graph)
nodes$value <- strength_value[match(nodes$id, names(strength_value))]


for(i in 1:nrow(links)){
  links$from_name[i] <- nodes$V1[links$from[i]]
  links$to_name[i] <- nodes$V1[links$to[i]]
}

links <- links[,c("from_name","to_name","from","to","strength","width")]
nodes1 <- nodes
net <- graph_from_data_frame(d=links, vertices=nodes, directed=F)
ceb <- cluster_edge_betweenness(net)
nodes1$group <- ceb$membership


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

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")
```



3.12 Parallel Plot

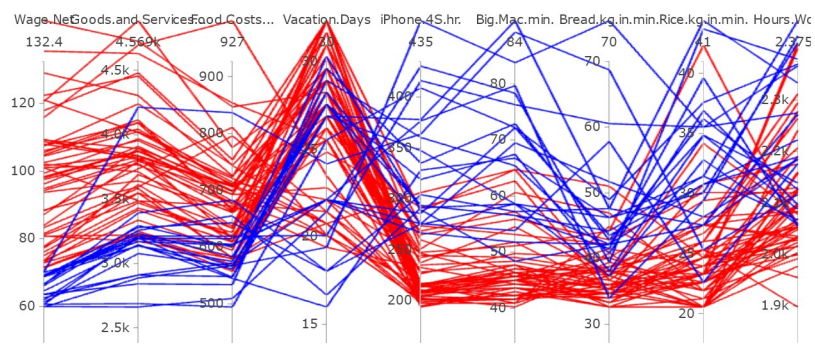
3.12.1 Parallel Plot using Plotly

```
# Function to create dimension list based on given variable order
get_dimension_list <- function(df, col_order){
  dim_list = list()
  i = 1
  for (col in col_order){
    dim_list[[i]] = list(label = col, values = df[[col]])
    i = i + 1
  }

  return(dim_list)
}

# Create dimension list for unordered data
unord_dim_list = get_dimension_list(ubs_data, colnames(ubs_data))

# Create dimension list for reordered data
var_order = c("Cloting.Index", "Wage.Net", "Goods.and.Services...", "Food.Costs...",
              "Vacation.Days", "iPhone.4S.hr.", "Big.Mac.min.",
              "Bread.kg.in.min.", "Rice.kg.in.min.", "Hours.Worked")

reord_dim_list = get_dimension_list(ubs_data, var_order)

# Brush different clusters with different colors
ubs_data$cluster = 1
ubs_data[ubs_data$Wage.Net < 24 , "cluster"] = 2

plot_ly(data = ubs_data, type = 'parcoords',
        dimensions = unord_dim_list) %>%
  layout(title = "")
```



3.12.2 Parallel Plot using Plotly and highlight one line

```
plot_ly(data = ubs_data, type = 'parcoords',
        line = list(color = ~cluster, colorscale = list(c(0,"red"), c(1,"blue"))),
        dimensions = reord_dim_list) %>%
  layout(title = "")
```



3.13 Radar Plots

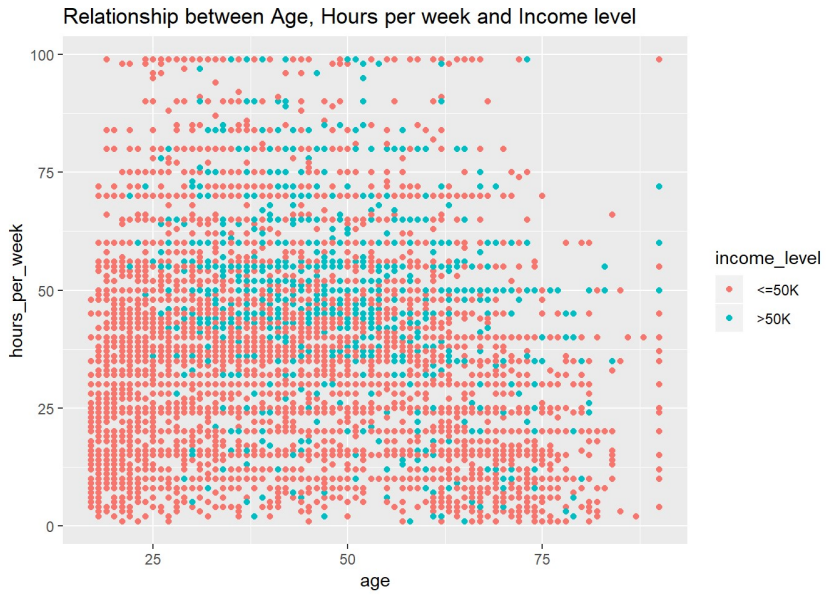
3.13.1 Radar Plots using Scatterpolar

```
ubs_reordered_euc <- as.data.frame(ubs_reordered_euc)
ubs_reordered_euc$City = rownames(ubs_reordered_euc)

ubs_reordered_euc <- ubs_reordered_euc%>%tidyr::gather(variable, value, -City, factor_key=T)%>%arr
ange(City)

first_col_order <- rownames(ubs_reordered_euc)
second_col_order <- unique(ubs_reordered_euc$variable)

ubs_reordered_euc$City <- factor(ubs_reordered_euc$City , levels = first_col_order)
ubs_reordered_euc$variable <- factor(ubs_reordered_euc$variable , levels = second_col_order)

ubs_reordered_euc <- ubs_reordered_euc[order(ubs_reordered_euc$City, ubs_reordered_euc$variable),]

radar_plot <- ubs_reordered_euc %>% ggplot(aes(x=variable, y=value, group=City)) + geom_polygon(fi
ll="blue") + coord_polar() + theme_bw() + facet_wrap(~ City) + theme(axis.text.x = element_text(si
ze = 5))

ggsave("radar_plot.png", width = 40, height = 60, units = "cm")
```

3.14 Trellis Plots

3.14.1 Trellis Plots using ggplot2

