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)
library(data.table)
library(reshape2)
```

```
library(tibble)

Sys.setenv('MAPBOX_TOKEN' = 'pk.eyJ1IjoibGFrc2hpZGFhIiwiYSI6ImNqbWIyOHN2NTRlZ3kzam10aTljeGNybWgifQ.8EG9
```

Reading Data

Data Mugging

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)))))
}
```

Scaling the Data

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

Distance Matrix between rows

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

Non-metric MDS

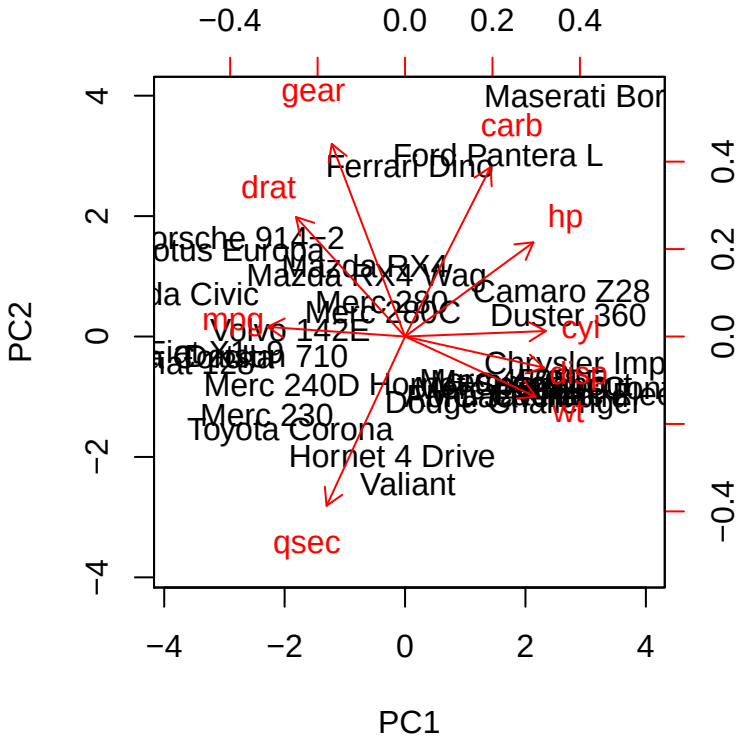
```
mds_result <- isoMDS(distance_matrix, k=2, p=2)
```

```
## 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
```

Principle Component Analysis

```
mtcars.pca <- prcomp(mtcars[,c(1:7,10,11)], center = TRUE,scale. = TRUE)
biplot(mtcars.pca, scale = 0)
```



Types of Projection

```
#projection = list(type = "mercator"))
#projection = list(type = "albers usa"))
#projection = list(type = "equirectangular"))
#projection = list(type = "conic equal area"))
#projection = list(type = "azimuthal equal area"))
#projection = list(type = "equirectangular"))
#projection = list(type = "orthographic"))
```

Types of easing

```
#animation_opts(500, easing = 'linear', redraw = F)
# animation_opts(500, easing = 'quad', redraw = F)
# animation_opts(500, easing = 'cubic', redraw = F)
# animation_opts(500, easing = 'sin', redraw = F)
# animation_opts(500, easing = 'exp', redraw = F)
```

```
# animation_opts(500, easing = 'circle', redraw = F)
# animation_opts(500, easing = 'elastic', redraw = F)
# animation_opts(500, easing = 'back', redraw = F)
# animation_opts(500, easing = 'bounce', redraw = F)
# animation_opts(500, easing = 'linear-in', redraw = F)
# animation_opts(500, easing = 'quad-in', redraw = F)
# animation_opts(500, easing = 'cubic-in', redraw = F)
# animation_opts(500, easing = 'sin-in', redraw = F)
# animation_opts(500, easing = 'exp-in', redraw = F)
# animation_opts(500, easing = 'circle-in', redraw = F)
# animation_opts(500, easing = 'elastic-in', redraw = F)
# animation_opts(500, easing = 'back-in', redraw = F)
# animation_opts(500, easing = 'bounce-in', redraw = F)
# animation_opts(500, easing = 'linear-out', redraw = F)
# animation_opts(500, easing = 'quad-out', redraw = F)
# animation_opts(500, easing = 'cubic-out', redraw = F)
# animation_opts(500, easing = 'sin-out', redraw = F)
# animation_opts(500, easing = 'exp-out', redraw = F)
# animation_opts(500, easing = 'circle-out', redraw = F)
# animation_opts(500, easing = 'elastic-out', redraw = F)
# animation_opts(500, easing = 'back-out', redraw = F)
# animation_opts(500, easing = 'bounce-out', redraw = F)
# animation_opts(500, easing = 'linear-in-out', redraw = F)
# animation_opts(500, easing = 'quad-in-out', redraw = F)
# animation_opts(500, easing = 'cubic-in-out', redraw = F)
# animation_opts(500, easing = 'sin-in-out', redraw = F)
# animation_opts(500, easing = 'exp-in-out', redraw = F)
# animation_opts(500, easing = 'circle-in-out', redraw = F)
# animation_opts(500, easing = 'elastic-in-out', redraw = F)
# animation_opts(500, easing = 'back-in-out', redraw = F)
# animation_opts(500, easing = 'bounce-in-out', redraw = F)
```

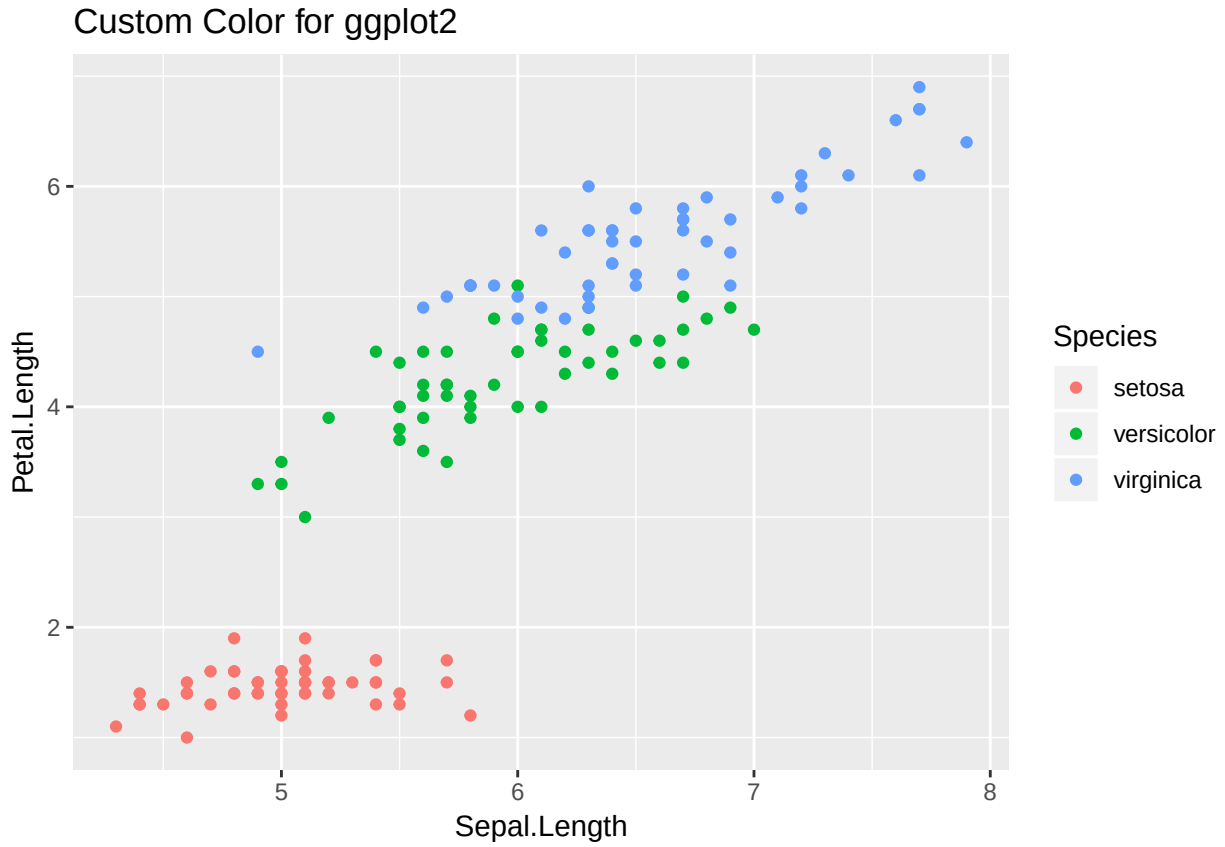
Sorting dataset

```
mtcars <- as.data.table(mtcars)
mtcars <- mtcars[order(-mpg, cyl)]

head(mtcars, 10)
```

```
##       mpg cyl  disp  hp drat    wt  qsec vs am gear carb
##  1: 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
##  2: 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
##  3: 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
##  4: 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
##  5: 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
##  6: 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
##  7: 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
##  8: 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
##  9: 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 10: 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
```

Colour selection palette

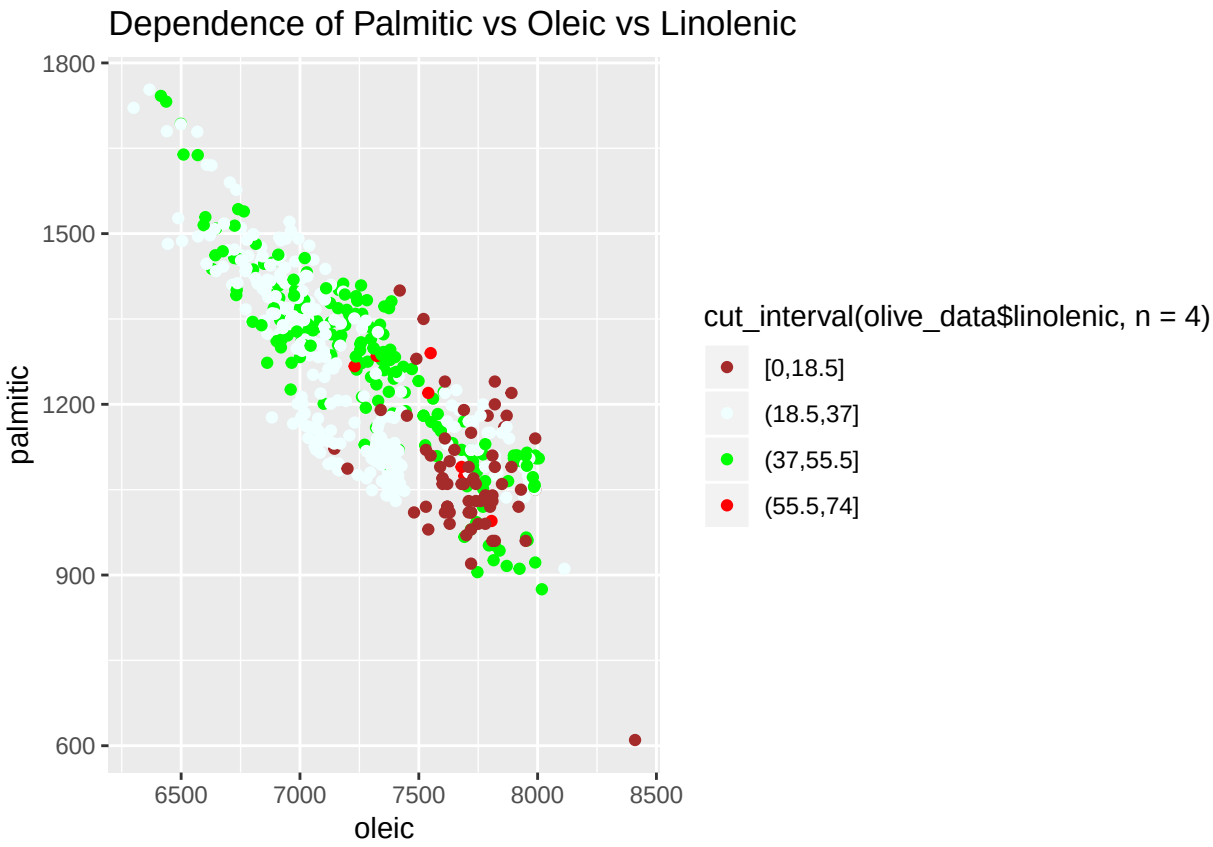
```
# The palette with grey:
default_colors <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79
# ggplot2
ggplot(data=iris, aes(x= Sepal.Length, y = Petal.Length, color = Species)) + geom_point() + scale_fill_
```



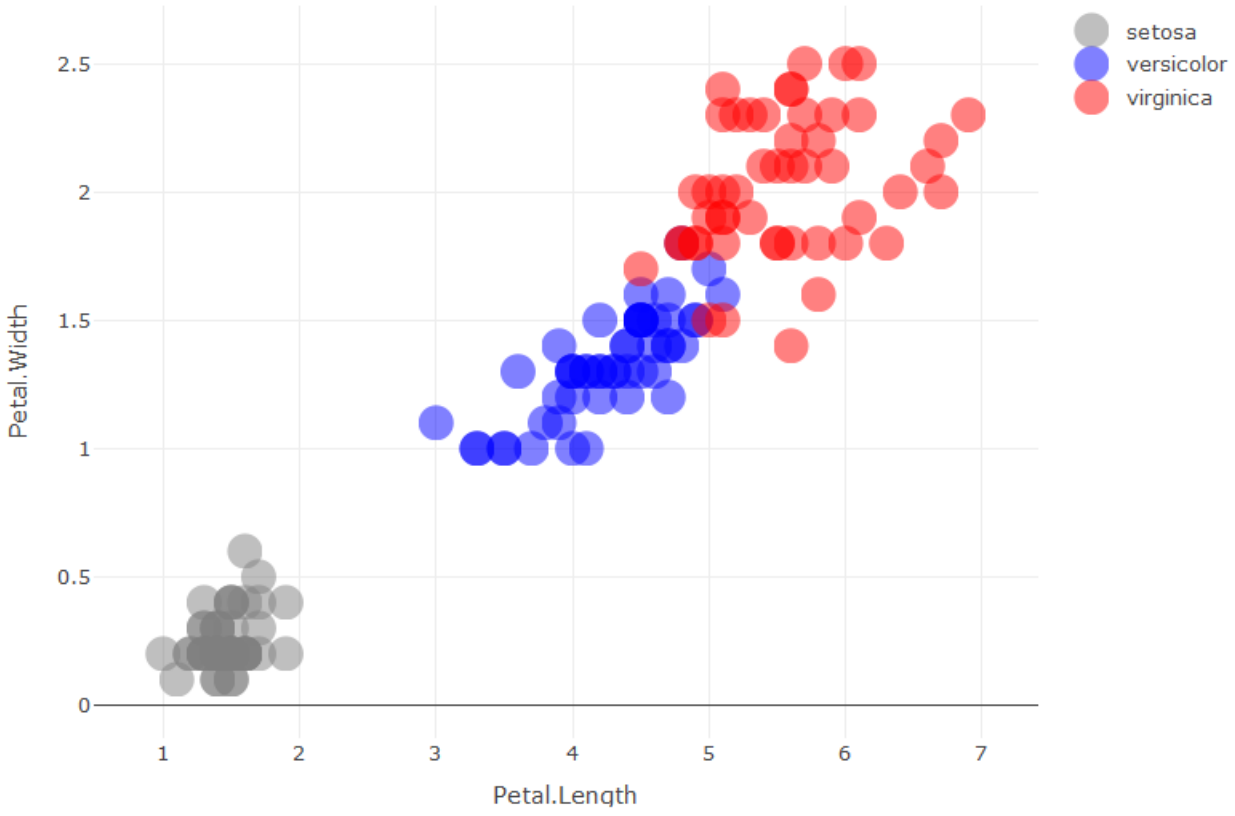

```
# plotly
x <- plot_ly(data = iris, x = ~Sepal.Length, y = ~Petal.Length, color = ~Species, colors = default_colo
```

Adding custom colours without palette

```
# ggplot2
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") +
  scale_colour_manual(values=c("brown", "azure", "green", "red"))
```




```
# plotly
x <- plot_ly(iris, x = ~Petal.Length, y = ~Petal.Width,
        type="scatter", mode = "markers" , color = ~Species,
        colors = c("grey50", "blue", "red"), marker=list( size=20 , opacity=0.5))

knitr::include_graphics('./plotly_custom_color.png')
```



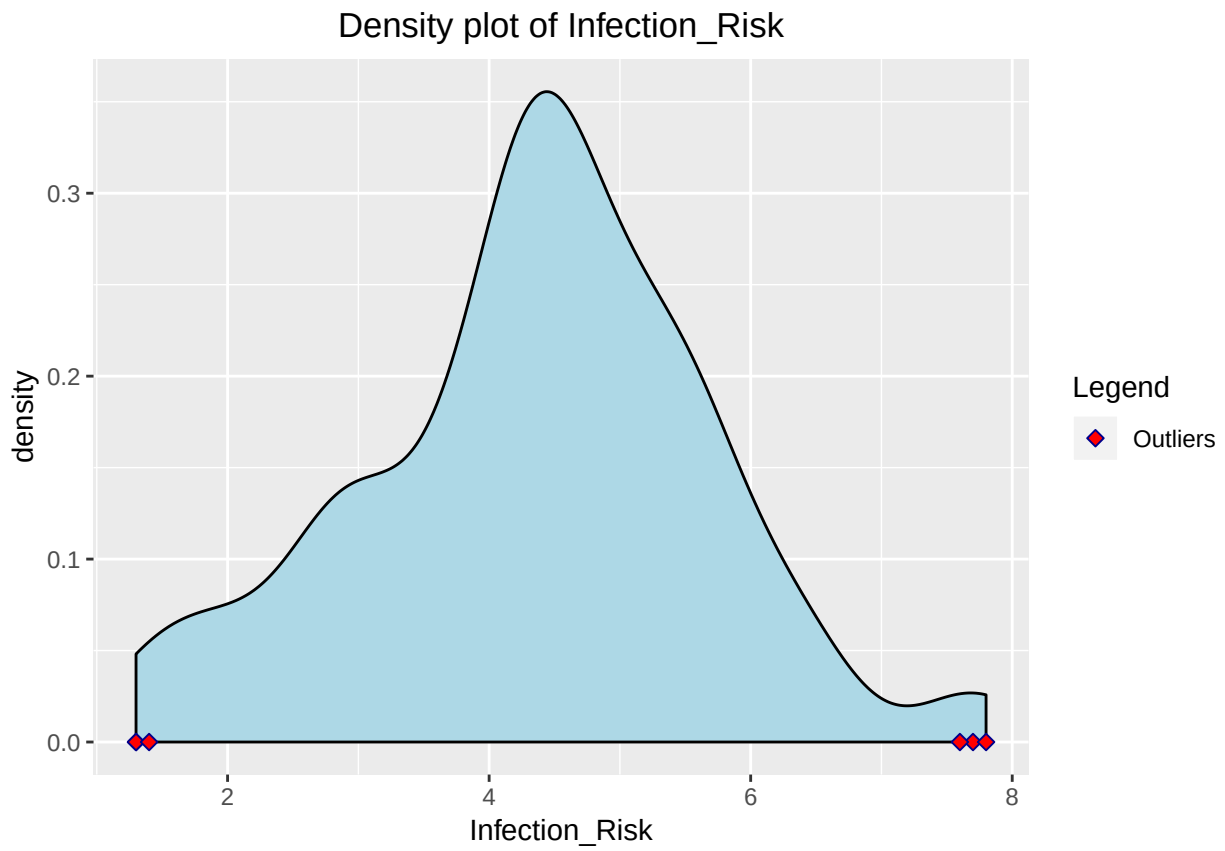
‘‘‘

Single Plots

Density Plot

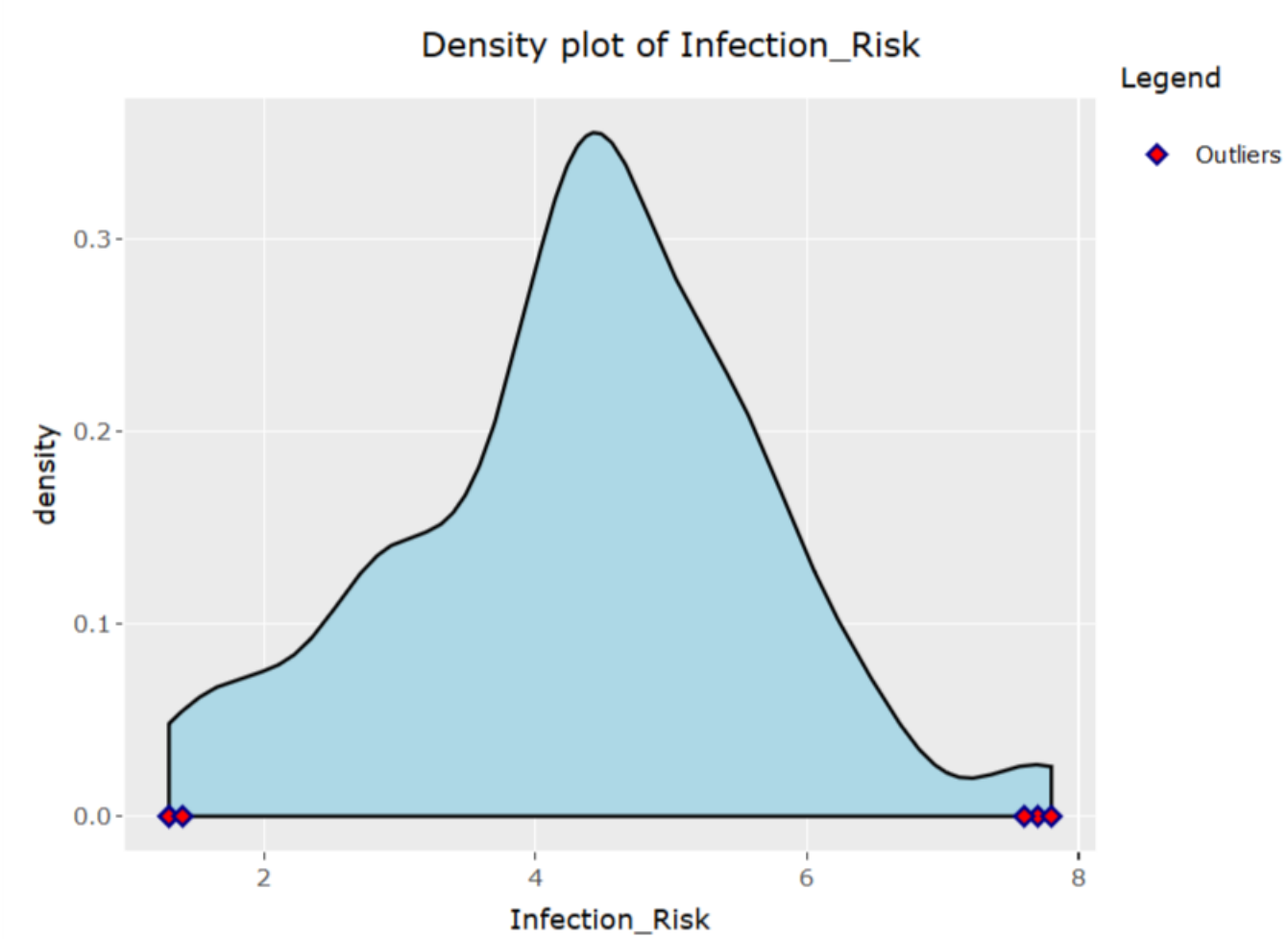
Density Plot with Outlier Highlight using GGplot2

```
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
```