```
ggplot(adult_data) +
  geom_point(aes(x = age, y = hours_per_week, color = income_level)) +
  ggtitle("Relationship between Age, Hours per week and Income level")
```

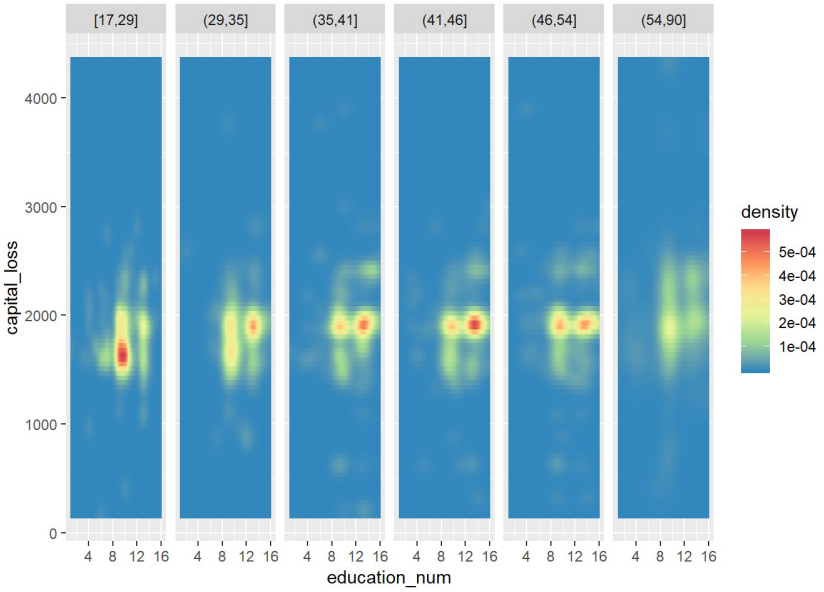


3.14.2 Trellis Plots with facet/condition on a variable

```
ggplot(adult_data) +
  geom_point(aes(x = age, y = hours_per_week, color = income_level)) +
  facet_grid(income_level~.) +
  ggtitle("Relationship between Age, Hours per week for each Income level")
```

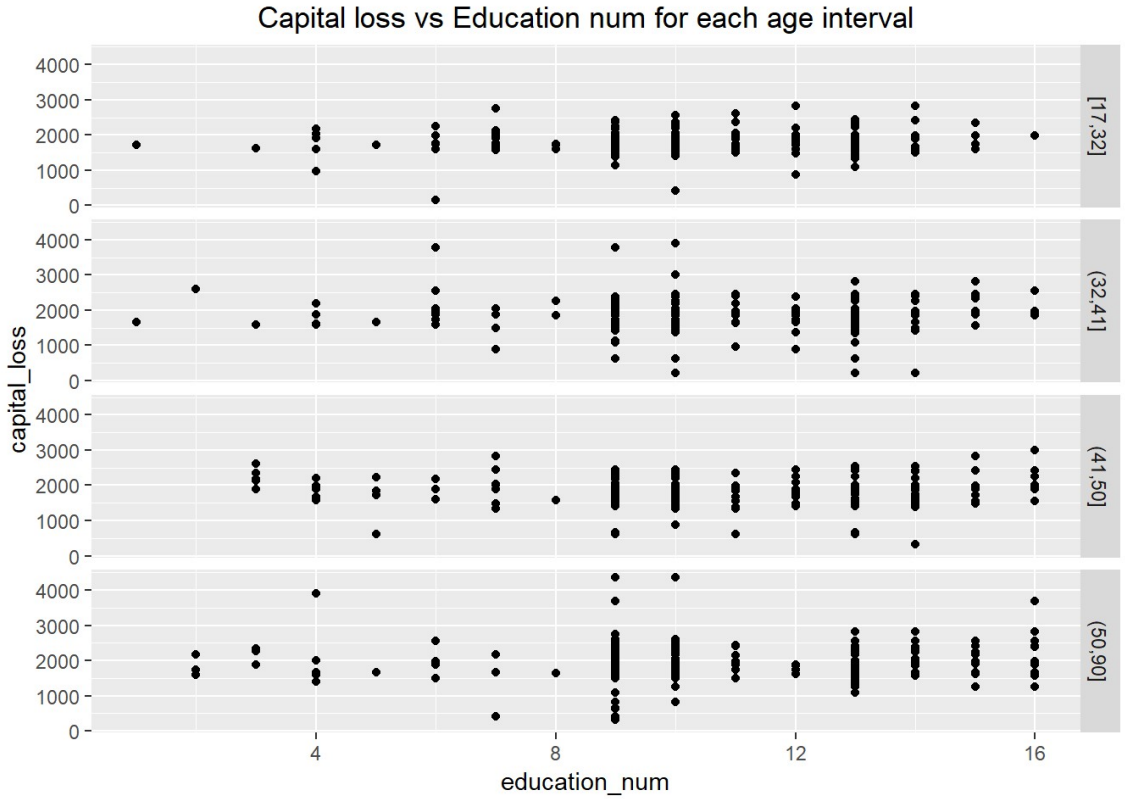
3.14.3 Trellis Plot with raster-type-2d-density plot

```
capital_loss_data = adult_data[adult_data$capital_loss != 0,]

ggplot(capital_loss_data) +
  stat_density_2d(geom = "raster",
                  aes(x = education_num, y = capital_loss,
                      fill = stat(density)), contour = FALSE) +
  scale_fill_distiller(palette = "Spectral") +
  facet_grid(cols = vars(cut_number(age, 6)))
```



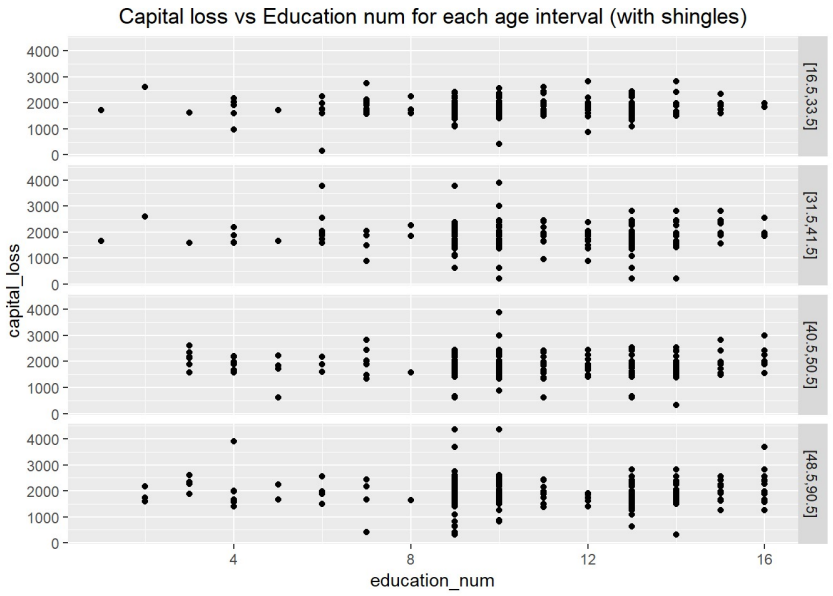
3.14.4 Trellis Plot with raster-type-2d-density plot with discreetization(cut_number)

```
ggplot(capital_loss_data) +
  geom_point(aes(x = education_num, y = capital_loss)) +
  facet_grid(vars(cut_number(age, 4))) +
  ggtitle("Capital loss vs Education num for each age interval") +
  theme(plot.title = element_text(hjust = 0.5))
```



3.14.5 Trellis Plot with Shingles

```
age_ranges = lattice::equal.count(capital_loss_data$age, number = 4, overlap = 0.1)

age_range_matrix = matrix(unlist(levels(age_ranges)), ncol=2, byrow = T)
age_range_df = data.frame(Lower = age_range_matrix[,1],
                          Upper = age_range_matrix[,2],
                          Interval = factor(1:nrow(age_range_matrix)))

index = c()
age_interval = c()
for(i in 1:nrow(age_range_df)){
  interval_name = paste("[", age_range_df$Lower[i], ",",
                        age_range_df$Upper[i], "]", sep="")
  indices_in_interval = which(capital_loss_data$age >= age_range_df$Lower[i] &
                              capital_loss_data$age <= age_range_df$Upper[i])
  index = c(index, indices_in_interval)
  age_interval = c(age_interval, rep(interval_name, length(indices_in_interval)))
}

shingles_df = capital_loss_data[index,]
shingles_df = cbind(shingles_df, age_interval)

ggplot(shingles_df) +
  geom_point(aes(x = education_num, y = capital_loss)) +
  facet_grid(vars(age_interval)) +
  ggtitle("Capital loss vs Education num for each age interval (with shingles)") +
  theme(plot.title = element_text(hjust = 0.5))
```



3.15 Word Clouds

3.15.1 Word Clouds using tm

```
set.seed(1)
par(mfrow=c(1,2))
#Word cloud for positive reviews
positive_data$doc_id=1:nrow(positive_data)
colnames(positive_data)[1]<-"text"

#Here we interpret each line in the document as separate document
mycorpus <- Corpus(DataframeSource(positive_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
v1 <- sort(rowSums(m),decreasing=TRUE) #Sum up the frequencies of each word
v2 <- v1[-1]

d1 <- data.frame(word = names(v1),freq=v1) #Create one column=names, second=frequences
pal1 <- brewer.pal(8,"Dark2")
pal1 <- pal1[-(1:2)] #Create palette of colors
wordcloud(d1$word,d1$freq, scale=c(5,.3),min.freq=2,max.words=100, random.order=F, rot.per=.15, co
lors=pal1, vfont=c("sans serif","plain"))

d2 <- data.frame(word = names(v2),freq=v2) #Create one column=names, second=frequences
pal2 <- brewer.pal(8,"Dark2")
pal2 <- pal2[-(1:2)] #Create palette of colors
wordcloud(d2$word,d2$freq, scale=c(2,.3),min.freq=2,max.words=100, random.order=F, rot.per=.15, co
lors=pal2, vfont=c("sans serif","plain"))
```

4 Linked Plots

4.1 Bar chart and Scatter Plot

4.1.1 Bar chart and scatter plot shared using crosstalk

```
olive_data = read.csv("olive.csv")

olive_data$Region_category <- "Sardinia Island"
olive_data[olive_data$Region == 1, c("Region_category")] <- "North"
olive_data[olive_data$Region == 2, c("Region_category")] <- "South"

ct_olive_2 = SharedData$new(olive_data)

scatter_eico_lino = plot_ly(ct_olive_2, x = ~linoleic, y = ~eicosenoic) %>%
  add_markers(color = I("black"))

bar_region <-plot_ly(ct_olive_2, x = ~Region_category) %>%
  add_histogram() %>% layout(barmode = "overlay")

plots = subplot(scatter_eico_lino, bar_region, titleX = TRUE, titleY = TRUE) %>%
  highlight(on = "plotly_select", dynamic = T, persistent = T, opacityDim = I(1)) %>%
  hide_legend() %>%
  layout(title = "Scatterplot: Eicosenoic vs Linoleic, Histogram: Region")
```