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

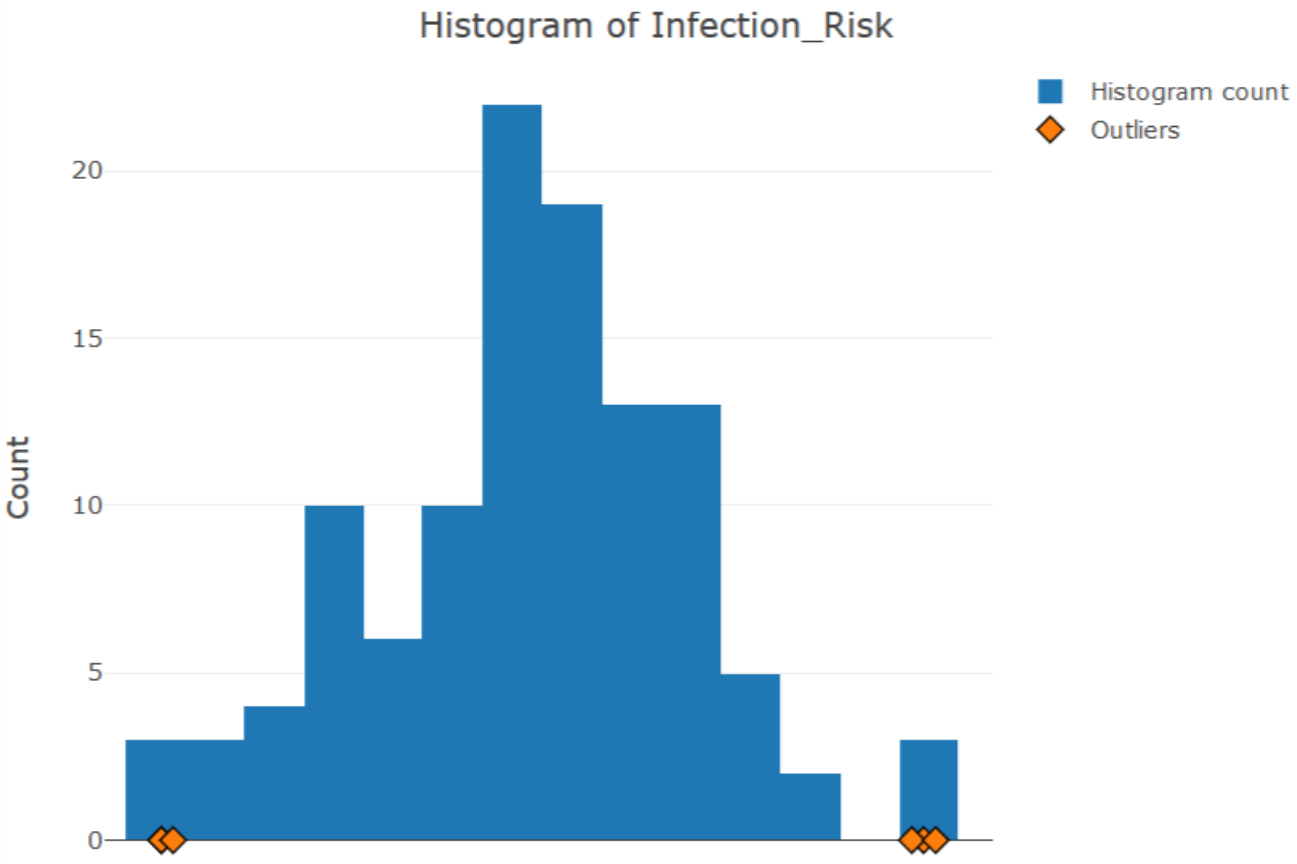
```
x <- ggplotly(p=density_plot_infection_risk)
knitr::include_graphics('./3.1.2.png')
```



Histogram Plot

Histogram plot with Outlier Highlight using Plotly

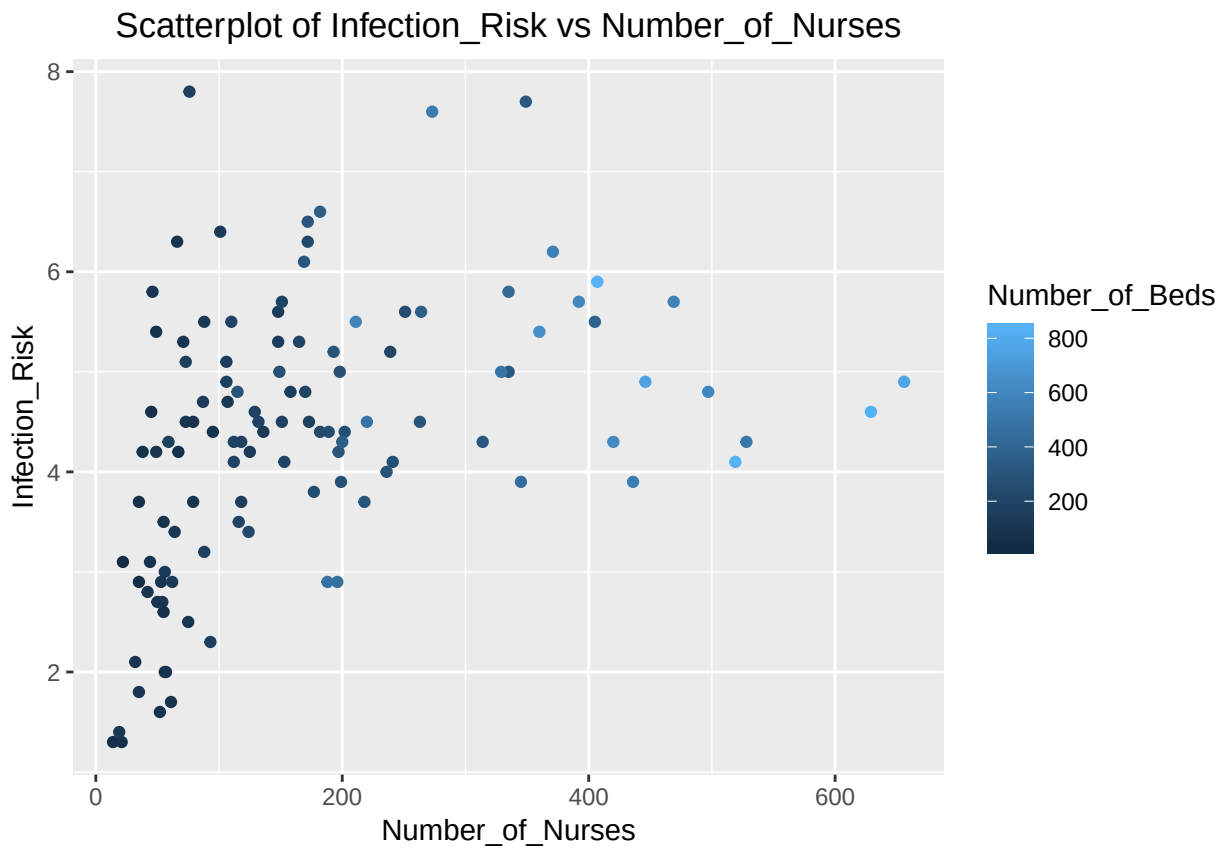
```
outliers = senic_data[get_outliers(senic_data$Infection_Risk),c("Infection_Risk")]
senic_data$zero = 0

x <- 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"))

knitr::include_graphics('./3.2.1.png')
```



Scatter Plot

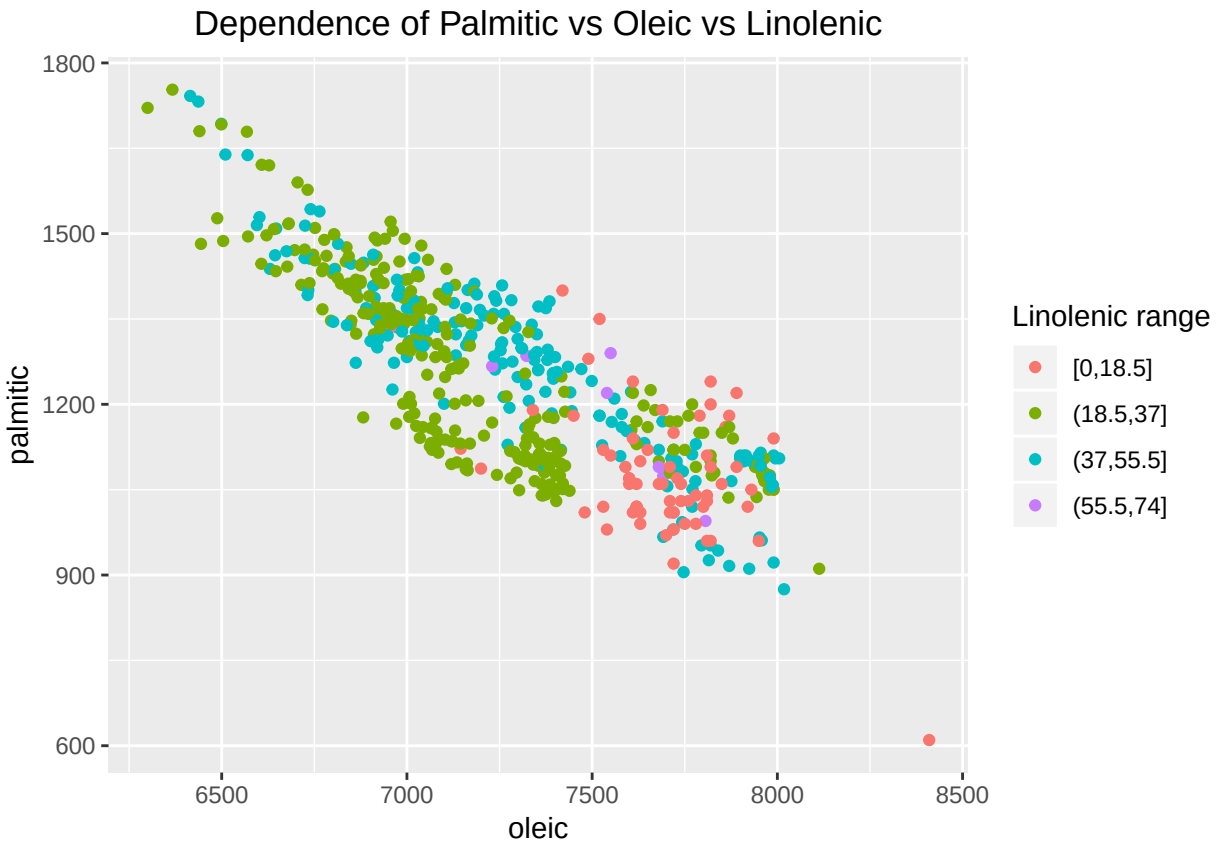
Simple scatter plot with colour

```
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))
```



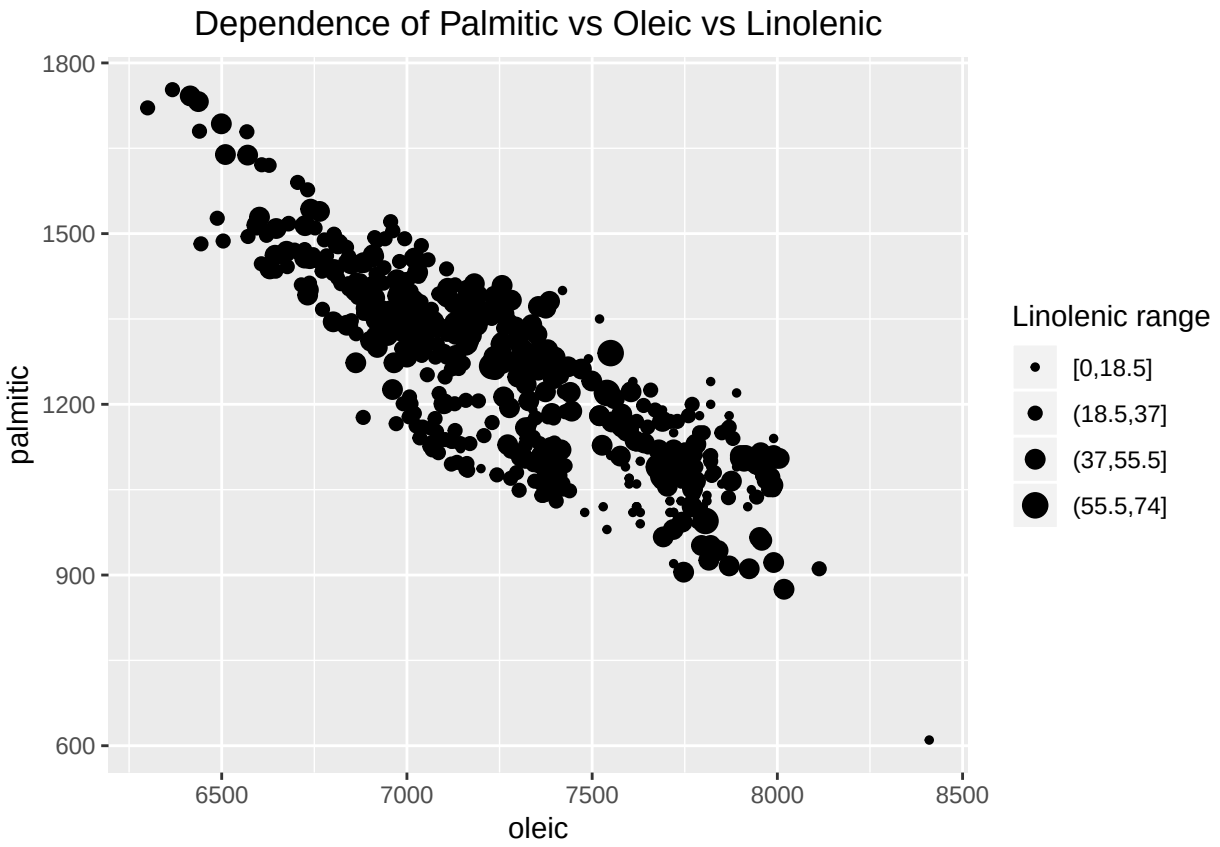
Scatter Plot with Discreetization (split a variable into classes)

```
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')
```



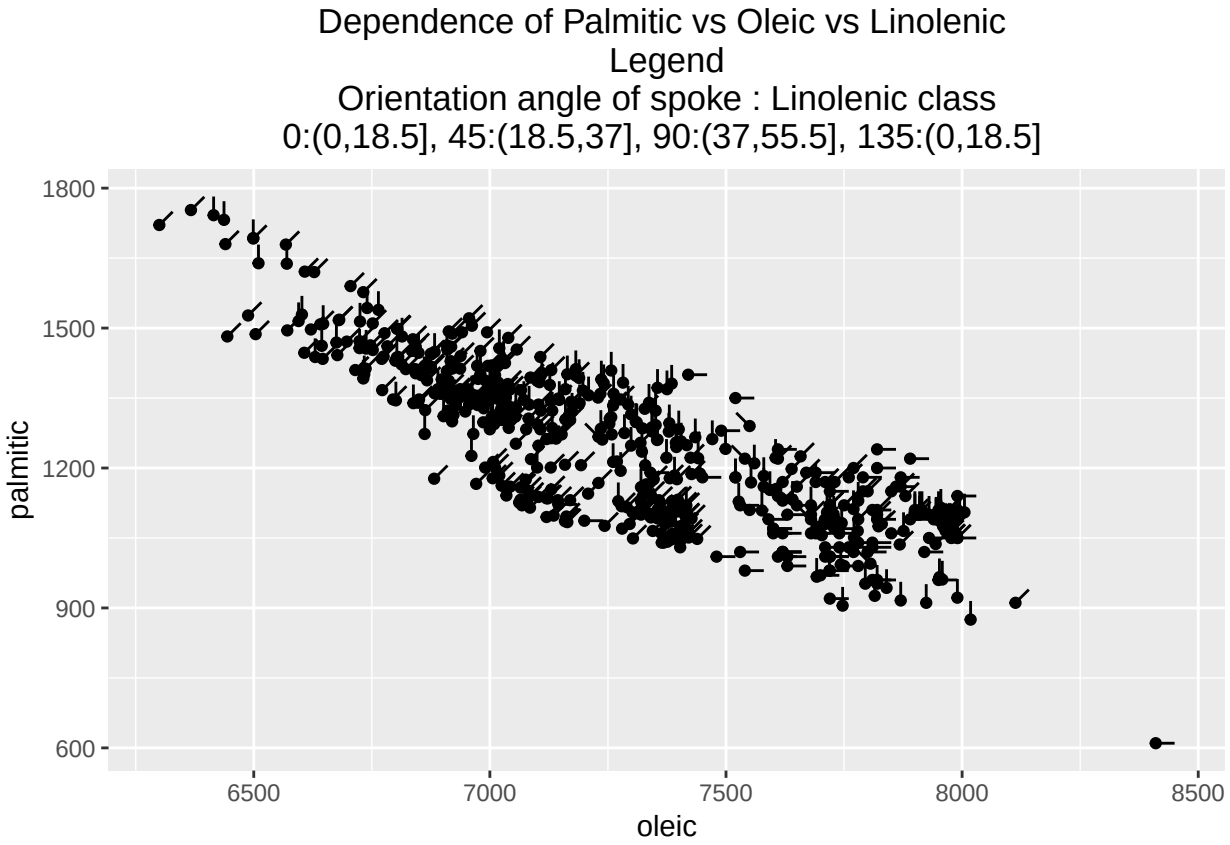
Scatter plot size varied

```
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))
```



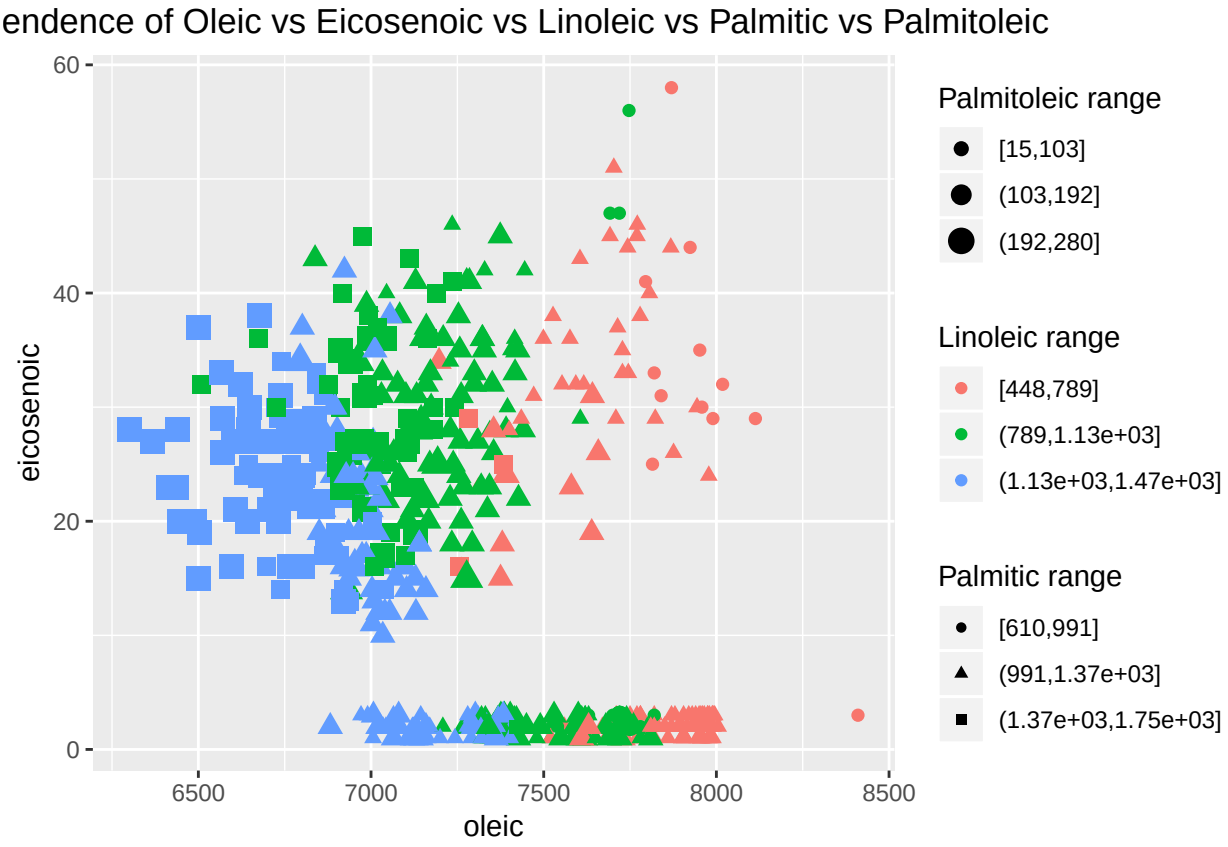
Scatter plot angle varied

```
# 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))
```



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))
```



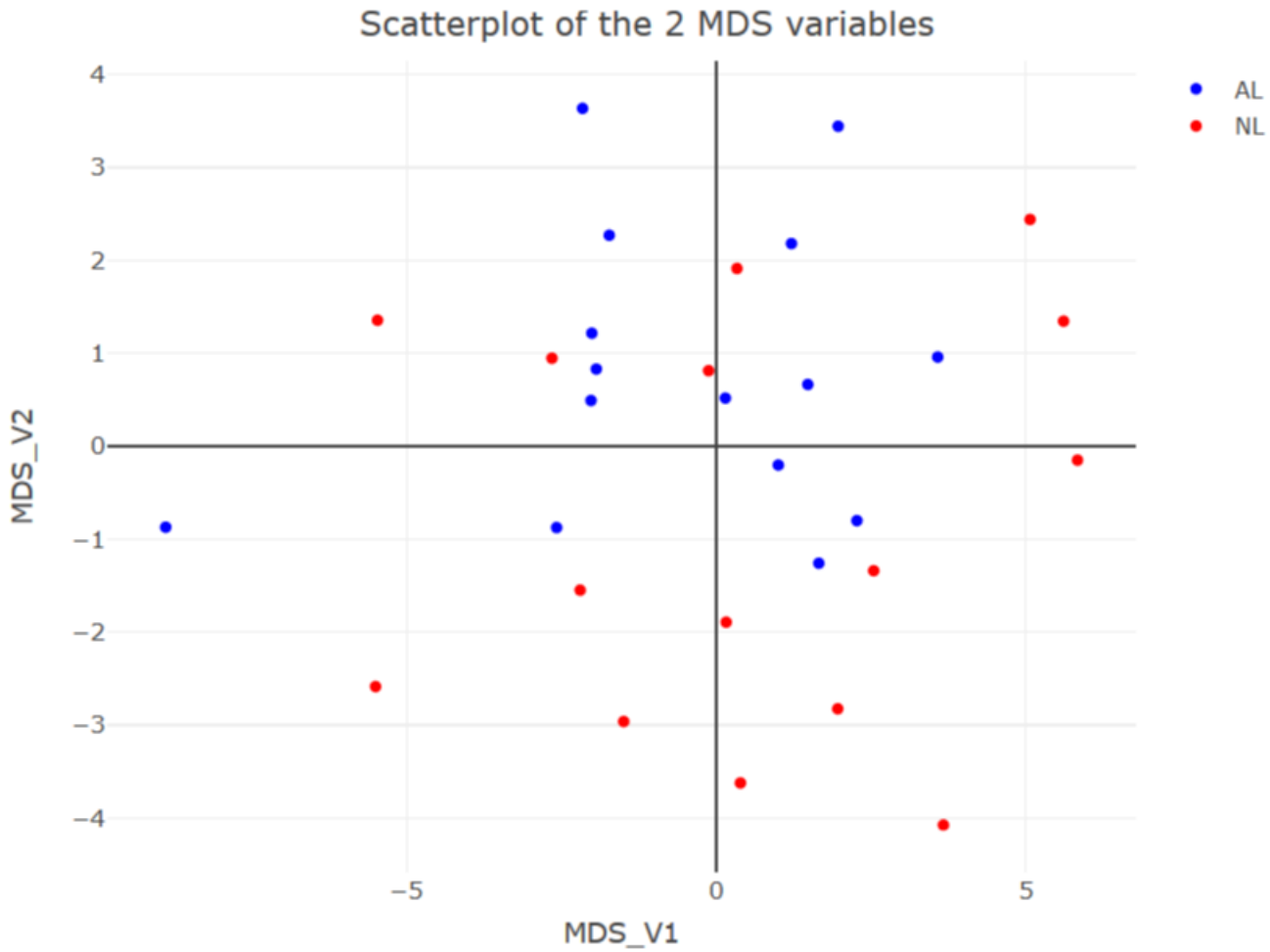
Scatter Plot of MDS