```
## Adding more colors to the selection color palette.
```

```
## We recommend setting `persistent` to `FALSE` (the default) because persistent selection mode ca
n now be used by holding the shift key (while triggering the `on` event).
```

```
bscols(widths = c(12), list(tags$h2("Analysis of variable relationships"),
                            tags$h4("(Eicosenoic, Linoleic, Region, Stearic)"),
                            filter_slider("stearic", "Stearic", ct_olive_2, ~stearic), plots))
```

```
## Setting the `off` event (i.e., 'plotly_relayout') to match the `on` event (i.e., 'plotly_select
ed'). You can change this default via the `highlight()` function.
```

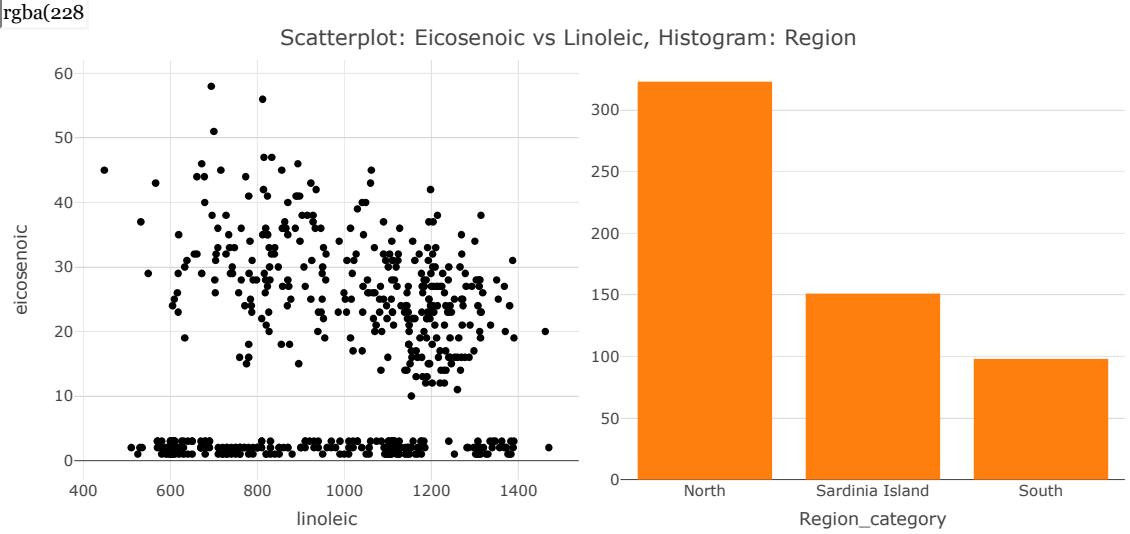
Analysis of variable relationships

(Eicosenoic, Linoleic, Region, Stearic)

Stearic

Brush color

rgba(228

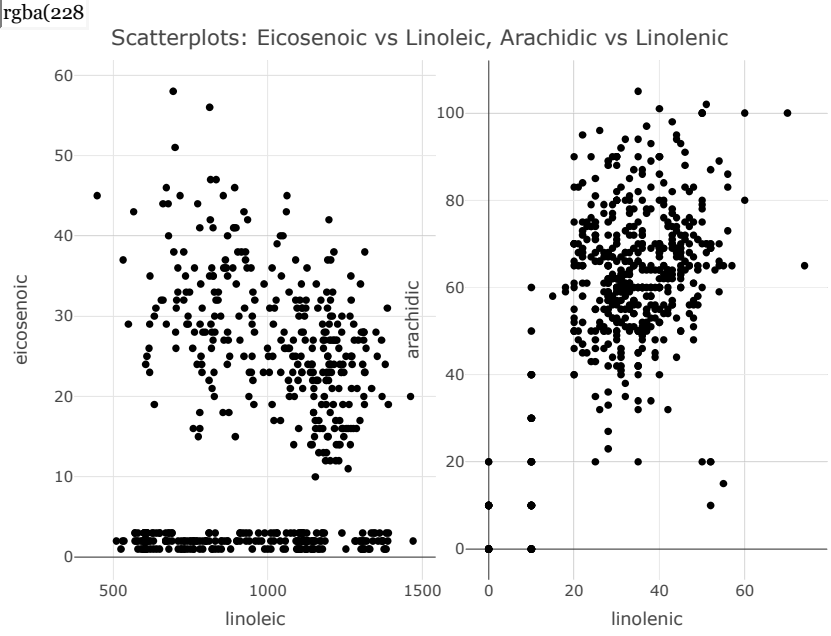


4.1.2 Scatter plot shared using crosstalk