```
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



x <- 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")

knitr::include_graphics('./3.3.6.png')
```



Shepard Plot

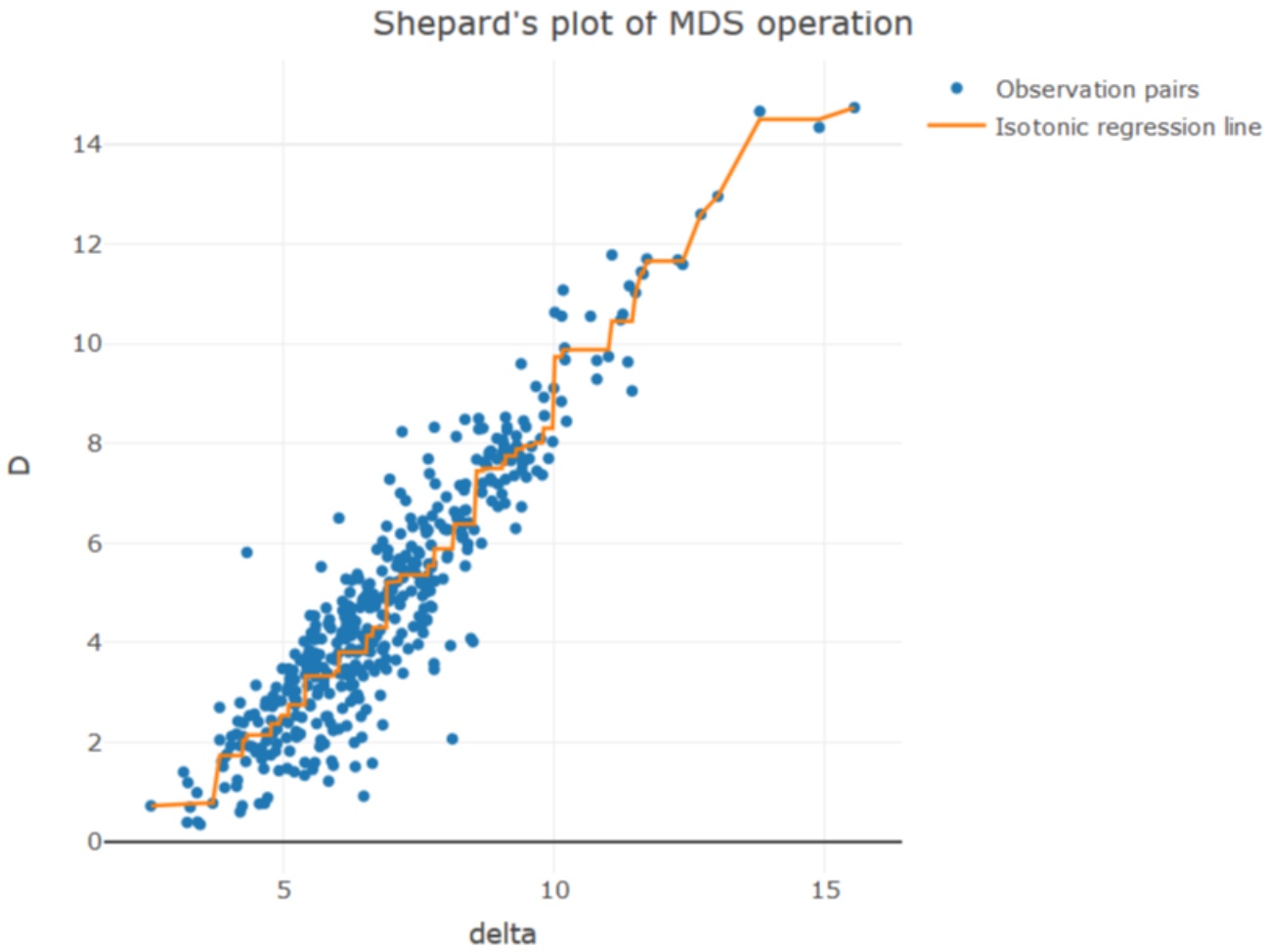
Shepard plot shows the fit of MDS, the distance in original dataset vs. distance in reordered dataset should be similar

```
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
```

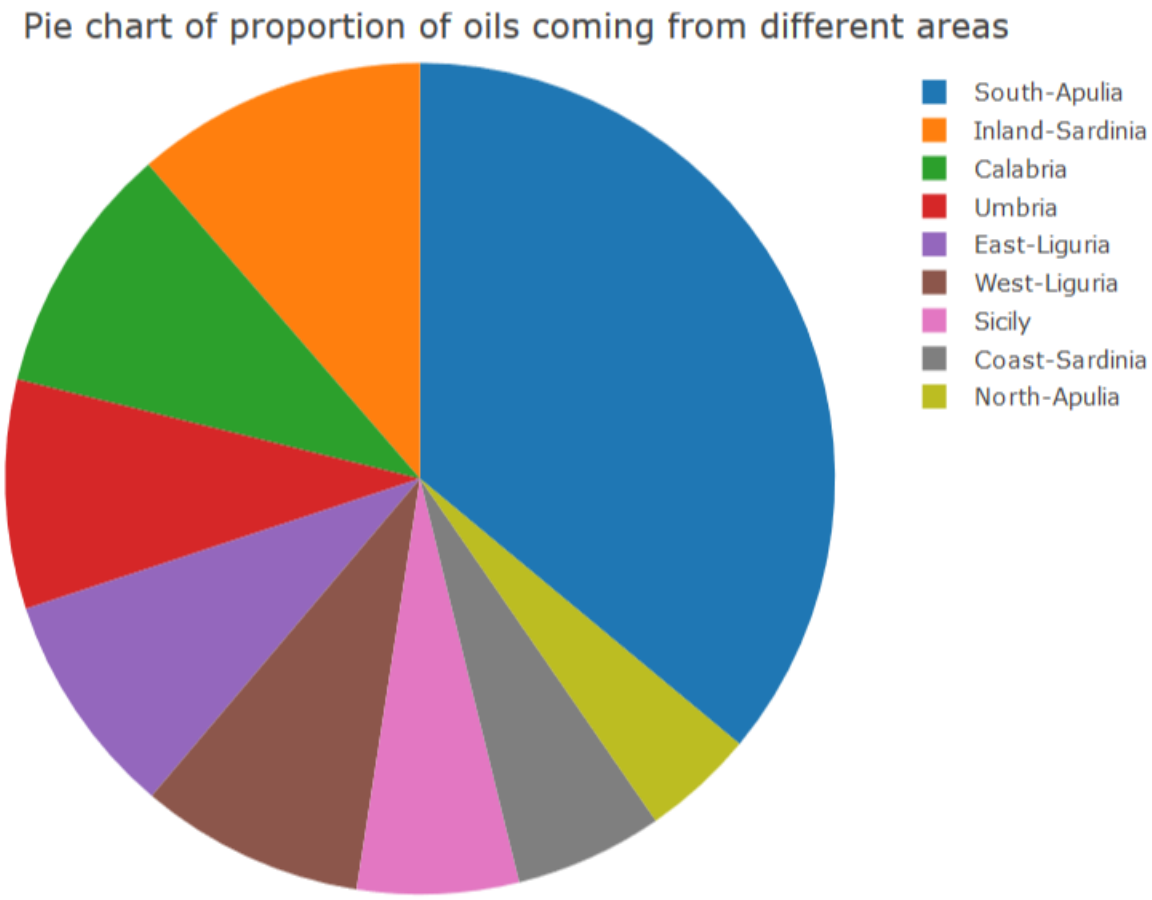
```
coords <- mds_result$points

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)])


x <- 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")

knitr::include_graphics('./3.4.1.png')
```



Pie Charts

Simple pie chart

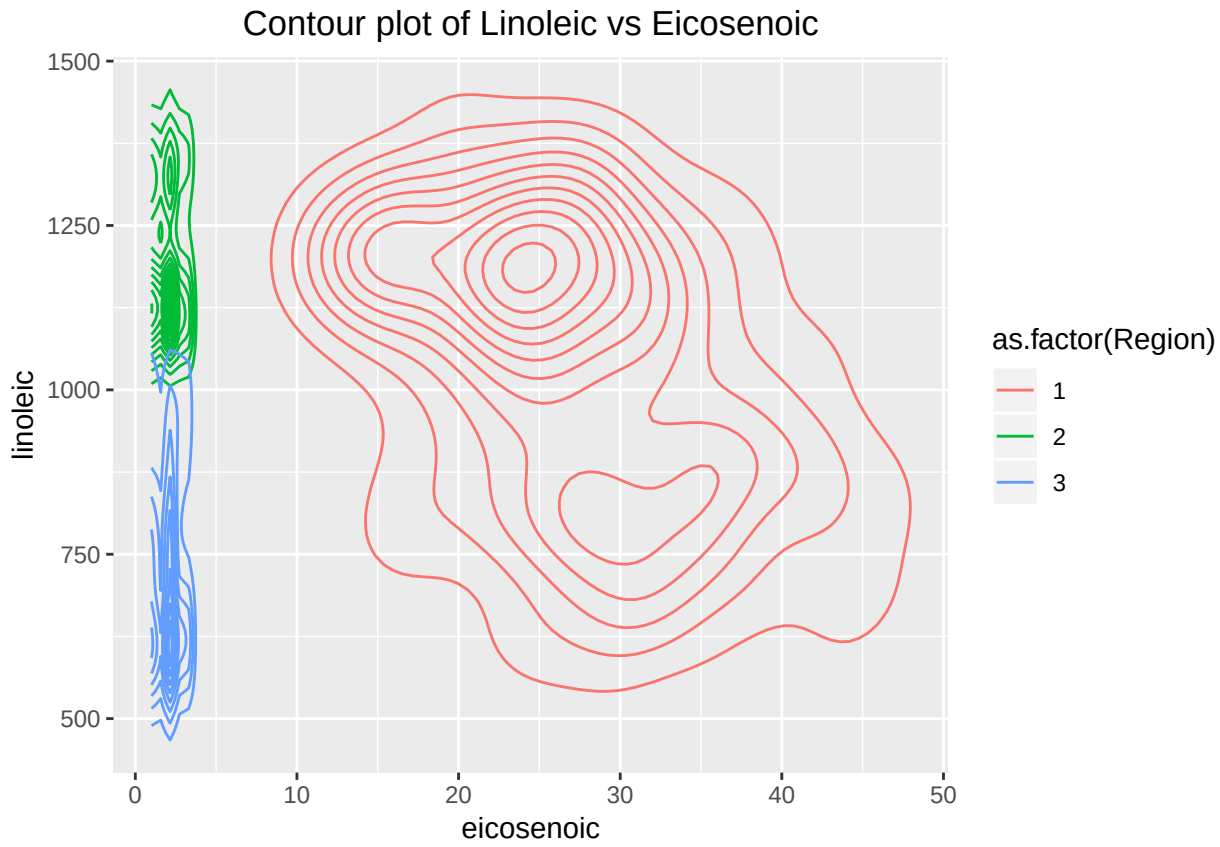
```
x <- plot_ly(olive_data,labels=~Area,type='pie',textinfo = "none") %>%
  layout(title = "Pie chart of proportion of oils coming from different areas")

knitr::include_graphics('./3.5.1.png')
```



2D density contour plot

Simple 2D plot using ggplot2

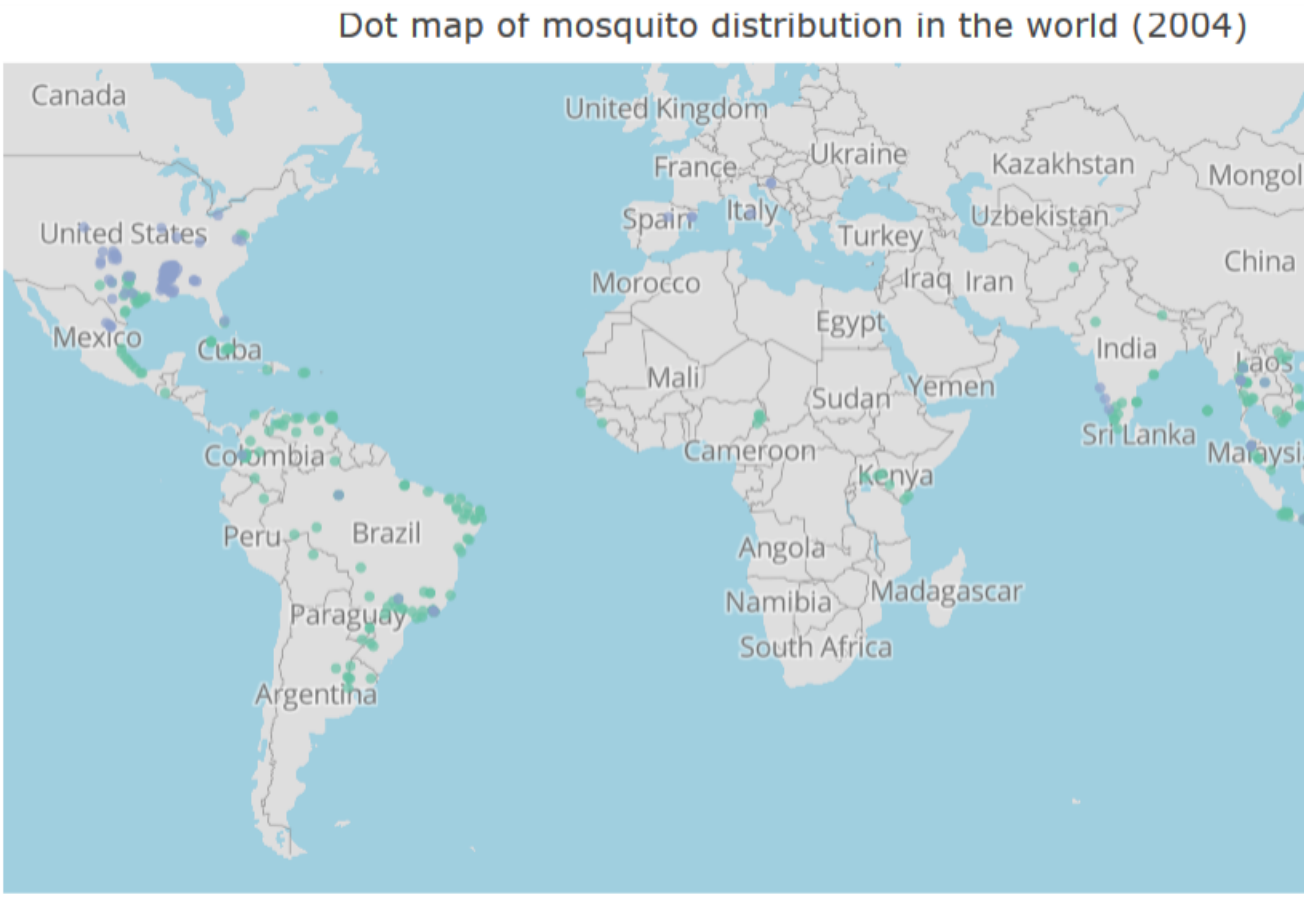
```
ggplot(olive_data)+geom_density_2d(aes(x=eicosenoic, y=linoleic, colour=as.factor(Region))) +
  ggtitle("Contour plot of Linoleic vs Eicosenoic") +
  theme(plot.title = element_text(hjust = 0.5))
```



Dot Map Plots (World Map)

Dot Map (Map with scatter plots)

```
x <- 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)")

knitr::include_graphics('./3.7.1.png')
```



Draw line between two places

Plotly between two places

```
x <- plot_geo(lat = c(40.7127, 51.5072), lon = c(-74.0059, 0.1275)) %>%
  add_lines(color = I("blue"), size = I(2)) %>%
  layout(
    title = 'London to NYC Great Circle',
    showlegend = FALSE,
    geo = list(
      resolution = 50,
      showland = TRUE,
      showlakes = TRUE,
      landcolor = toRGB("grey80"),
      countrycolor = toRGB("grey80"),
      lakecolor = toRGB("white"),
      projection = list(type = "equirectangular"),
      coastlinewidth = 2,
      lataxis = list(
        range = c(20, 60),
        showgrid = TRUE,
```

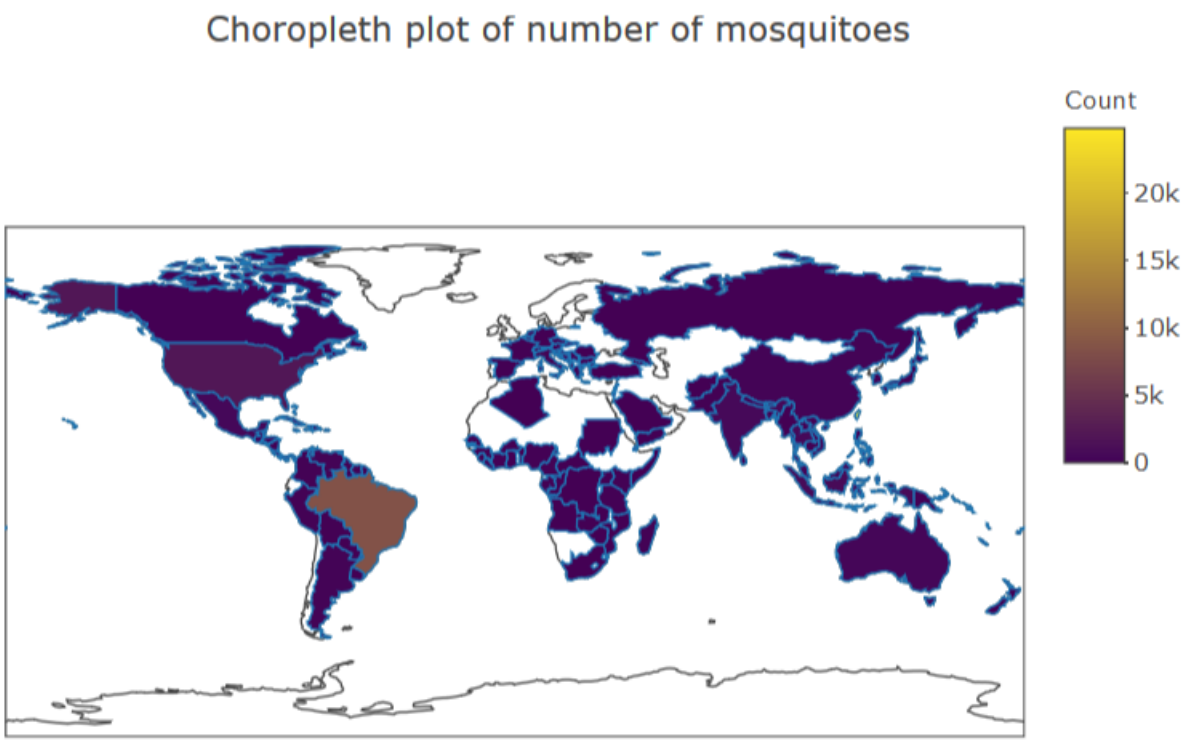
```
        tickmode = "linear",
        dtick = 10
      ),
      lonaxis = list(
        range = c(-100, 20),
        showgrid = TRUE,
        tickmode = "linear",
        dtick = 20
      )
    )
  )

knitr::include_graphics('./world_map.png')
```

Choropleth Map

Chorpleth Map with Equirectangular Projection

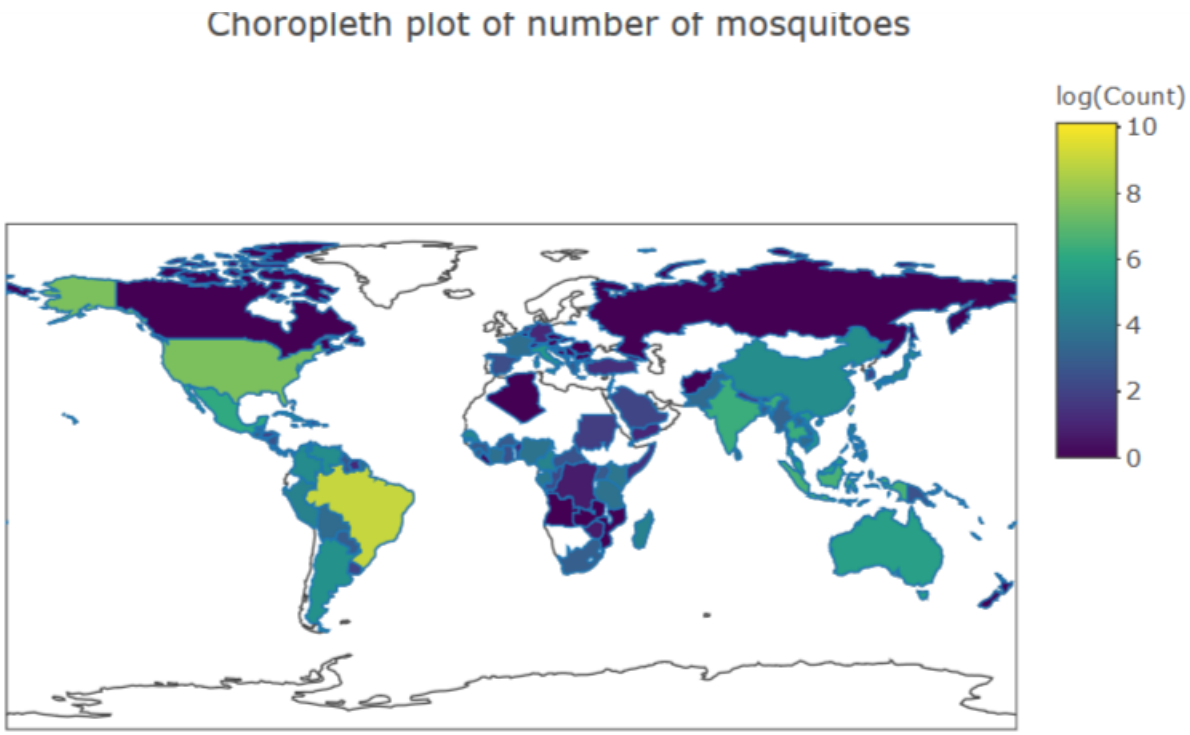
```
# Data aggregation
country_aggregate = aggregate(aegypti_data[,c("COUNTRY", "COUNTRY_ID")],
                              by = list(aegypti_data$COUNTRY, aegypti_data$COUNTRY_ID), FUN=length)
country_aggregate$COUNTRY = NULL
colnames(country_aggregate) = c("COUNTRY", "COUNTRY_ID", "Count")
```

```
x <- 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")))

knitr::include_graphics('./3.8.1.png')
```



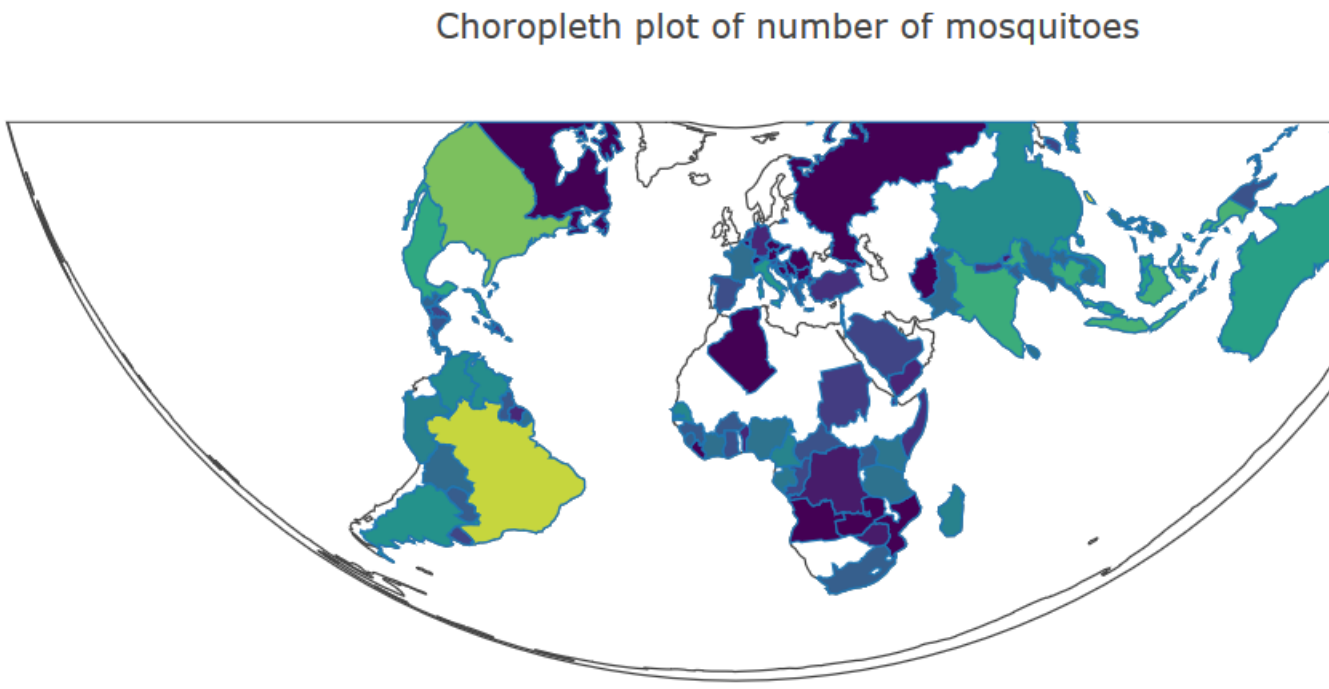
Choropleth plot with Equirectangular Projection and log

```
x <- 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")))

knitr::include_graphics('./3.8.2.png')
```



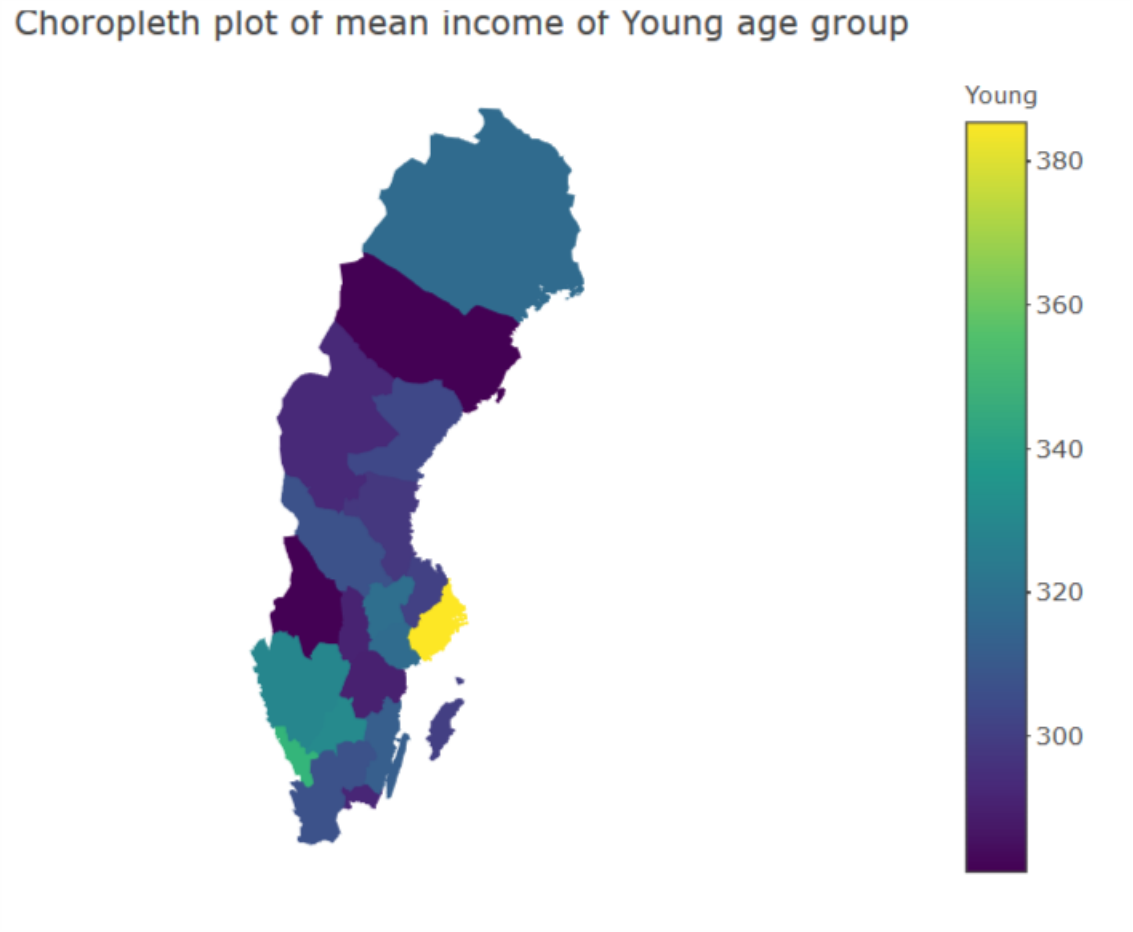
Choropleth plot with Conic Area Projection and log

```
x <- 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")))

knitr::include_graphics('./3.8.3.png')
```



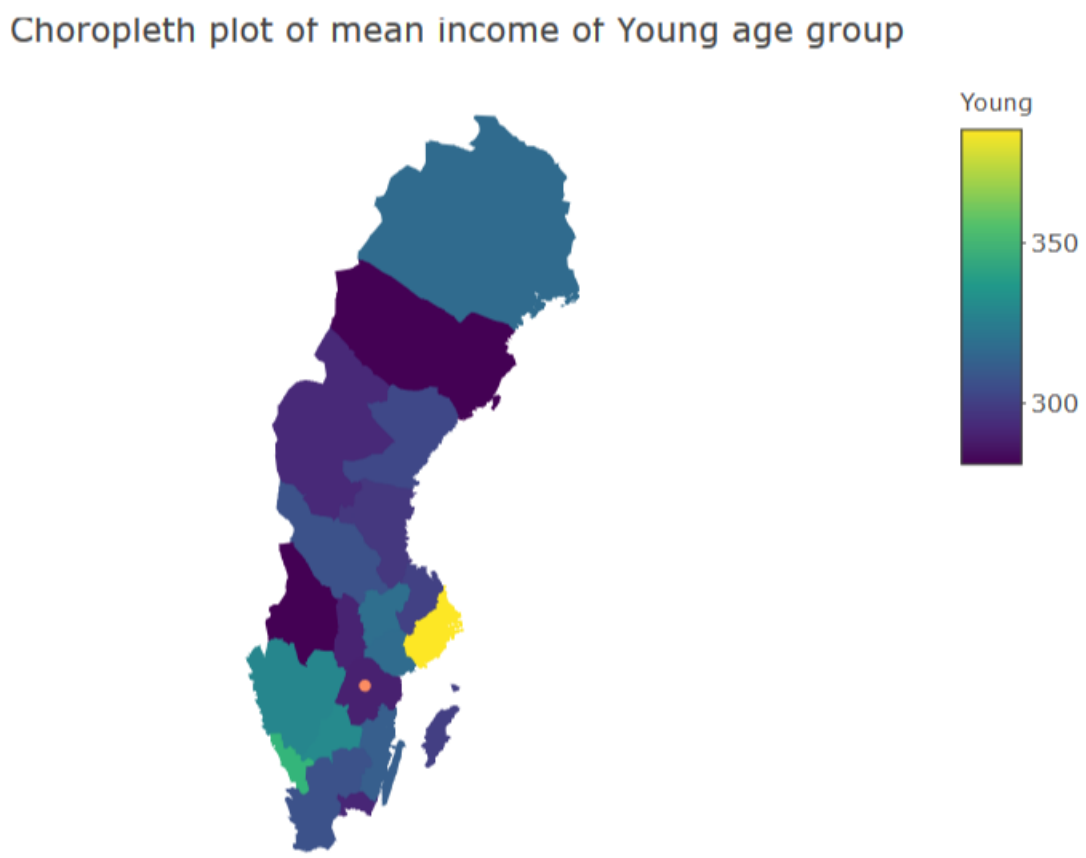
Choropleth plot using custom shape files(sf file)

```
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"]

x <- 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")

knitr::include_graphics('./3.8.4.png')
```



Choropleth plot with a custom maker

```
rds$Young = swe_data_processed[rds$NAME_1, "Young"]

x <- 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")

knitr::include_graphics('./3.8.5.png')
```



Violin Plots

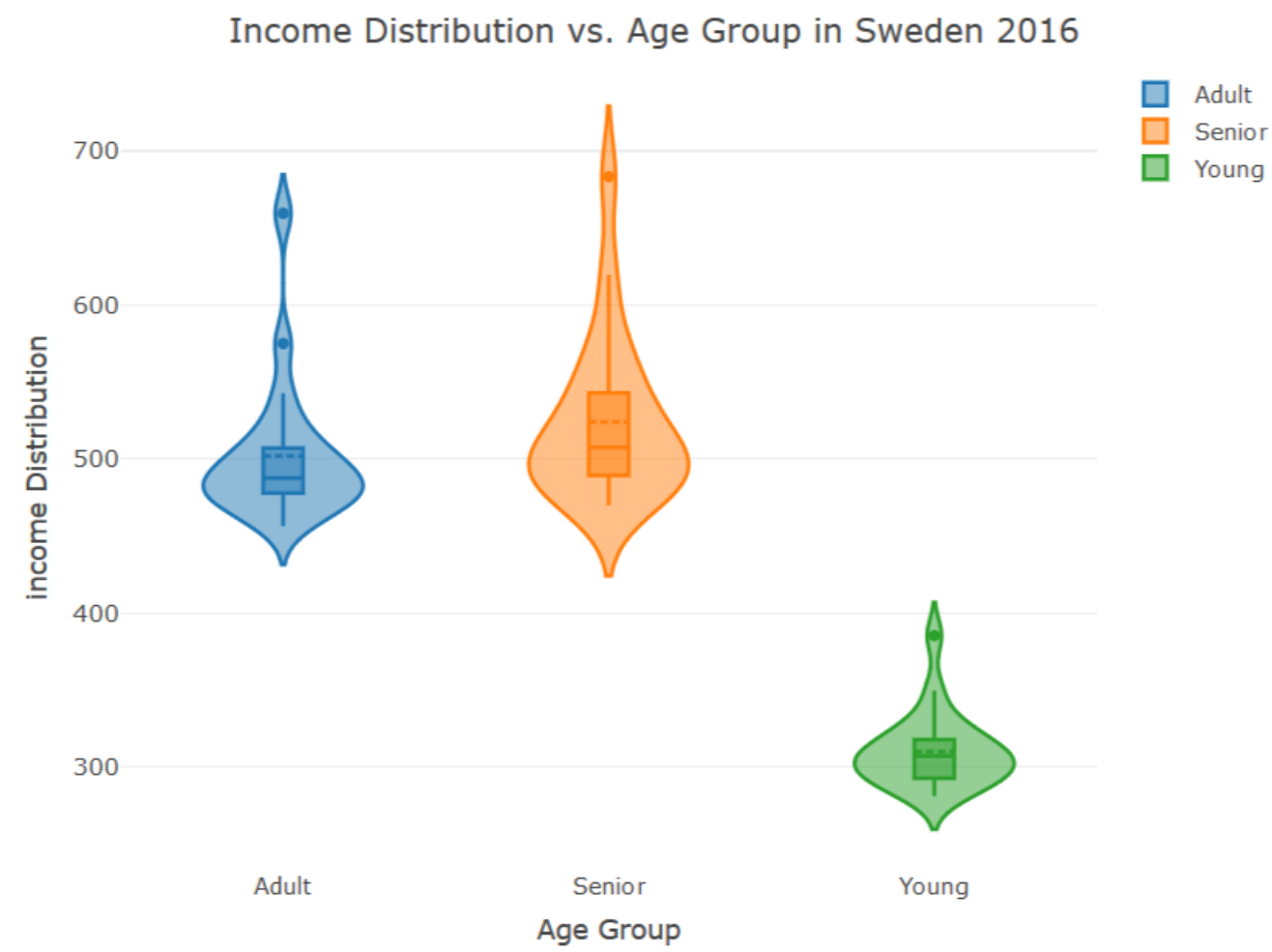
Violin Plot Simple 2 variable (one cateogrical and one numeric)

```
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

swe_data$age_group <- ifelse(swe_data$age == "18-29 years", "Young", ifelse(swe_data$age == "30-49 years
```

```
ggplot(swe_data) +
  geom_violin(aes(x = age, y = X2016, fill = age_group),
              draw_quantiles = c(0.25, 0.5, 0.75)) +
  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")
```

Violing using plotly

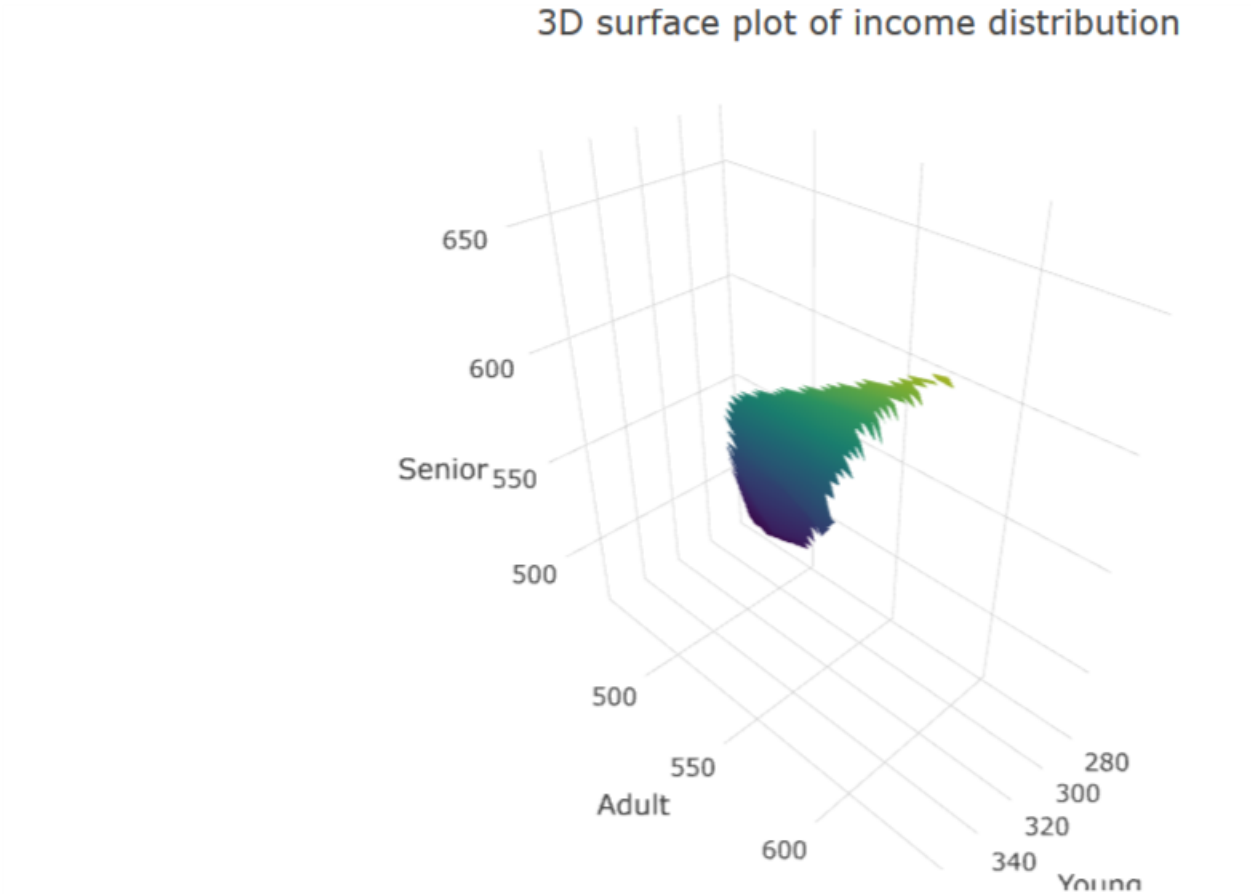
```
income_data_recasted <- dcast(income_data, region~age, value.var = "X2016")
setnames(income_data_recasted, old=c("18-29 years", "30-49 years", "50-64 years" ), new=c("Young", "Adul

income_data$age_group <- ifelse(income_data$age == "18-29 years", "Young",
                                ifelse(income_data$age == "30-49 years", "Adult","Senior"))

x <- income_data %>% plot_ly(x =~age_group ,y = ~X2016, type = 'violin', split = ~age_group, box = list

knitr::include_graphics('./3.9.1.png')
```



3D surface plots

3D surface plots using plotly

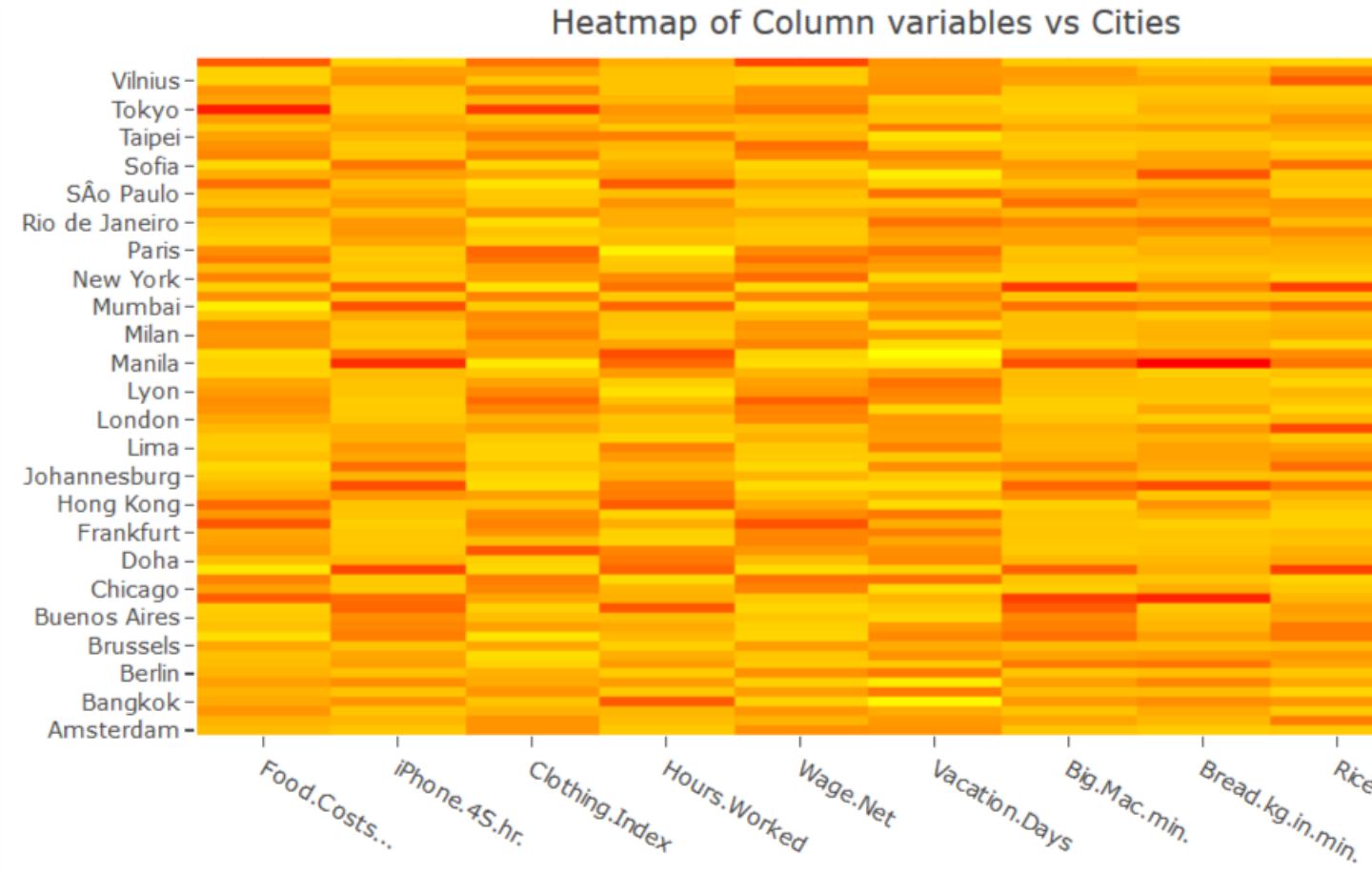
```
s = interp(swe_data_processed$Young, swe_data_processed$Adult,
           swe_data_processed$Senior, duplicate = "mean")

x <- 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")

knitr::include_graphics('./3.10.1.png')
```



Heat Map

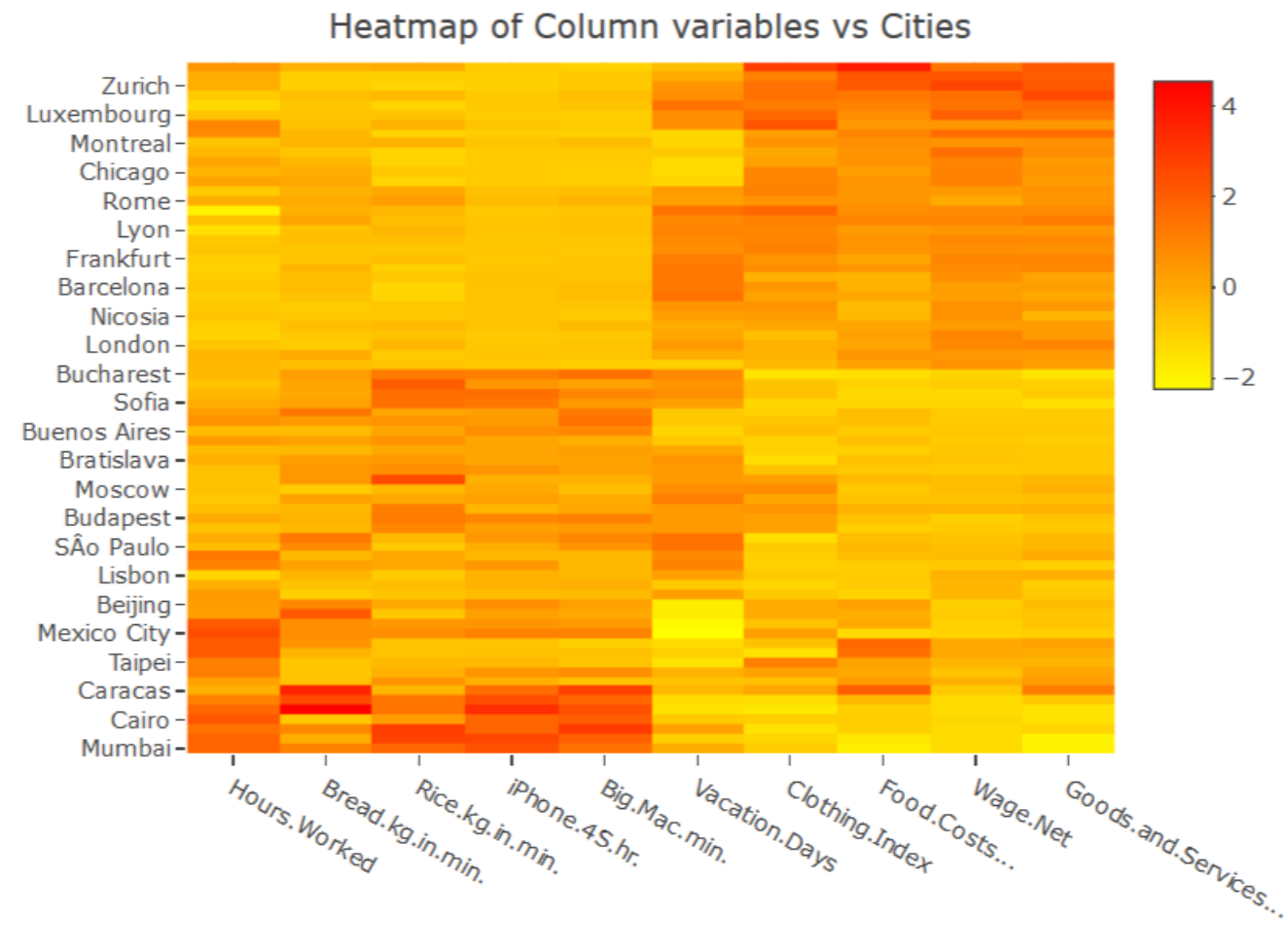
Heat Map without reordering