```
ct_olive_3 = SharedData$new(olive_data)

sctr_eico_lino = plot_ly(ct_olive_3, x = ~linoleic, y = ~eicosenoic) %>%
  add_markers(color = I("black"))

sctr_arac_lino = plot_ly(ct_olive_3, x = ~linolenic, y = ~arachidic) %>%
  add_markers(color = I("black"))

subplot(sctr_eico_lino, sctr_arac_lino, titleX = TRUE, titleY = TRUE) %>%
  highlight(on = "plotly_select", dynamic = T, persistent = T, opacityDim = I(1)) %>%
  hide_legend() %>%
  layout(title = "Scatterplots: Eicosenoic vs Linoleic, Arachidic vs Linolenic")
```

Brush color

rgba(228



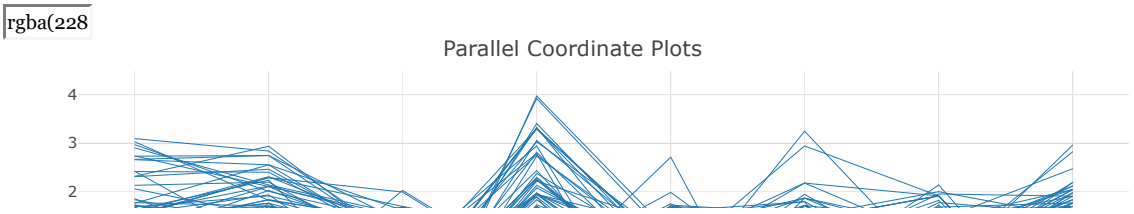
4.1.3 Parallel Chord/Trellis Plot Scatter plot and Bar chart

```
var_order = c("palmitic", "palmitoleic", "linoleic", "stearic",
              "oleic", "linolenic", "arachidic", "eicosenoic")

par_plot = ggparcoord(olive_data[var_order], columns = 1:8)

plot_data = plotly_data(ggplotly(par_plot)) %>% group_by(.ID)
ct_olive_4 = SharedData$new(plot_data, ~.ID, group = "olive")
par_coord_plot = plot_ly(ct_olive_4, x = ~variable, y = ~value) %>%
  add_lines(line = list(width = 0.3))%>%
  add_markers(marker = list(size = 0.3),
              text = ~.ID, hoverinfo = "text") %>%
  layout(title = "Parallel Coordinate Plots")

ct_olive = SharedData$new(olive_data, group = "olive")

get_buttons = function(df, axis){
  buttons = list()
  i = 1
  for(col in colnames(df)){
    buttons[[i]] = list(method = "restyle",
                        args = list(axis, list(olive_data[[col]])),
                        label = paste(axis , ": ", col, sep=""))
    i = i + 1
  }

  return(buttons)
}

buttons_x = get_buttons(olive_data[, 4:11], "x")
buttons_y = get_buttons(olive_data[, 4:11], "y")
buttons_z = get_buttons(olive_data[, 4:11], "z")

annot = list(list(text = "X", x=-0.6, y = 0.78, xref = 'paper', yref = 'paper', showarrow = FALSE)
,
             list(text = "Y", x=-0.6, y = 0.55, xref = 'paper', yref = 'paper', showarrow = FALSE)
,
             list(text = "Z", x=-0.6, y = 0.34, xref = 'paper', yref = 'paper', showarrow = FALSE)
)

scatter_plot_3d = plot_ly(ct_olive, x = ~palmitic, y = ~palmitoleic, z = ~stearic) %>%
  add_markers(marker = list(size=4), opacity = 0.5) %>%
  layout(scene = list(xaxis = list(title = "X"),
                      yaxis = list(title = "Y"),
                      zaxis = list(title = "Z")),
         title = "3D Scatterplot",
         # annotations = annot,
         updatemenus = list(
           list(y = 0.4, buttons = buttons_x, text = "X", active = 0),
           list(y = 0.6, buttons = buttons_y, text = "Y", active = 1),
           list(y = 0.8, buttons = buttons_z, text = "Z", active = 2)))

bar_chart = plot_ly(ct_olive, x = ~Region_category) %>%
  add_histogram() %>% layout(title = "Histogram: Region", barmode = "overlay")

bscols(widths = c(12, 12,6,6),
  tags$h2("Interactive plots to analyze relationships between variables"),
       par_coord_plot %>%
         highlight(on = "plotly_select", dynamic = T, persistent = T, opacityDim = I(1)) %>%
         hide_legend(),
       scatter_plot_3d %>%
         highlight(on = "plotly_click", dynamic = T, persistent = T) %>% hide_legend(),
       bar_chart %>%
         highlight(on = "plotly_click", dynamic = T, persistent = T) %>% hide_legend())
```

Interactive plots to analyze relationships between variables

Brush color

rgba(228



Brush color

rgba(228

3D Scatterplot

z: stearic

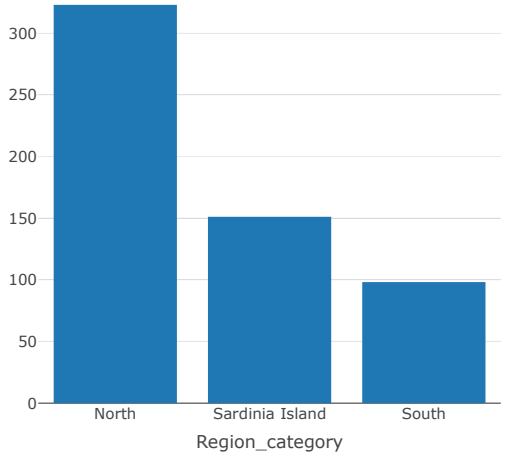
y: palmitoleic

x: palmitic

Brush color

rgba(228

Histogram: Region



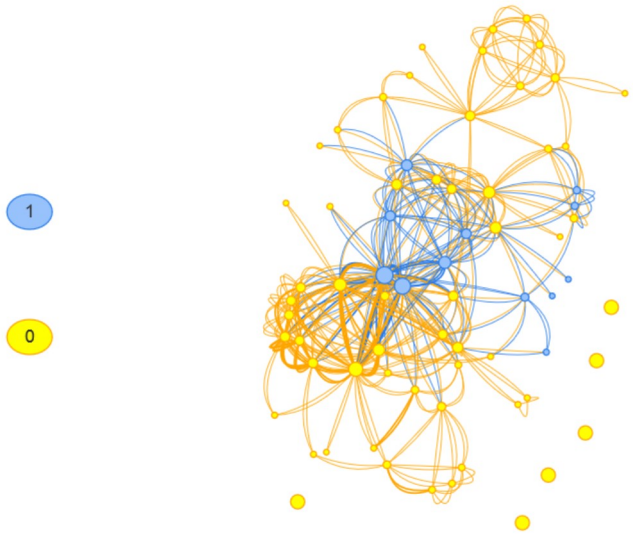
5 Network Graphs

5.1 Network Graph using visNetwork

5.1.1 Replusion Optimzed Network Graph