```
ubs_scaled = scale(ubs_data)

x <- 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")

knitr::include_graphics('./3.11.1.png')
```



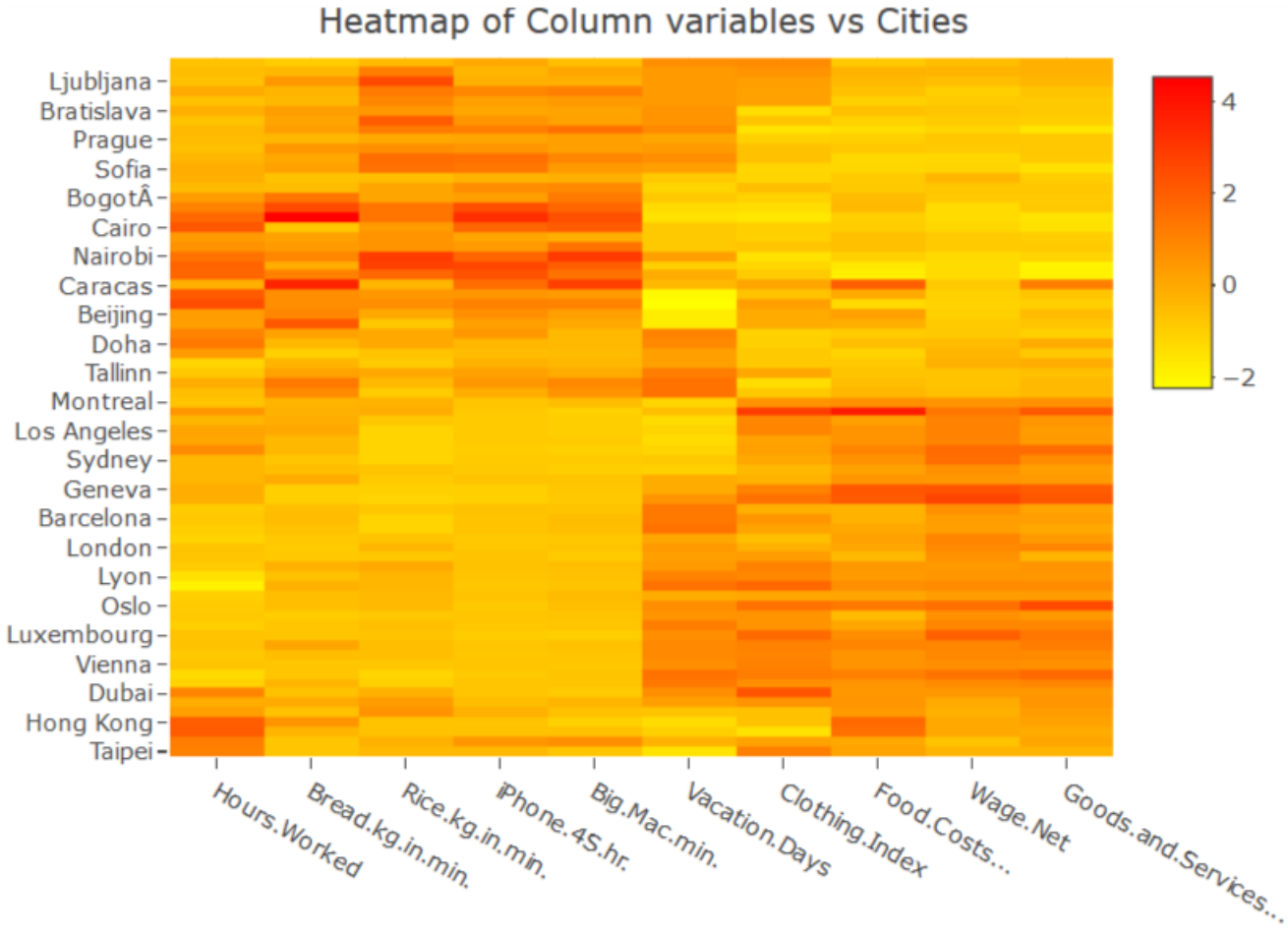
Heat Map with ordering using Hiearchical Clustering (HC) using Euclidean distance

```
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]

x <- 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")

knitr::include_graphics('./3.11.2.png')
```



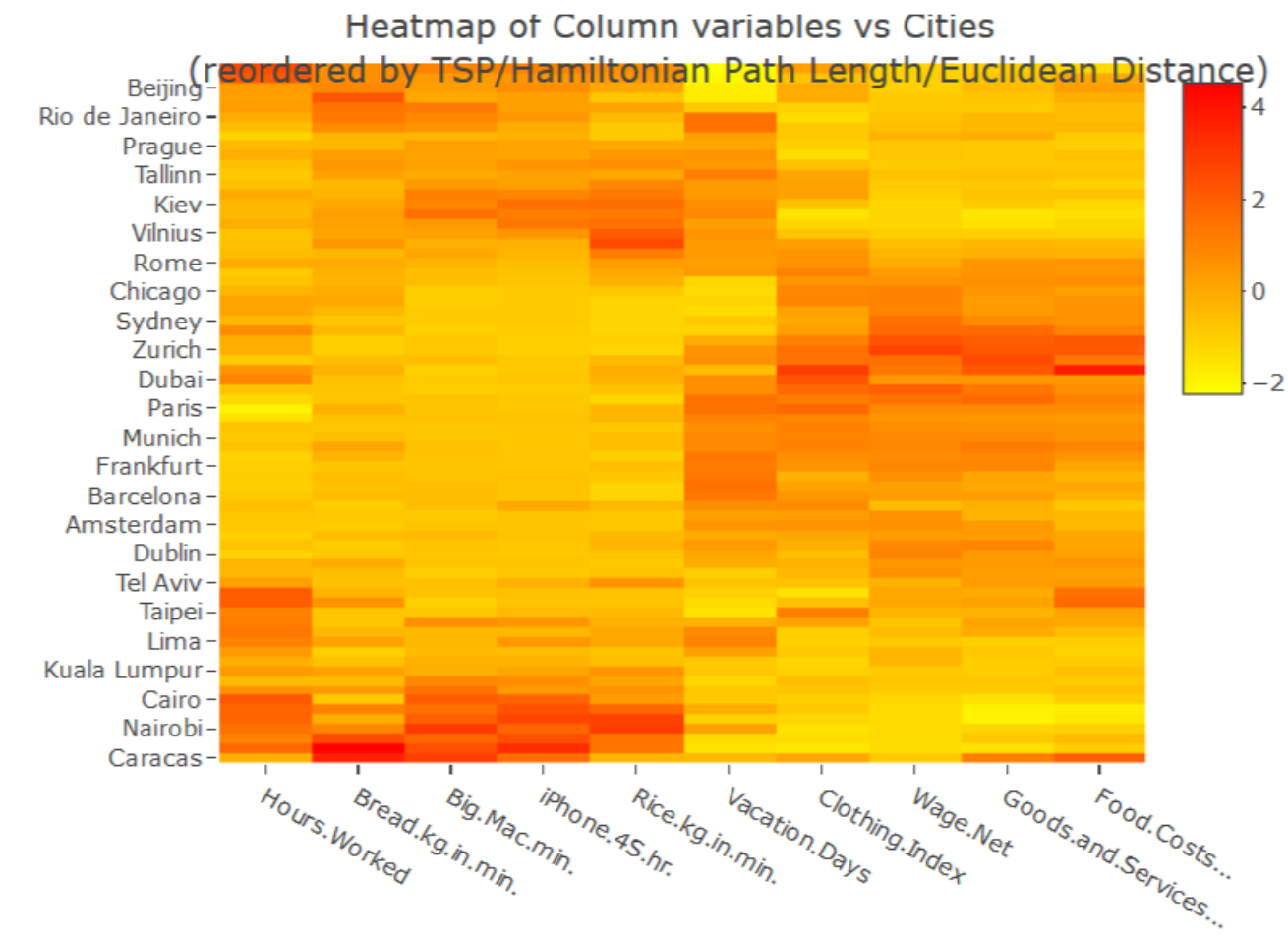
Heat Map with ordering using Corellation using Euclidean distance

```
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]

x <- 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")

knitr::include_graphics('./3.11.3.png')
```



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]

x <- 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)")

knitr::include_graphics('./3.11.4.png')
```



HeatMap QC and cluster information

```
library(heatmaply)
```

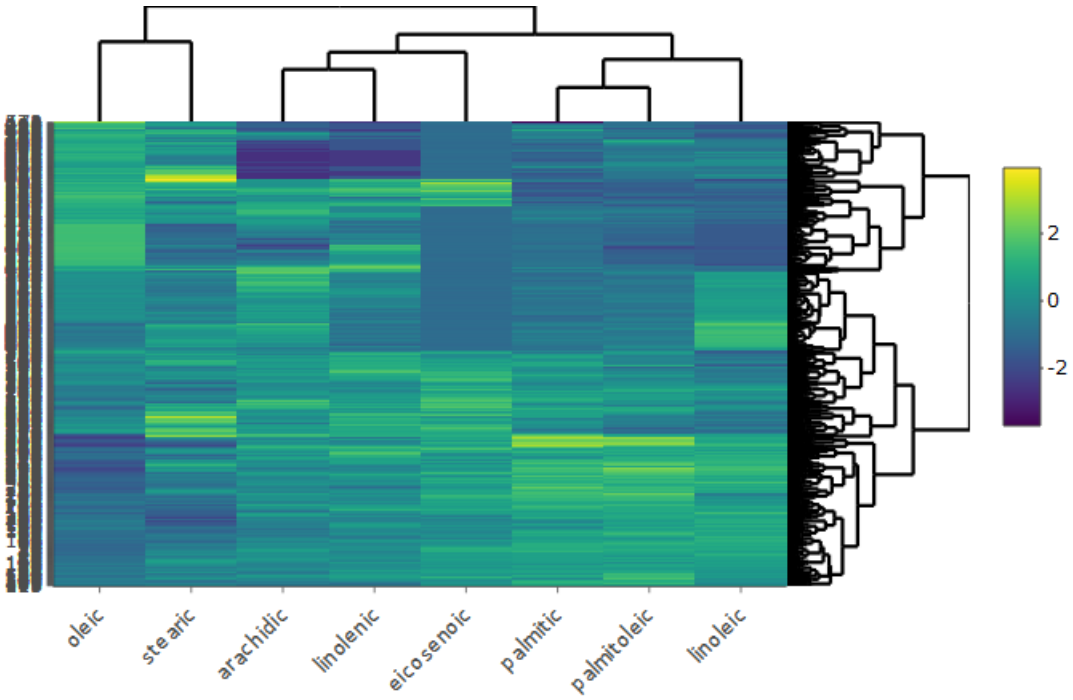
```
## Loading required package: viridis
## Loading required package: viridisLite
##
## Attaching package: 'viridis'
## The following object is masked from 'package:scales':
##
##     viridis_pal
##
## ======================
## Welcome to heatmaply version 0.15.2
##
## Type citation('heatmaply') for how to cite the package.
## Type ?heatmaply for the main documentation.
##
## The github page is: https://github.com/talgalili/heatmaply/
```

```
## Please submit your suggestions and bug-reports at: https://github.com/talgalili/heatmaply/issues
## Or contact: <tal.galili@gmail.com>
## ======================
```

```
##
## Attaching package: 'heatmaply'
```

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

```
x <- heatmaply(ubs_reordered_tsp)
knitr::include_graphics('./heatmap_qc.png')
```



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 :  120.217615573872"
```

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

```
## [1] "Gradient Measure :  21232"
```

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

```r
paste("Hamiltonian Path Length : ", ham_hc)
```

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

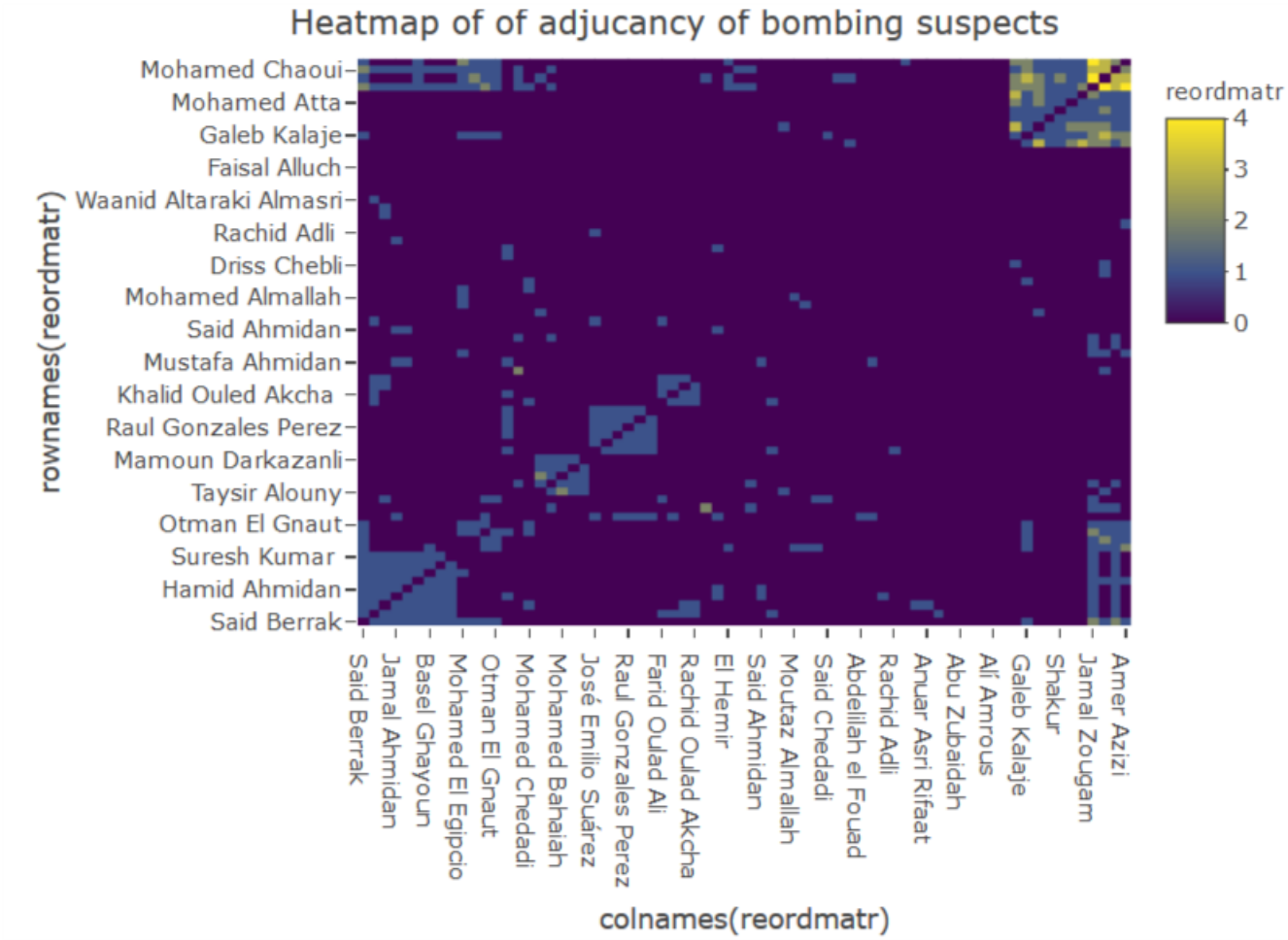
```r
#Gradient Measure
gm_hc = criterion(row_dist_euc, order = ord_hc, "Gradient_raw")
paste("Gradient Measure : ", gm_hc)
```

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

Heat Map using Adjacency

```r
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]

x <- plot_ly(z=~reordmatr, x=~colnames(reordmatr),
        y=~rownames(reordmatr), type="heatmap") %>%
layout(title = "Heatmap of of adjucancy of bombing suspects")

knitr::include_graphics('./3.11.6.png')
```

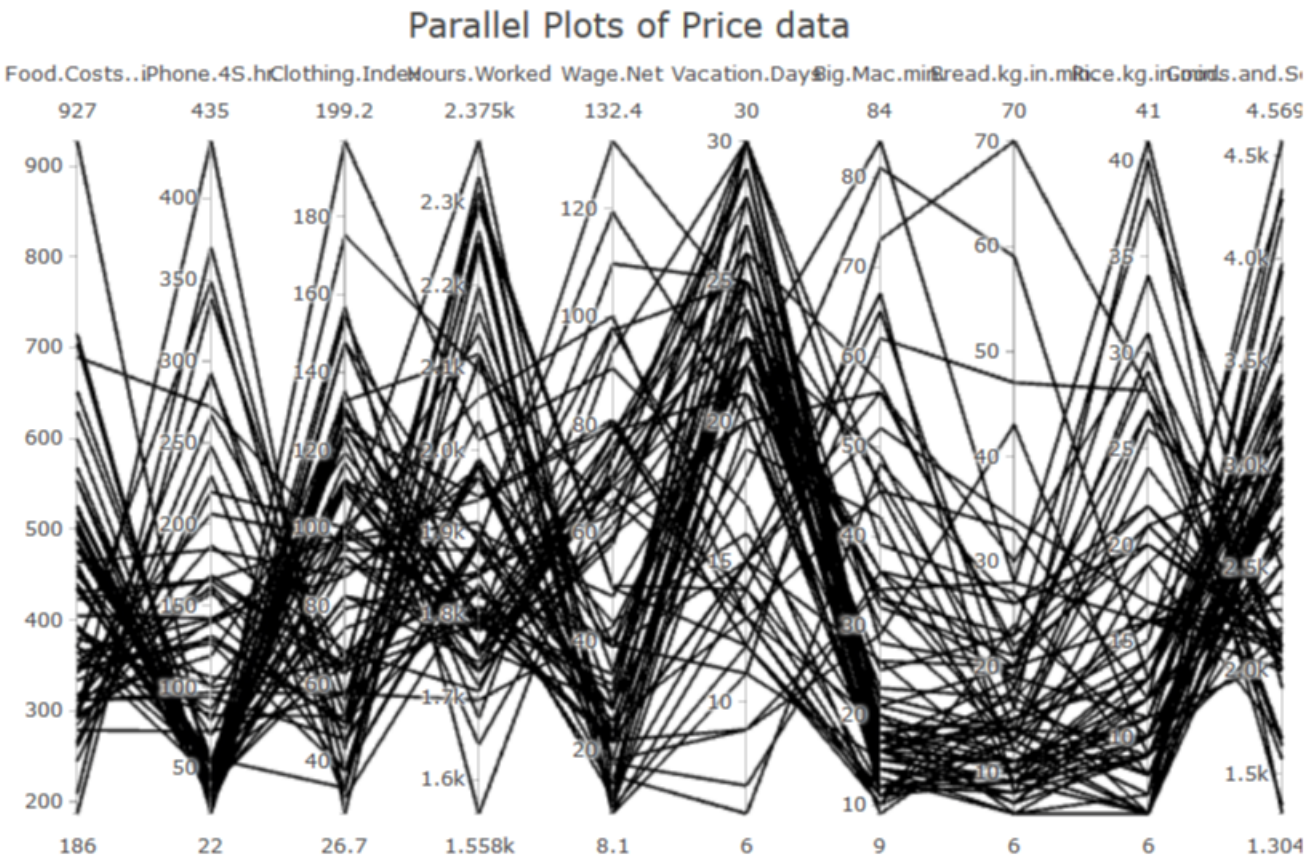


Parallel Plot

Parallel Plot using Plotly

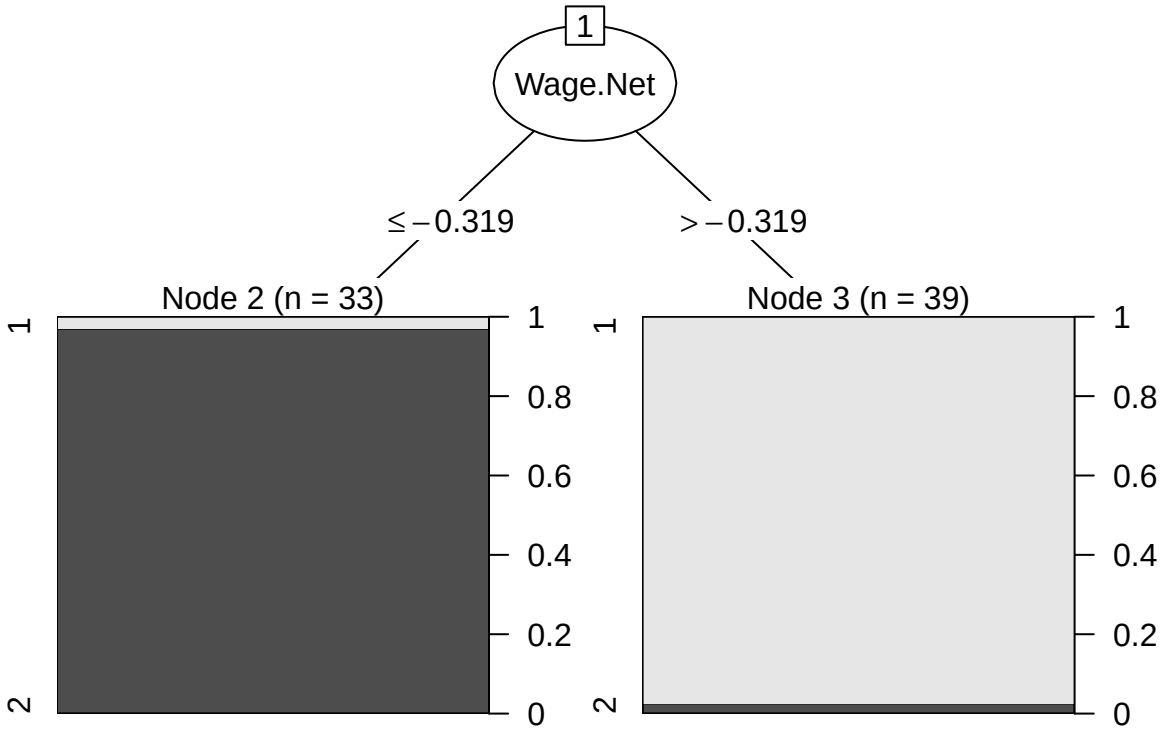
```
# Function to create dimension list based on given variable order
get_dimension_list <- function(df, col_order){
  dim_list = list()
  i = 1
  for (col in col_order){
    dim_list[[i]] = list(label = col, values = df[[col]])
    i = i + 1
  }
```

```
  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

x <- plot_ly(data = ubs_data, type = 'parcoords',
        dimensions = unord_dim_list) %>%
  layout(title = "Parallel Plots of Price data")

knitr::include_graphics('./3.12.1.png')
```



QC of Parallel plots

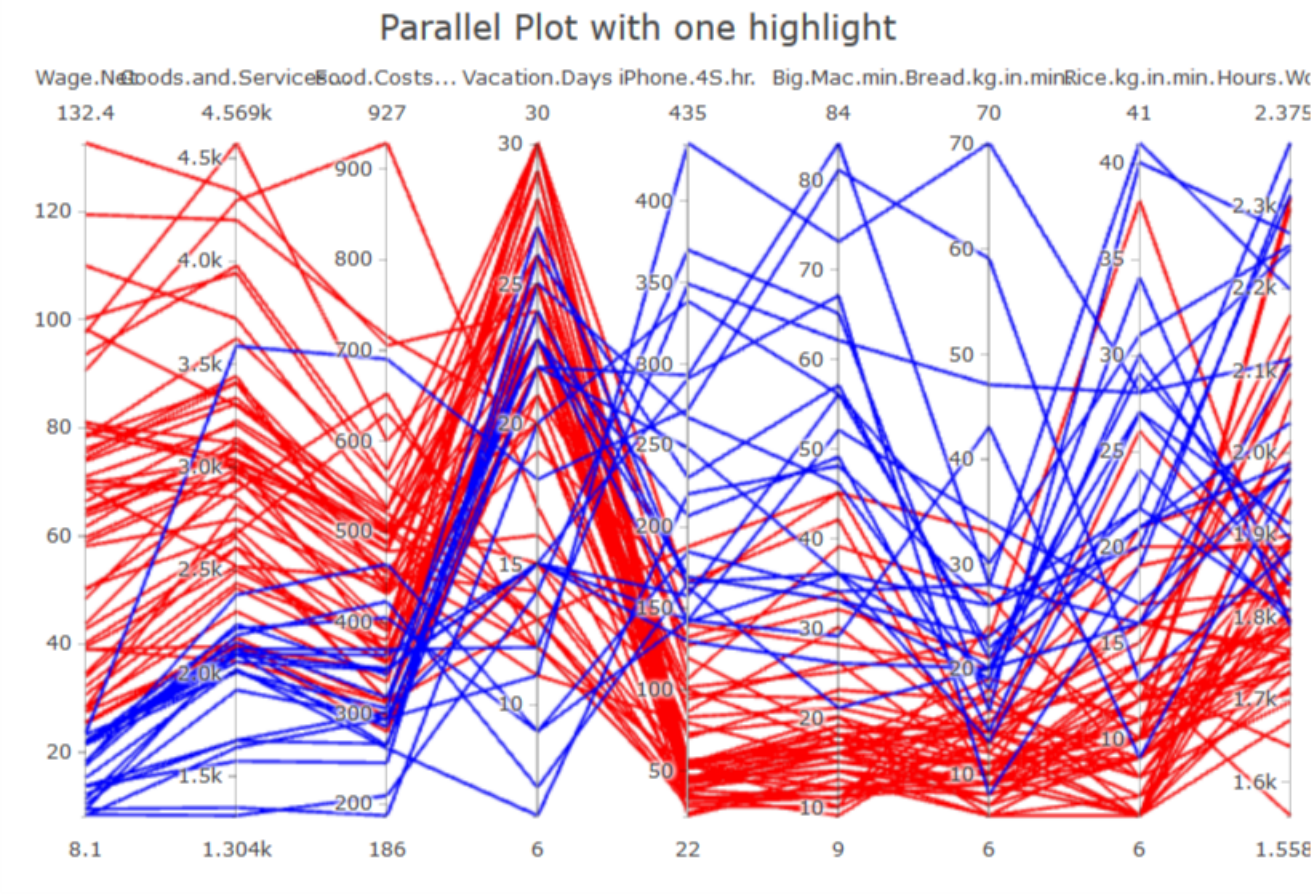
```
df = read.delim("prices-and-earnings.txt", header = TRUE)
df = df[, c(2,5,6,7,9,10,16,17,18,19)]

df = scale(df)

temp <- kmeans(x = df, centers = 2, iter.max = 10)


df <- as.data.frame(df)
df2 <- df

df$cluster <- as.numeric(temp$cluster)
df$cluster <- as.factor(df$cluster)

temp2 <- C50::C5.0(x = df2 , y=df$cluster, rules = FALSE)

plot(temp2)
```



Parallel Plot using Plotly and highlight one line

```
x <- 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 = "Parallel Plot with one highlight")

knitr::include_graphics('./3.12.2.png')
```

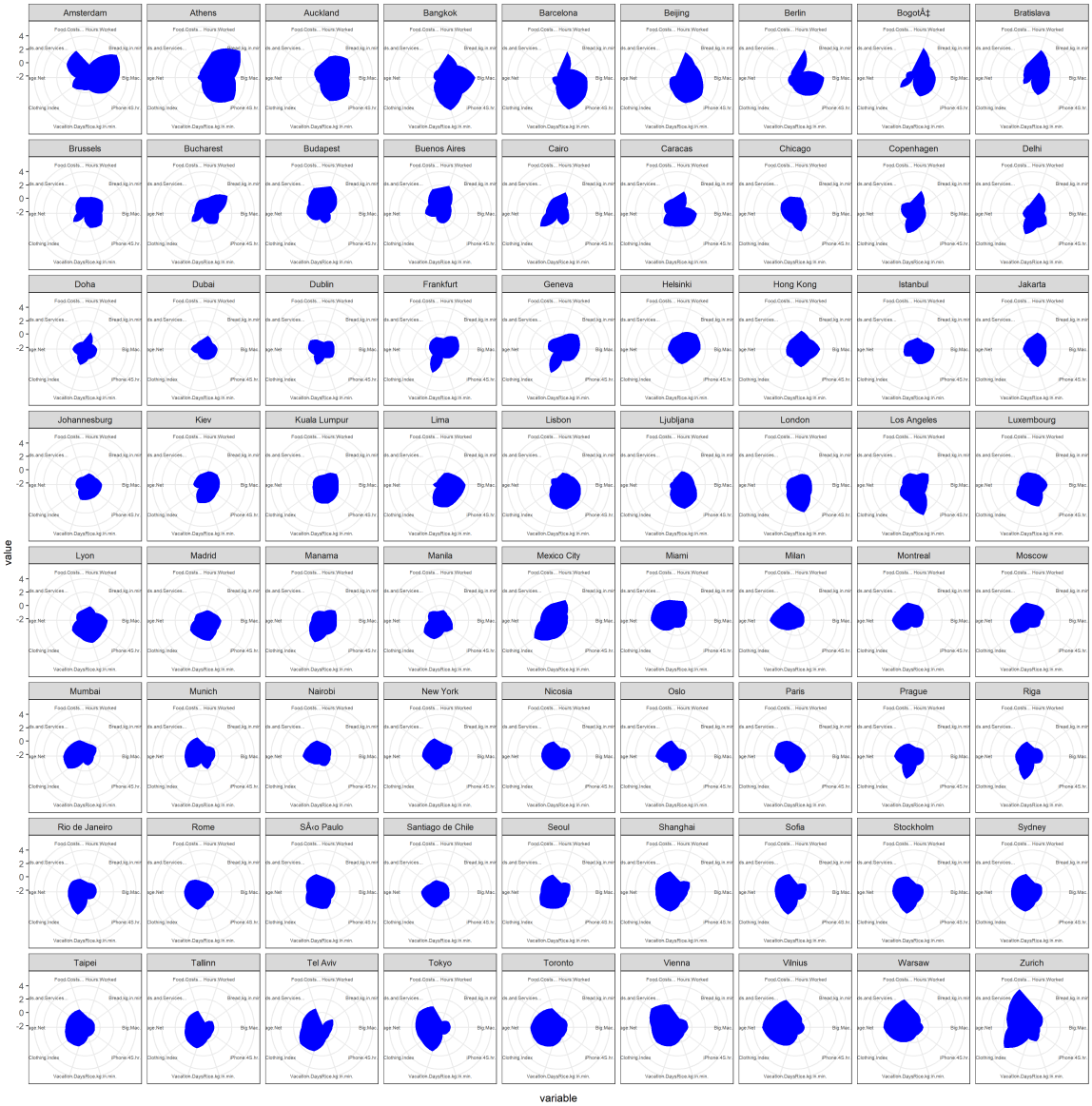


Radar Plots

Radar Plots using Scatterpolar

```
price_data_scale <- scale(price_data[,-1])

price_row_dist <- dist(x=price_data_scale, method = "euclidean", diag = TRUE)
price_col_dist <- dist(x=t(price_data_scale), method = "euclidean", diag = TRUE)


order1 <- seriate(price_row_dist, "OLO")
order2 <- seriate(price_col_dist, "OLO")
ord1 <- get_order(order1)
ord2 <- get_order(order2)

reordmatr <- price_data_scale[rev(ord1),ord2]
reordmatr <- as.data.frame(reordmatr)
```

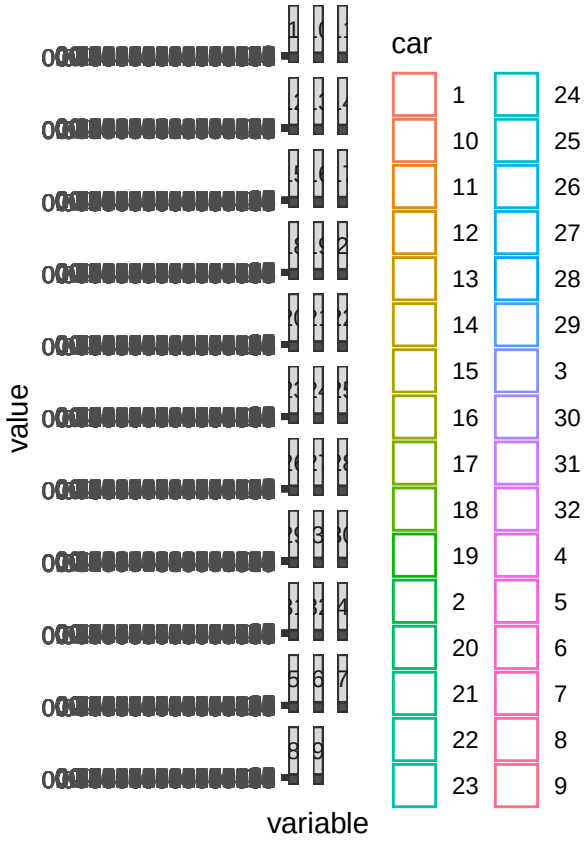
```r
reordmatr$City = price_data$City

reordmatr_transformed <- reordmatr%>%tidyr::gather(variable, value, -City, factor_key=T)%>%arrange(City)

radar_plot <- reordmatr_transformed %>% ggplot(aes(x=variable, y=value, group=City)) + geom_polygon(fill

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

knitr::include_graphics('./radar_plot.png')
```


Amsterdam
Athens
Auckland
Bangkok
Barcelona
Beijing
Berlin
BogotÃ‡
Bratislava
Brussels
Bucharest
Budapest
Buenos Aires
Cairo
Caracas
Chicago
Copenhagen
Delhi
Doha
Dubai
Dublin
Frankfurt
Geneva
Helsinki
Hong Kong
Istanbul
Jakarta
Johannesburg
Kiev
Kuala Lumpur
Lima
Lisbon
Ljubljana
London
Los Angeles
Luxembourg
Lyon
Madrid
Manama
Manila
Mexico City
Miami
Milan
Montreal
Moscow
Mumbai
Munich
Nairobi
New York
Nicosia
Oslo
Paris
Prague
Riga
Rio de Janeiro
Rome
SÃ£o Paulo
Santiago de Chile
Seoul
Shanghai
Sofia
Stockholm
Sydney
Taipei
Tallinn
Tel Aviv
Tokyo
Toronto
Vienna
Vilnius
Warsaw
Zurich
value
variable

Radar plot using simple options

```
#{r}

df <- as.matrix(mtcars)
df2 <- rescale(df)
df2 <- as.data.frame(df2)
df2$car <- rownames(df2)

df3 <- melt(df2 ,id.vars=c('car'), measure.vars=colnames(df2))

radar_plot2 <- df3 %>%
 ggplot(aes(x=variable, y=value, group=car, color=car)) +
 geom_polygon(fill=NA) +
 coord_polar() + theme_bw() + facet_wrap(~ car, ncol = 3)
```

```
radar_plot2
```

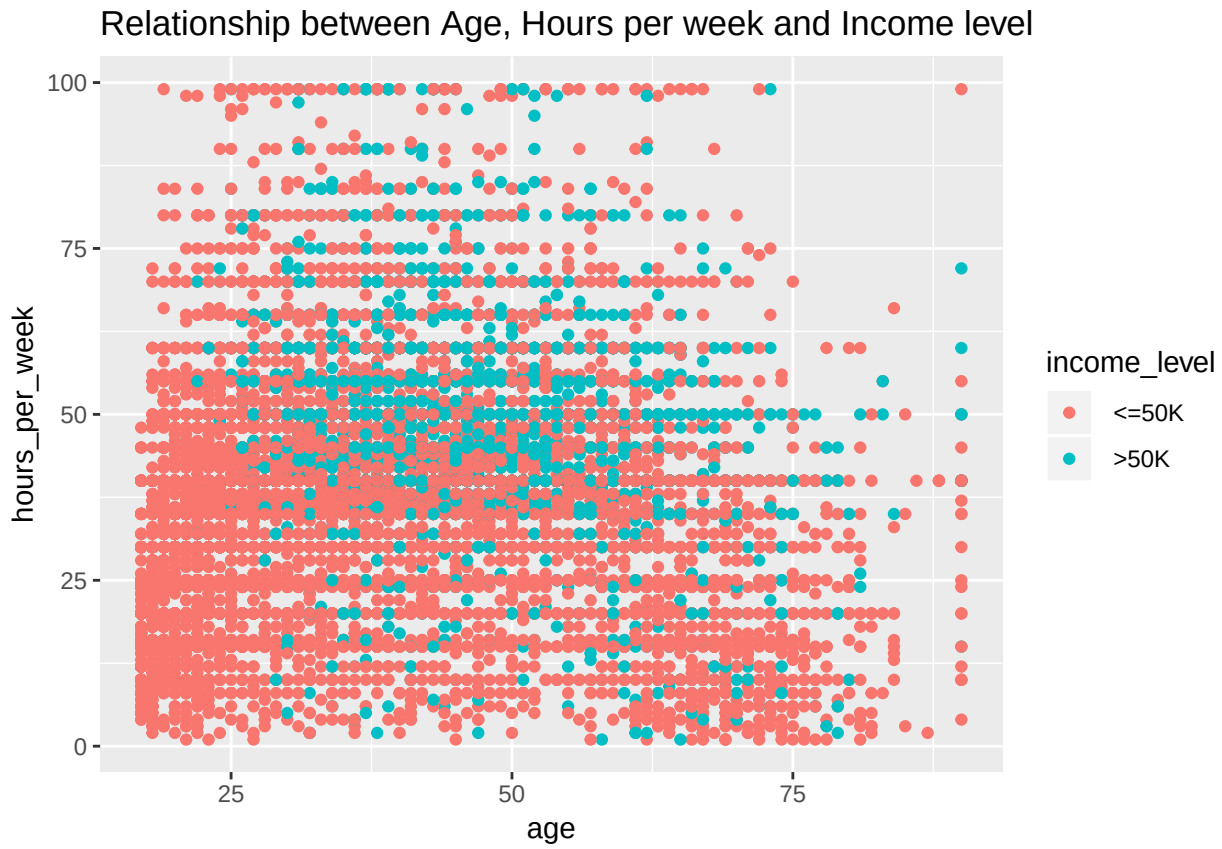


Trellis Plots

Trellis Plots using ggplot2

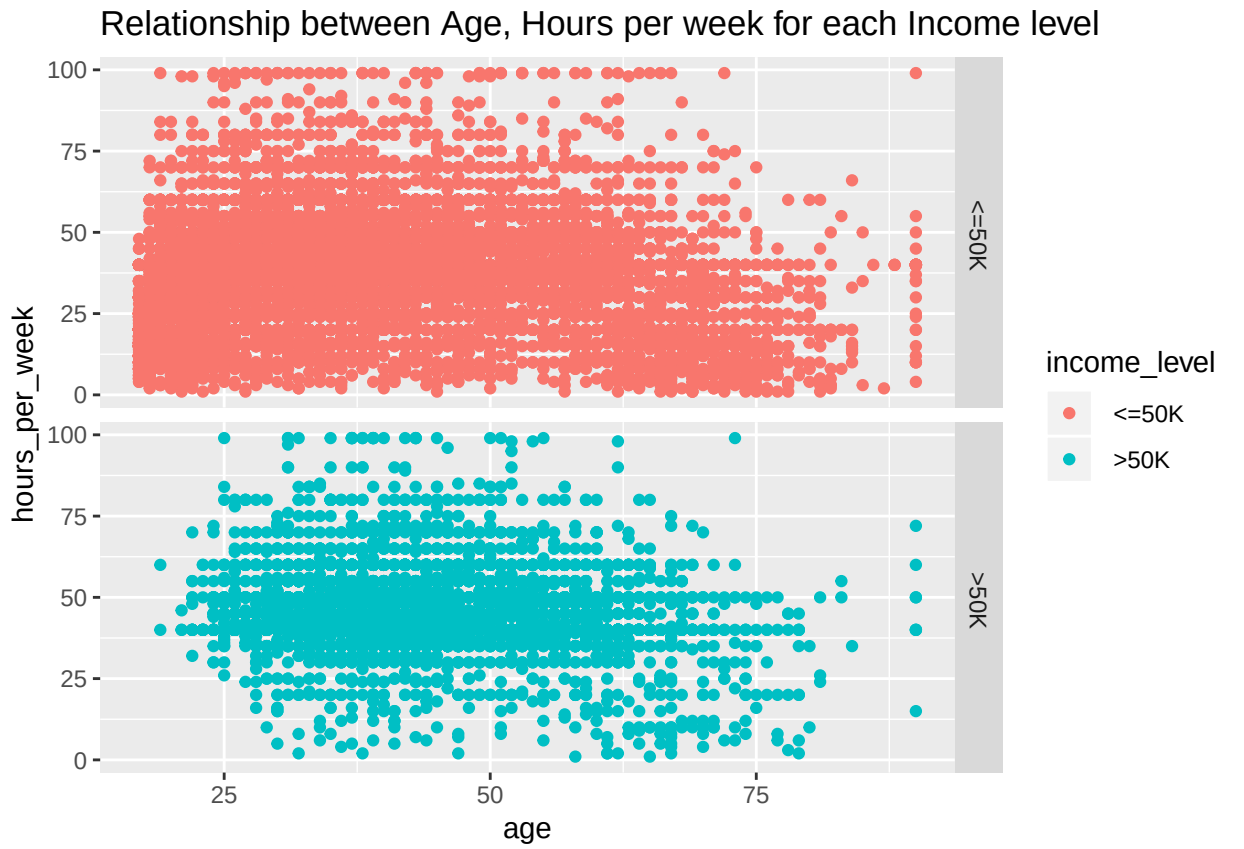
```
ggplot(adult_data) +
  geom_point(aes(x = age, y = hours_per_week, color = income_level)) +
```

```
  ggtitle("Relationship between Age, Hours per week and Income level")
```



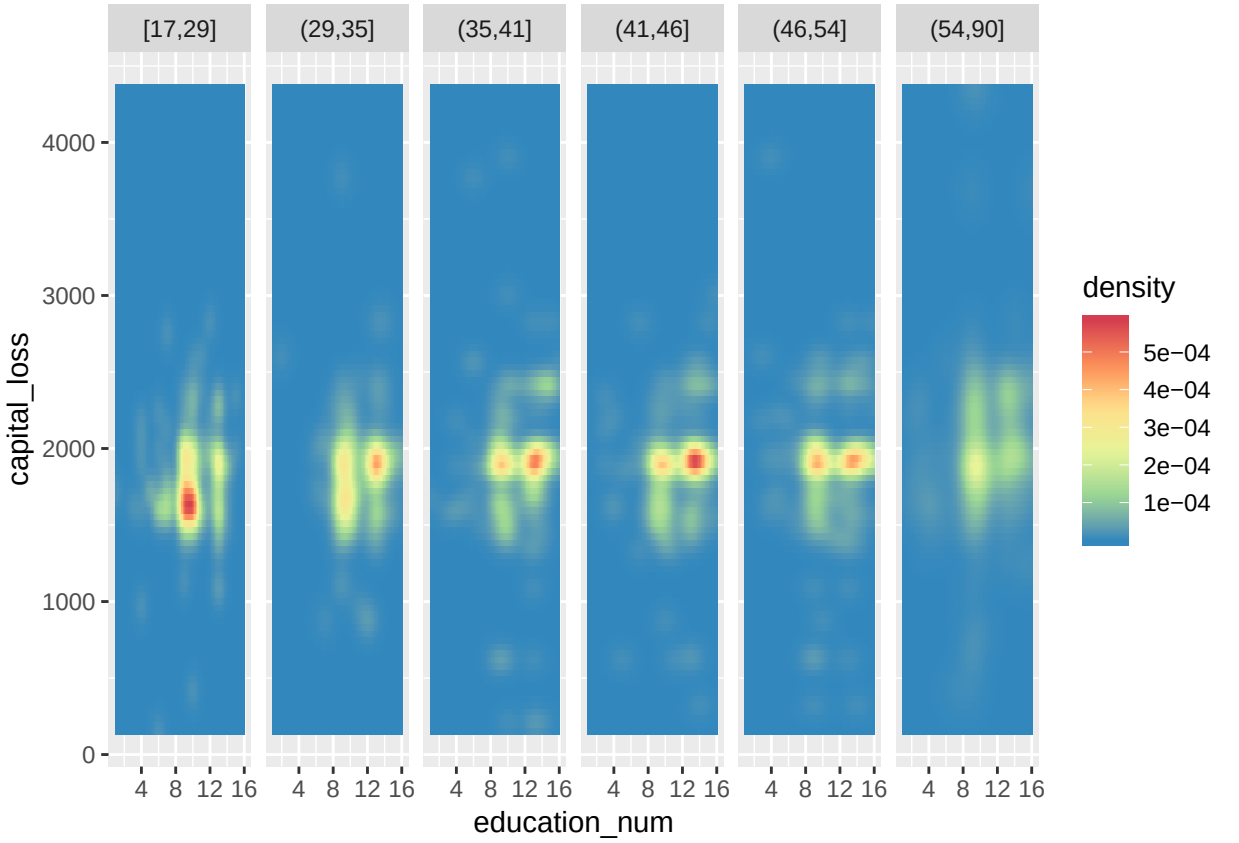
Trellis Plots with facet/condition on a variable