```
links <- read.table("trainData.dat", as.is=T)
nodes <- as.data.frame(read.table("trainMeta.dat", as.is=T))

colnames(links) <- c("from","to","strength")
links <- links[order(links$from, links$to),]
nodes$id <- 1:70
rownames(links) <- NULL

colnames(nodes)[2] <- "BombingGrp"

#Size of links based on "strength of links"
links$width <- links$strength*3

#Nodes colored based on Bombing Group
nodes$label <- nodes$V1
nodes$group <- nodes$BombingGrp

#Size of nodes proportional to the number of connections
graph <- graph.data.frame(links, directed = F)
strength_value <- strength(graph)
nodes$value <- strength_value[match(nodes$id, names(strength_value))]

visNetwork(nodes, links, main = "Network of people invloved in Madrid Bombing") %>%
  visLegend() %>%
  visLayout(randomSeed = 8) %>%
  visPhysics(solver="repulsion") %>%
  visOptions(highlightNearest = list(enabled = T, degree = 1, hover = T))
```

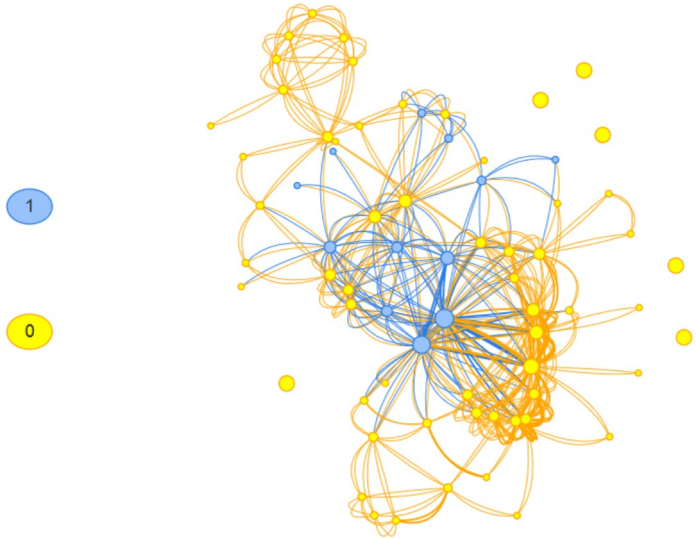
Network of people invloved in Madrid Bombing



5.1.2 Replusion Optimzed Network Graph with highlights

```
visNetwork(nodes, links, main = "Network of people invloved in Madrid Bombing") %>%
  visLegend() %>%
  visLayout(randomSeed = 12) %>%
  visPhysics(solver="repulsion") %>%
  visOptions(highlightNearest = list(enabled = T, degree = 2, hover = T))
```

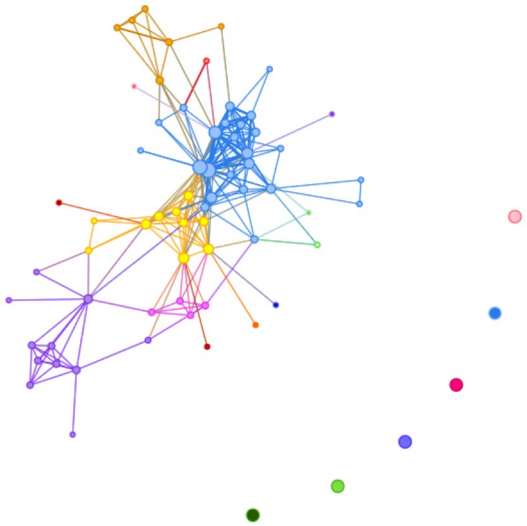
Network of people invloved in Madrid Bombing



5.1.3 Edge Between Optimizing Network Graph

```
for(i in 1:nrow(links)){
  links$from_name[i] <- nodes$V1[links$from[i]]
  links$to_name[i] <- nodes$V1[links$to[i]]
}