```
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")
```



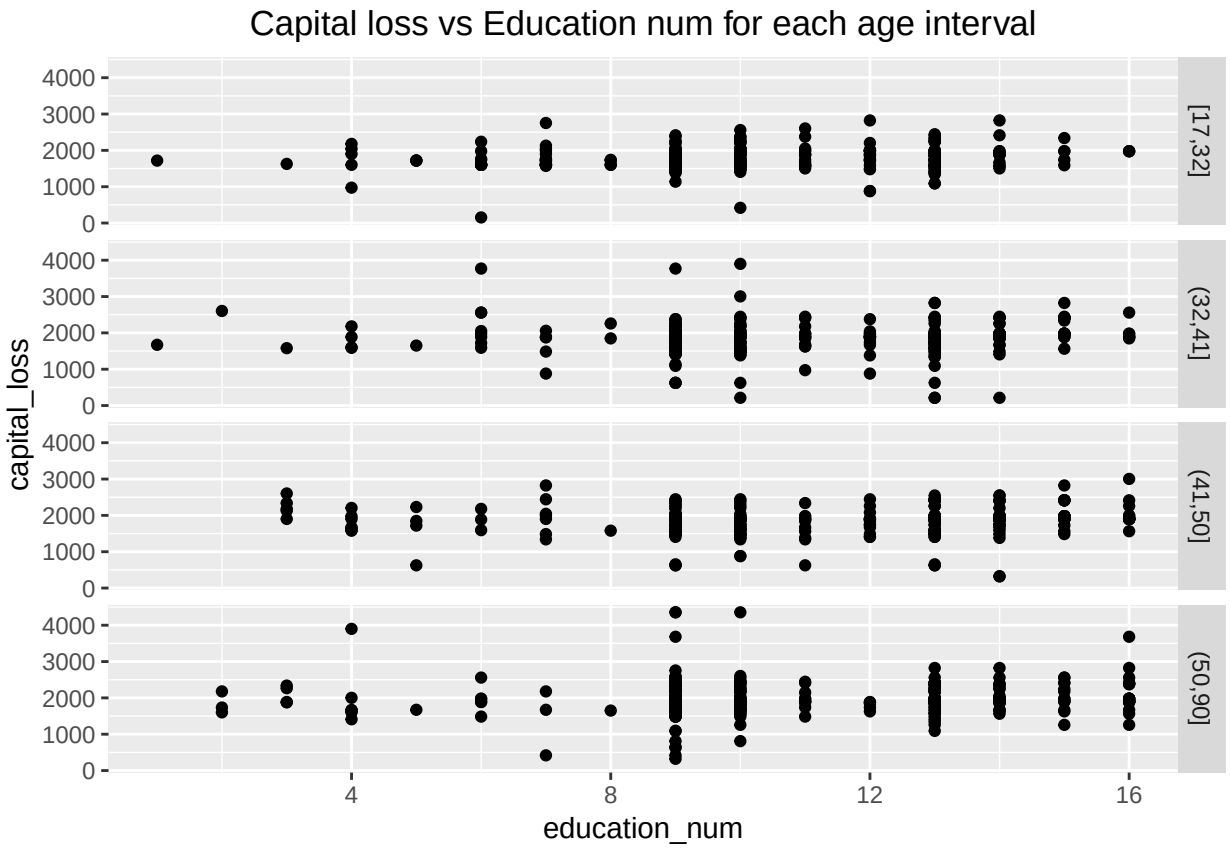
Trellis Plot with raster-type-2d-density plot

```
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)))
```



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

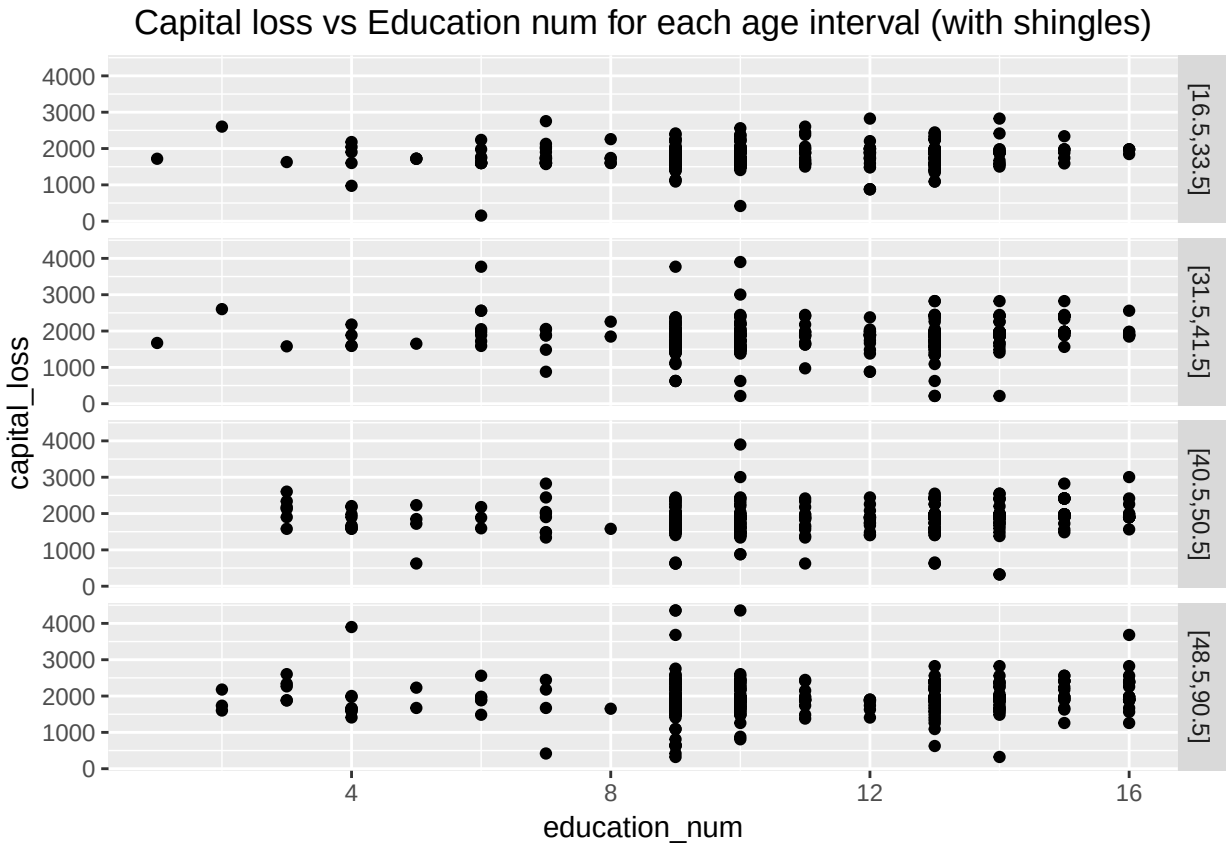
```
ggplot(capital_loss_data) +
  geom_point(aes(x = education_num, y = capital_loss)) +
  facet_grid(vars(cut_number(age, 4))) +
  ggtitle("Capital loss vs Education num for each age interval") +
  theme(plot.title = element_text(hjust = 0.5))
```



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))
```



Word Clouds

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, colors=p

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, colors=p
```

Linked Plots

Bar chart and Scatter Plot

Bar chart and scatter plot shared using crosstalk

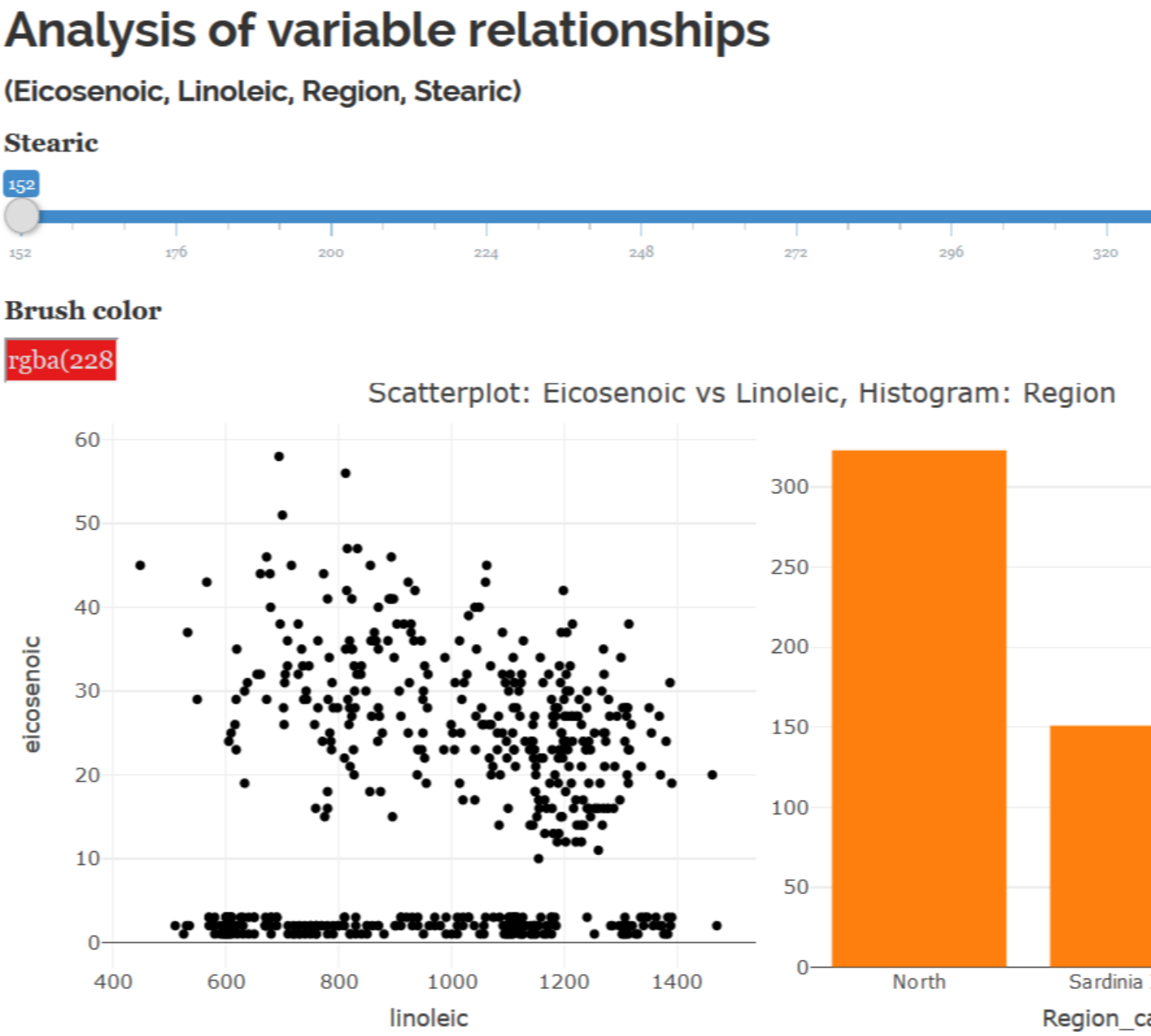
```
olive_data = read.csv("olive.csv")

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 can now
```

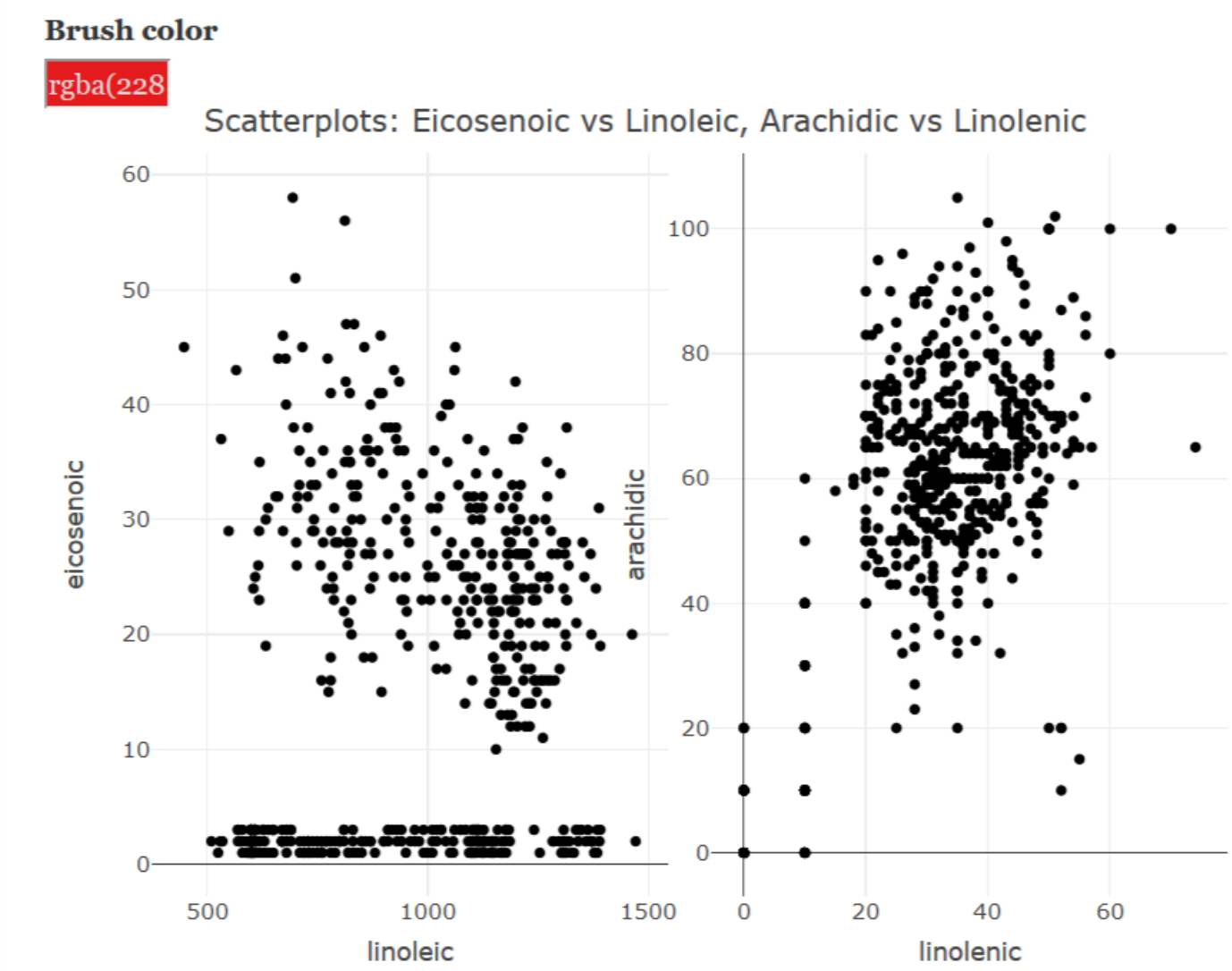
```
x <- bscols(widths = c(12), list(tags$h2("Analysis of variable relationships"),
                                 tags$h4("(Eicosenoic, Linoleic, Region, Stearic)"),
                                 filter_slider("stearic", "Stearic", ct_olive_2, ~stearic), plots))

knitr::include_graphics('./4.1.1.png')
```



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"))
```

```
x <- 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")

knitr::include_graphics('./4.1.2.png')
```

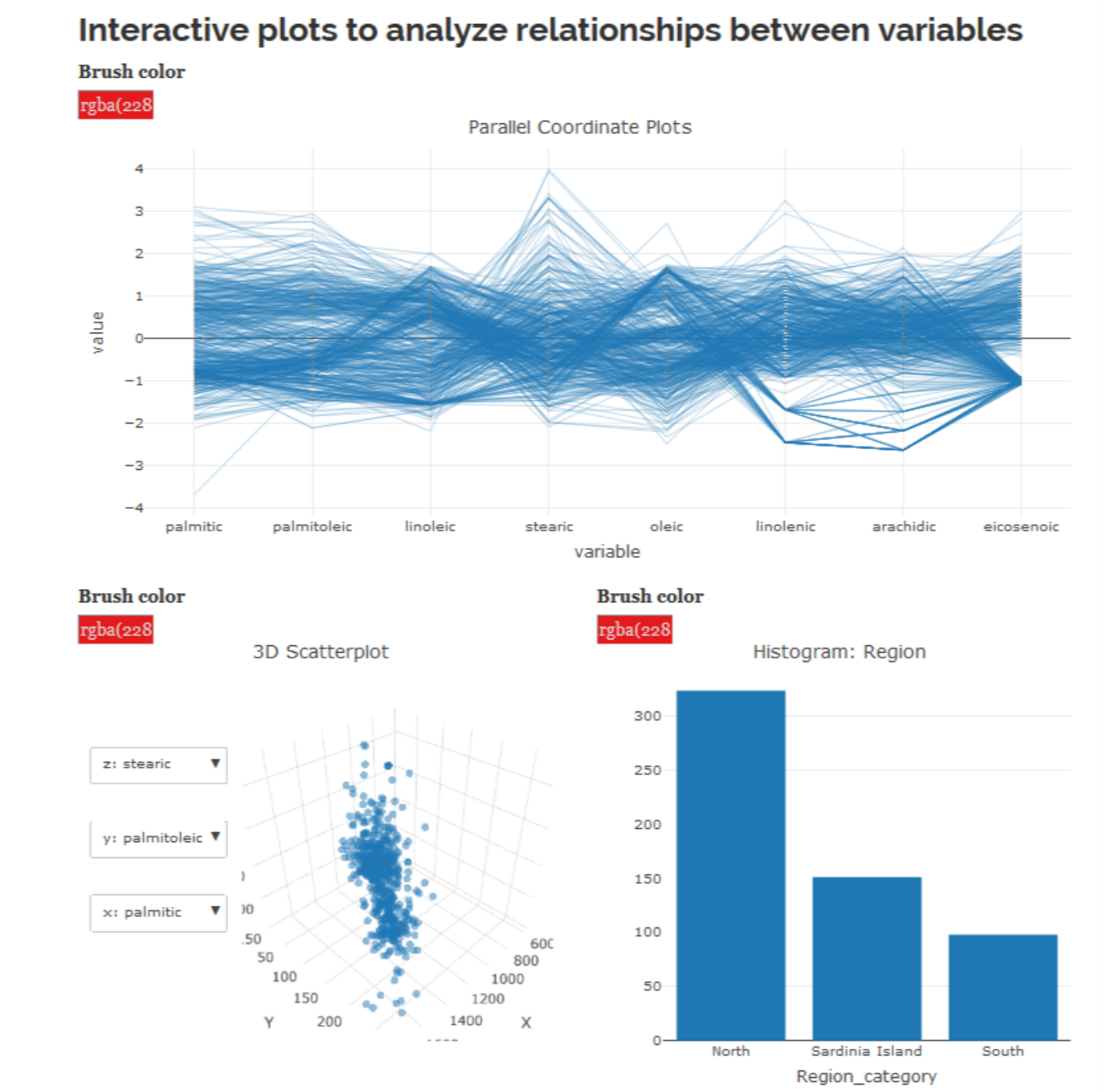


Parallel Chord/Trellis Plot Scatter plot and Bar chart

```
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")

x <- 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())
```

```
knitr::include_graphics('./4.1.3.png')
```