links <- links[,c("from_name","to_name","from","to","strength","width")]
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, main = "Network of people invloved in Madrid Bombing") %>%
  visIgraphLayout() %>%
  visOptions(highlightNearest = list(enabled = T, degree = 2, hover = T))
```

Network of people invloved in Madrid Bombing

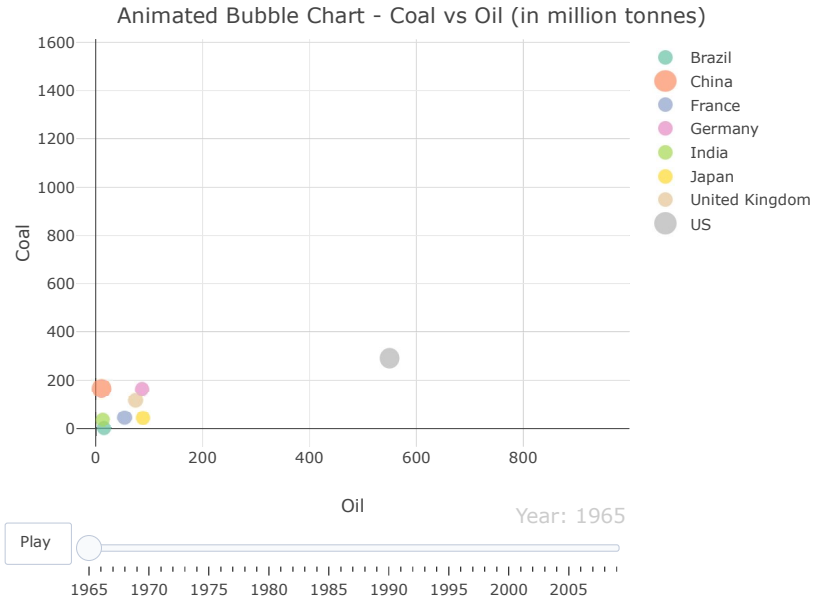


6 Animated Plots

6.1 Bubble Chart

6.1.1 Bubble Chart Animated with Cubic Easing

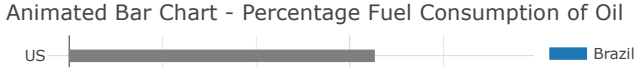
```
oc_data = read.csv2("Oilcoal.csv")

plot_ly(oc_data, x= ~Oil, y= ~Coal, frame = ~Year, color = ~Country) %>%
  add_markers(size = ~Marker.size, marker = list(sizemin = 5)) %>%
  animation_opts(500, easing = "cubic", redraw = F) %>%
  layout(title = "Animated Bubble Chart - Coal vs Oil (in million tonnes)")
```



6.2 Bar Chart

6.2.1 Bar Chart Animated with Elastic Easing

```
oc_data$oil_prop = 100* oc_data$Oil / (oc_data$Oil + oc_data$Coal)
oc_data_zero = oc_data
oc_data_zero$oil_prop = 0
oc_line_anim_data = rbind(oc_data, oc_data_zero)

plot_ly(oc_line_anim_data, x = ~oil_prop, y = ~Country, frame = ~Year) %>%
  add_lines(split = ~Country, line = list(width = 10)) %>%
  animation_opts(500, easing = "elastic", redraw = F) %>%
  layout(title = "Animated Bar Chart - Percentage Fuel Consumption of Oil")
```

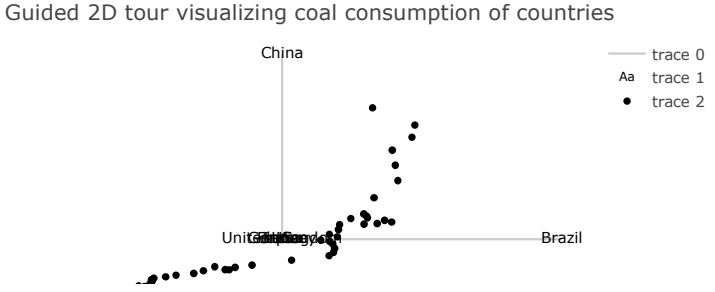


7 Tour and Projection

7.1 Animated with grand tour

```
oc_tour_data = cast(oc_data[,1:3], Year ~ Country, value = "Coal")

mat <- rescale(oc_tour_data)
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")

# step = 7.67
```

```
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, title = "Guided 2D tour visualizing coal consumption of countries
")
```

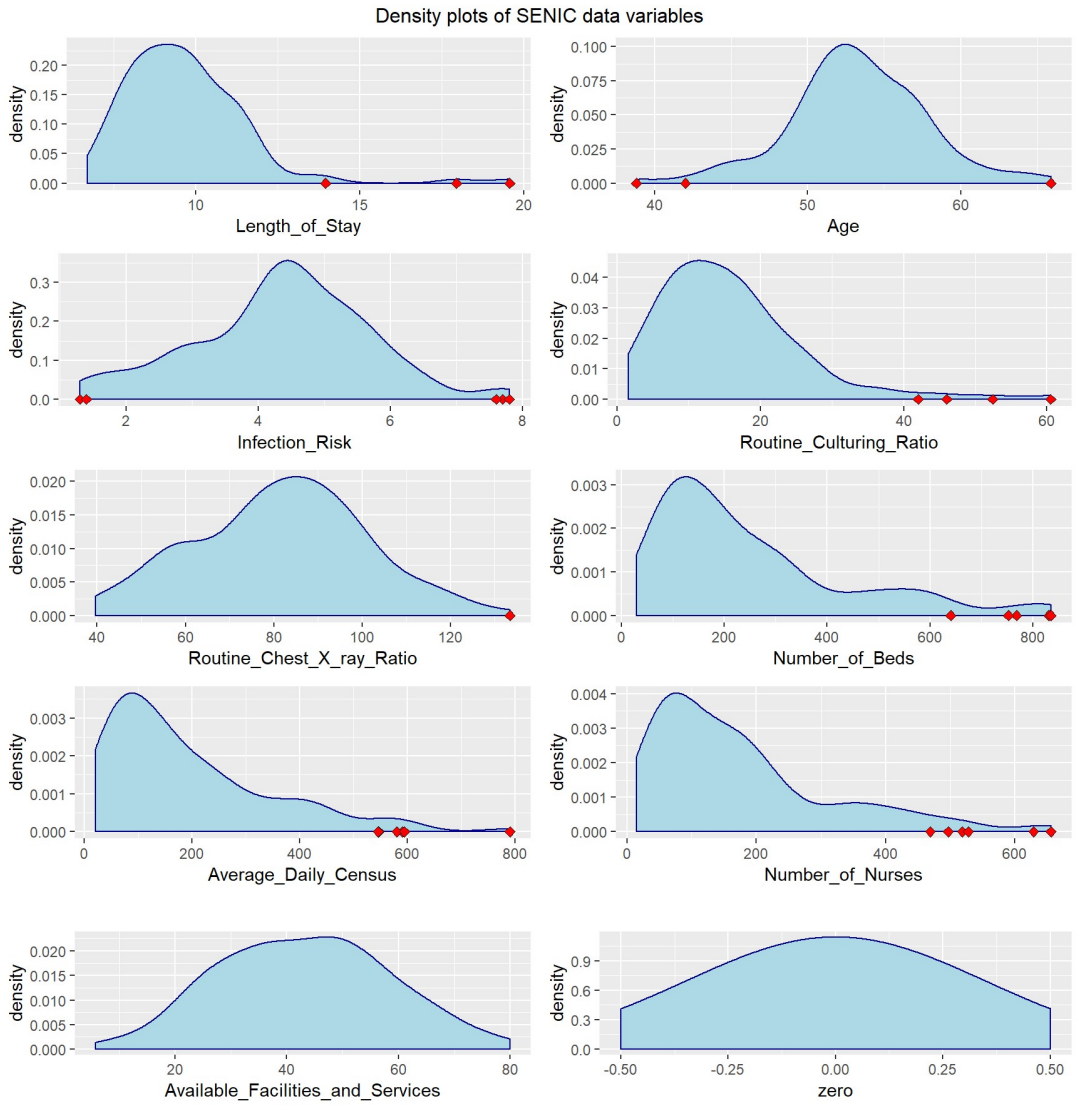


8 Multiple Plots

8.1 Density Plots

8.1.1 Density Plot with Outlier Highlight

```
plot_density_with_outliers <- function(var_data, col_name){
  p <- NULL
  df_data = setNames(data.frame(var_data),col_name)
  if(length(get_outliers(df_data[[col_name]])) > 0){
    p <- ggplot(df_data) +
      geom_density(aes_string(x=col_name), fill = "lightblue", color = "darkblue") +
      geom_point(data=df_data[get_outliers(df_data[[col_name]]),,drop=FALSE],
                 aes_string(x=col_name), y=0, shape=23, size=2, colour="black", fill="red")
  }
  else{
    p <- ggplot(df_data) +
      ggtitle(paste("")) +
      theme(plot.title = element_text(hjust = 0.5)) +
      geom_density(aes_string(x=col_name), fill = "lightblue", color = "darkblue")
  }

  return(p)
}

categorical_columns = c("Medical_School_Affiliation", "Region")
ID_columns = c("Identification_Number")
quantitative_columns = setdiff(colnames(senic_data), c(categorical_columns, ID_columns))

plot_list = mapply(plot_density_with_outliers, senic_data[, quantitative_columns],
                   colnames(senic_data[, quantitative_columns]), SIMPLIFY = FALSE)
plot_matrix <- arrangeGrob(grobs = plot_list, ncol = 2)
grid.arrange(plot_matrix, respect=TRUE, top="Density plots of SENIC data variables")
```


Density plots of SENIC data variables
density
Length_of_Stay
Age
Infection_Risk
Routine_Culturing_Ratio
Routine_Chest_X_ray_Ratio
Number_of_Beds
Average_Daily_Census
Number_of_Nurses
Available_Facilities_and_Services
zero

9 Shiny

```
#UI component
ui <- fluidPage(
  sliderInput(inputId="bw_value", label="Choose bandwidth size", value=4.5, min=0.1, max=80),
  checkboxGroupInput("selected_variables", "Variables to show: ", quantitative_columns, inline=TRU
E),
  plotOutput("densPlot", height = "650px")
)

plot_density_with_outliers_shiny <- function(df_data, col_name, bw){
  p <- NULL
  if(length(get_outliers(senic_data[[col_name]])) > 0){
    p <- ggplot(df_data) +
      ggtitle(paste("Density plot of ", col_name)) +
      theme(plot.title = element_text(hjust = 0.5)) +
      geom_density(aes_string(x=col_name), fill = "lightblue", color = "darkblue", bw=bw) +
      geom_point(data=df_data[get_outliers(df_data[[col_name]]),],
                 aes_string(x=col_name, y=0), shape=23, size=2, colour="black", fill="red")
  }
  else{
    p <- ggplot(df_data) +
      ggtitle(paste("Density plot of ", col_name)) +
      theme(plot.title = element_text(hjust = 0.5)) +
      geom_density(aes_string(x=col_name), fill = "lightblue", color = "darkblue", bw=bw)
  }

  return(p)
}


server <- function(input, output) {

  output$densPlot <- renderPlot({

    selected_columns = input$selected_variables
    plot_list = vector("list", length(selected_columns))

    if(length(selected_columns) > 0){
      for(i in 1:length(selected_columns)){
        plot_list[[i]] = plot_density_with_outliers_shiny(senic_data, selected_columns[i],
                                                          bw = input$bw_value)
      }
      plot_matrix <- arrangeGrob(grobs = plot_list, ncol = 2)
      grid.arrange(plot_matrix)
    }

  })
}

shinyApp(ui = ui, server = server, options = list(width="800px", height="900px"))
```

10 Appendix Code

11

{r, ref.label=knitr::all_labels(), echo=TRUE, eval=FALS