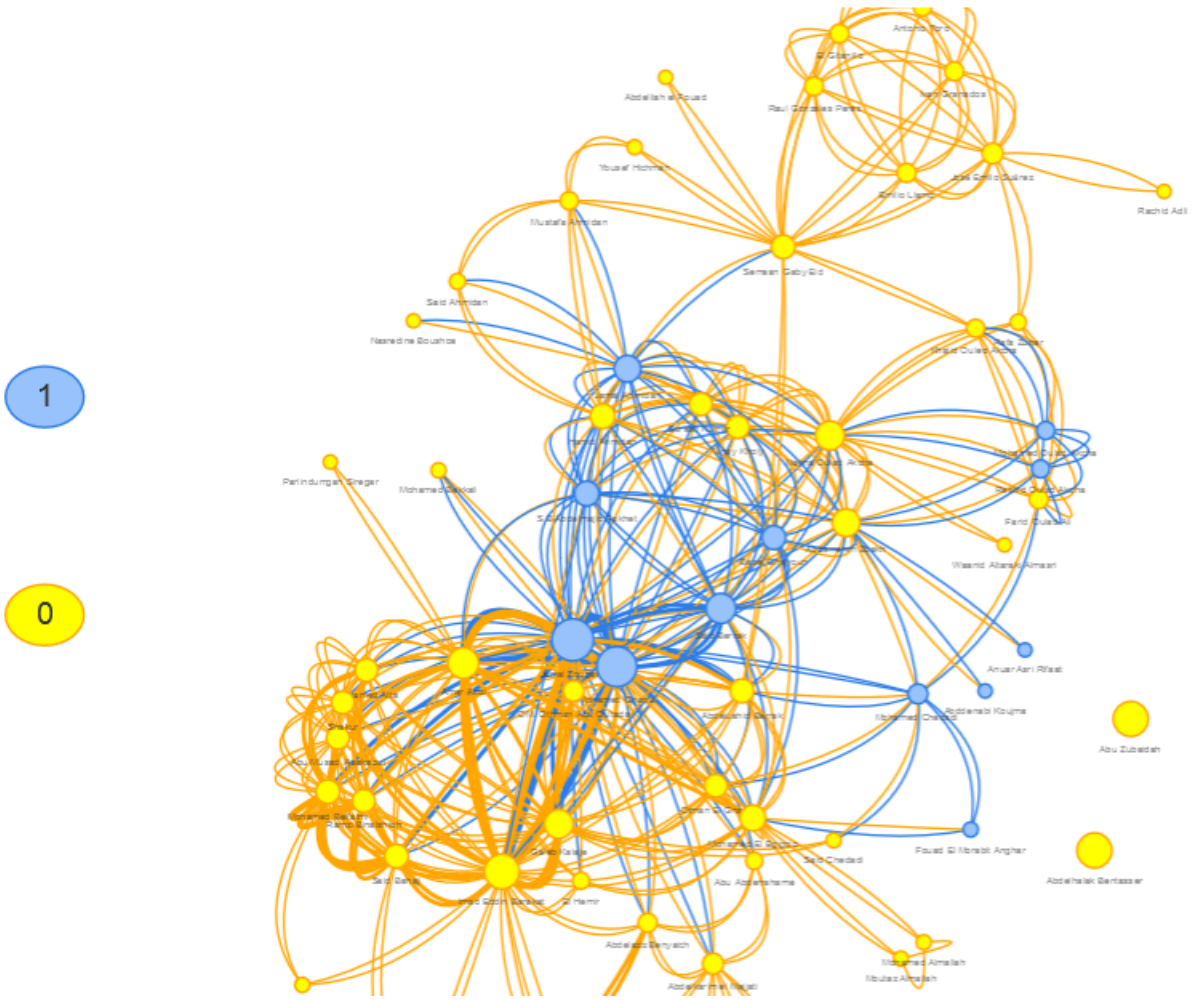


Network Graphs

Network Graph using visNetwork

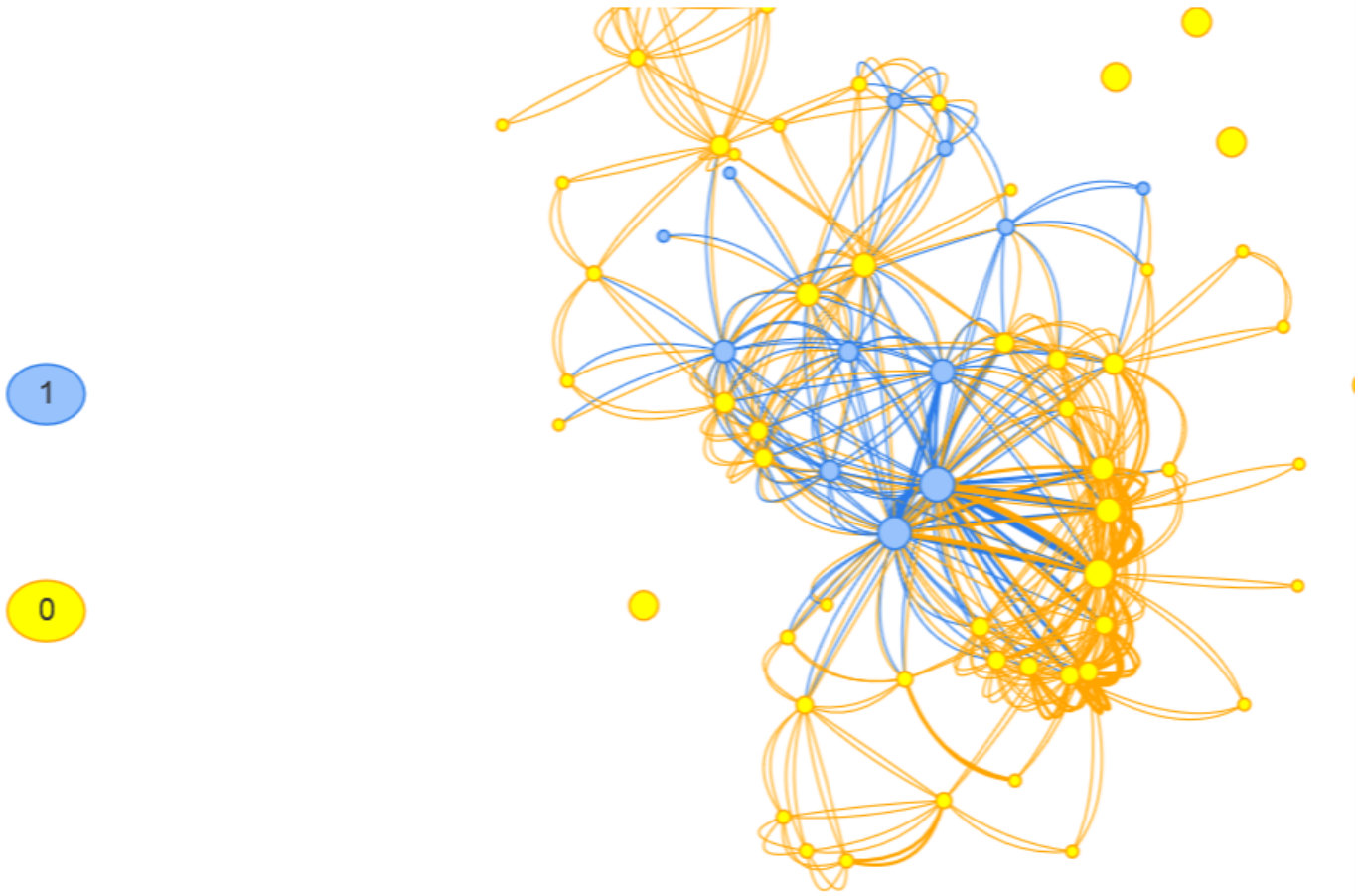
Replusion Optimzed Network Graph

```
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))]

x <- 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))

knitr::include_graphics('./5.1.1.png')
```



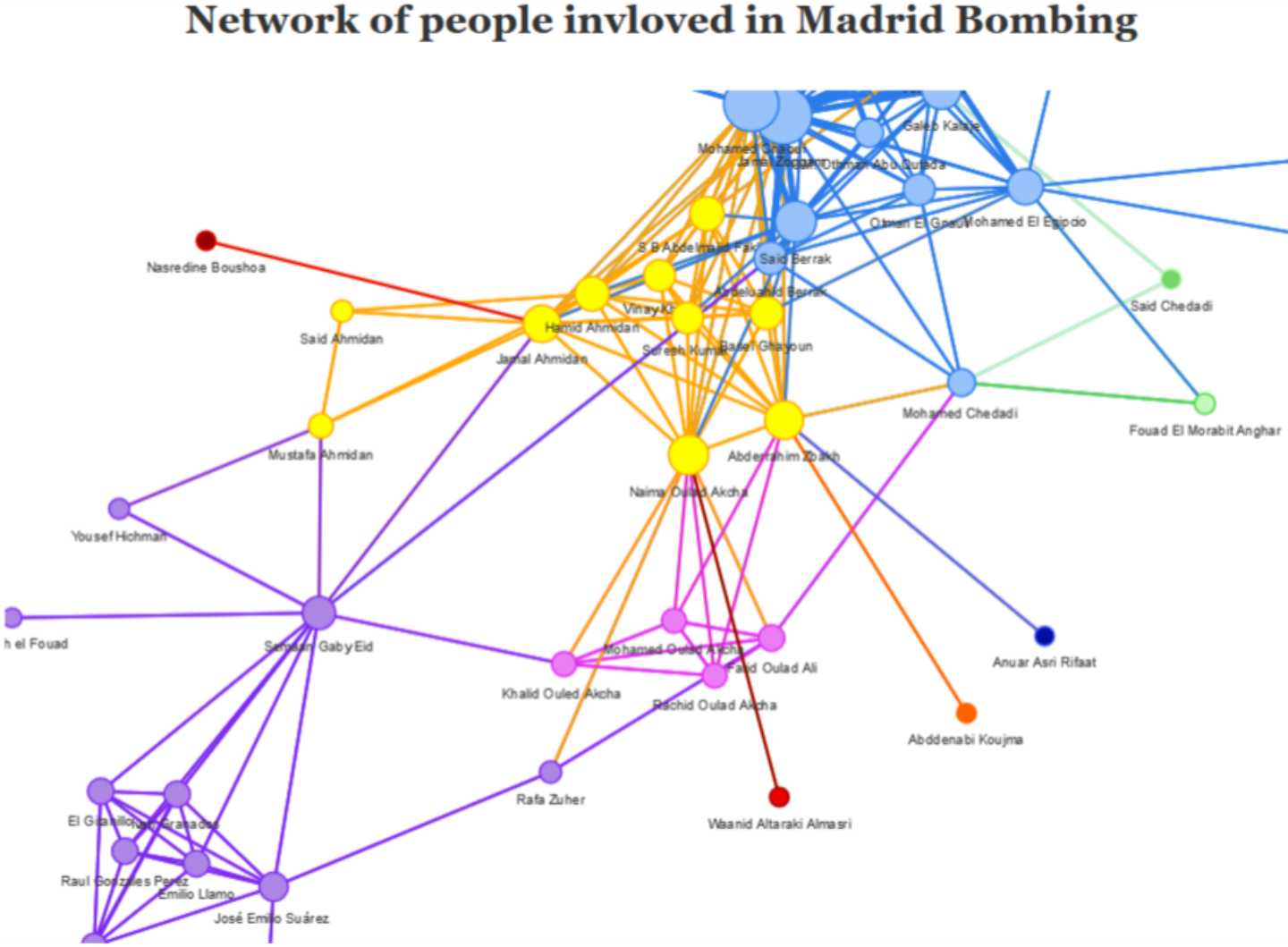
Replusion Optimzed Network Graph with highlights

```
x <- 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))

knitr::include_graphics('./5.1.2.png')
```



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

x <- visNetwork(nodes1,links, main = "Network of people invloved in Madrid Bombing") %>%
  visIgraphLayout() %>%
  visOptions(highlightNearest = list(enabled = T, degree = 2, hover = T))

knitr::include_graphics('./5.1.3.png')
```

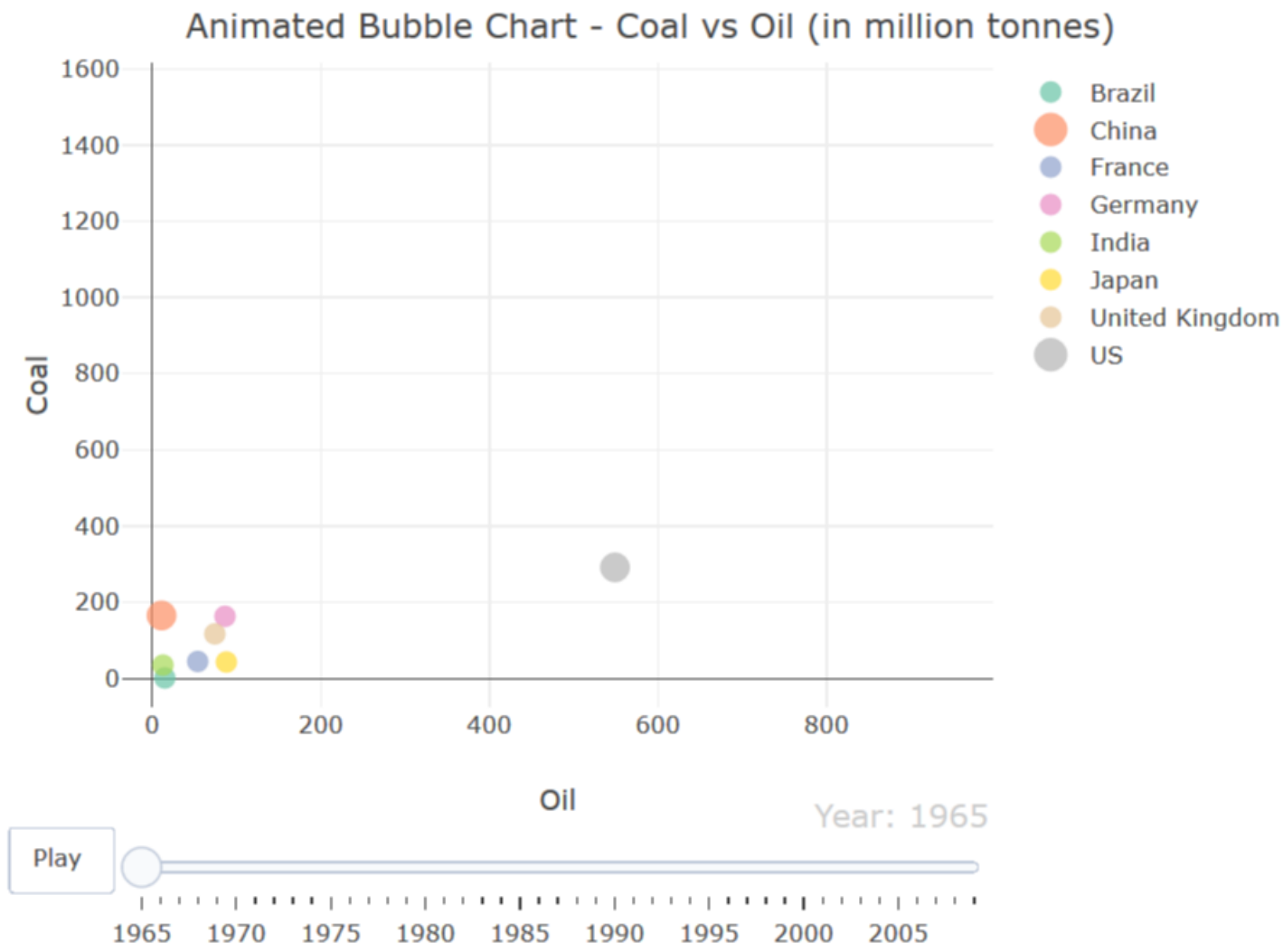


Animated Plots

Bubble Chart

Bubble Chart Animated with Cubic Easing

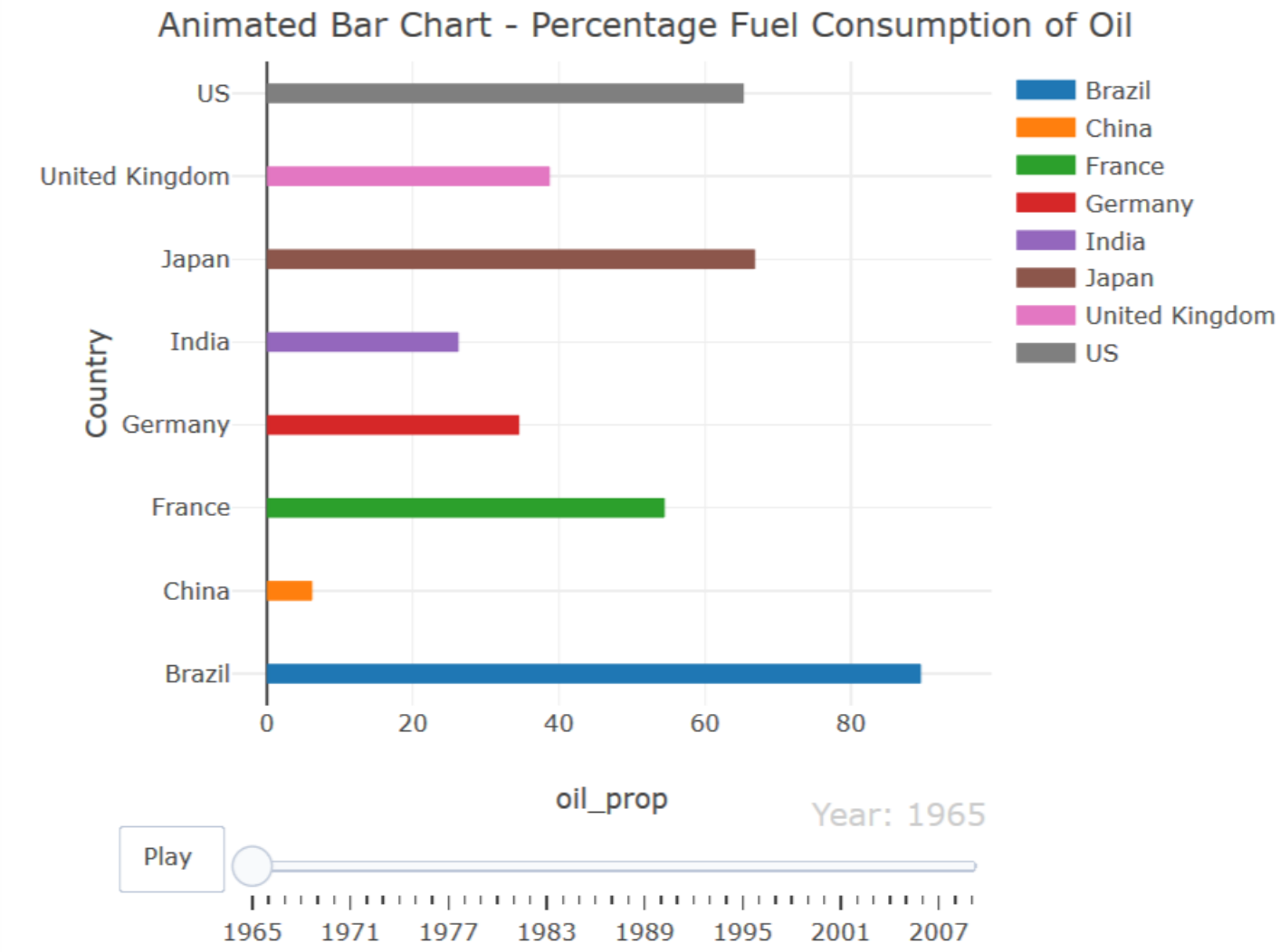
```
oc_data = read.csv2("Oilcoal.csv")

x <- 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)")

knitr::include_graphics('./6.1.1.png')
```



Bar Chart

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)

x <- 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")

knitr::include_graphics('./6.2.1.png')
```



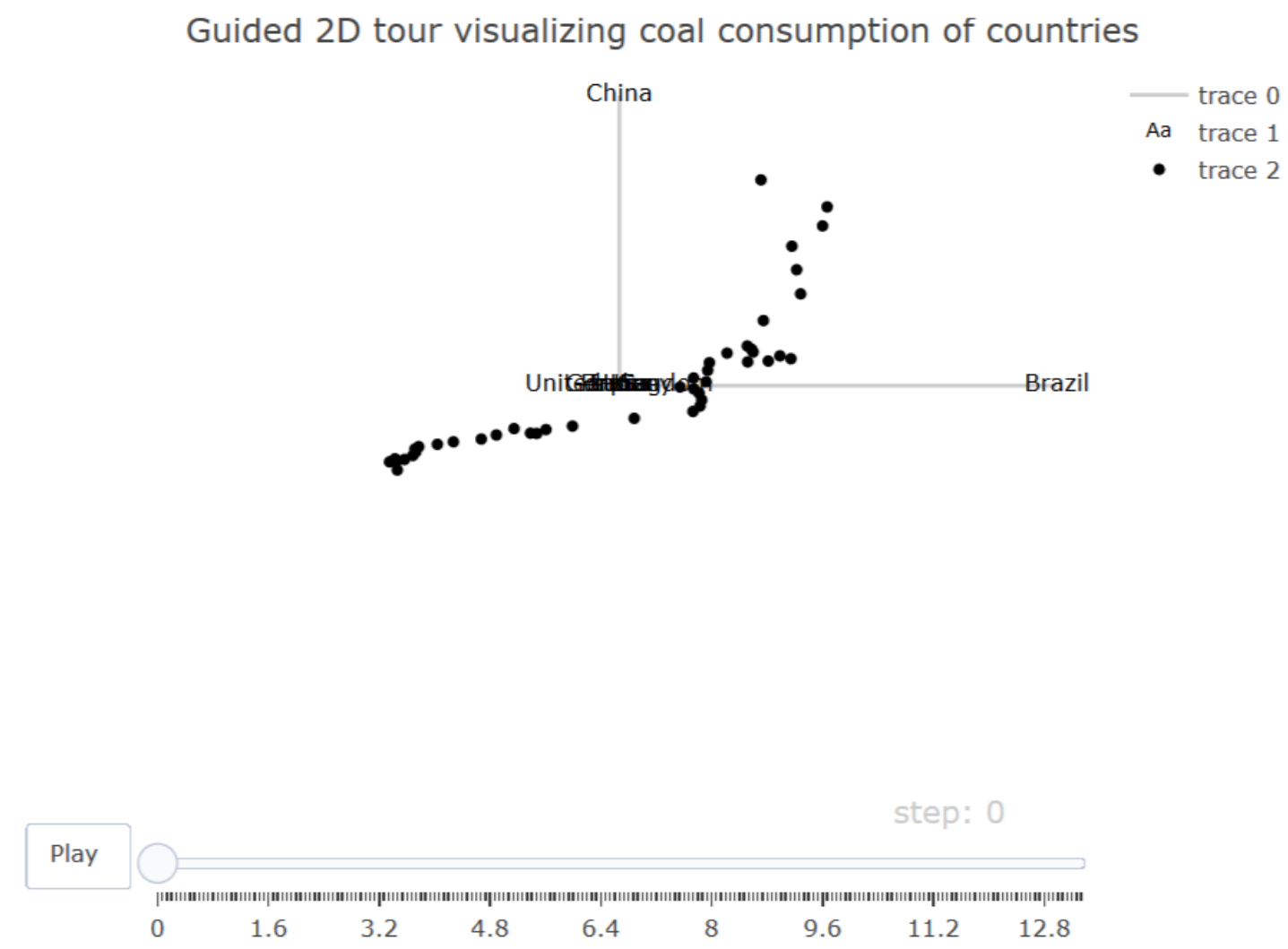
Tour and Projection

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
```

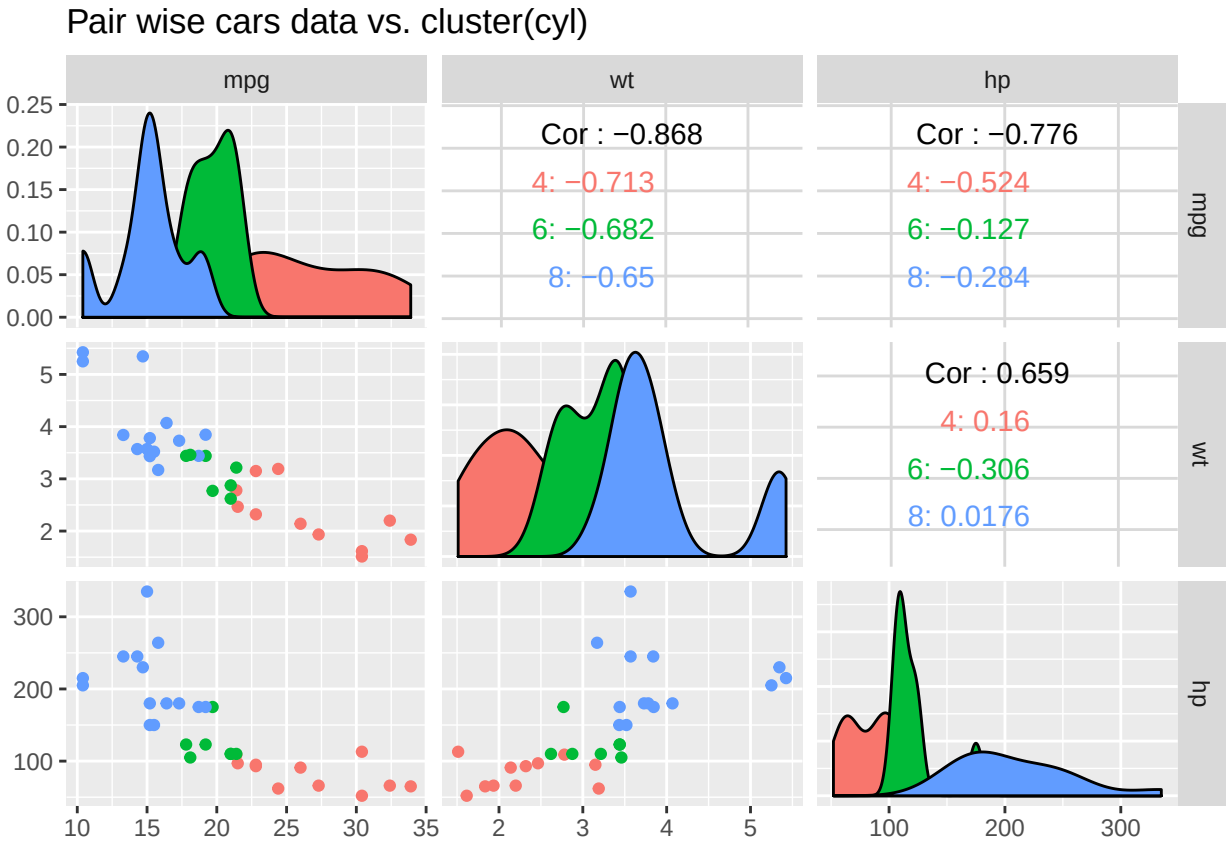
```
x <- proj_dat %>%
  plot_ly(x = ~x, y = ~y, frame = ~step, color = I("black")) %>%
  add_segments(xend = 0, yend = 0, color = I("gray80")) %>%
  add_text(text = ~variable) %>%
  add_markers(data = tour_dat, text = ~state, ids = ~state, hoverinfo = "text") %>%
  layout(xaxis = ax, yaxis = ax, title = "Guided 2D tour visualizing coal consumption of countries")

knitr::include_graphics('./7.1.1.png')
```



Multiple Plots

QC Scatter Matrix Plots using GGally library

```
ggpairs(mtcars, columns = c("mpg", "wt", "hp"),
        title = "Pair wise cars data vs. cluster(cyl)",
        ggplot2::aes(colour=as.factor(cyl)))
```

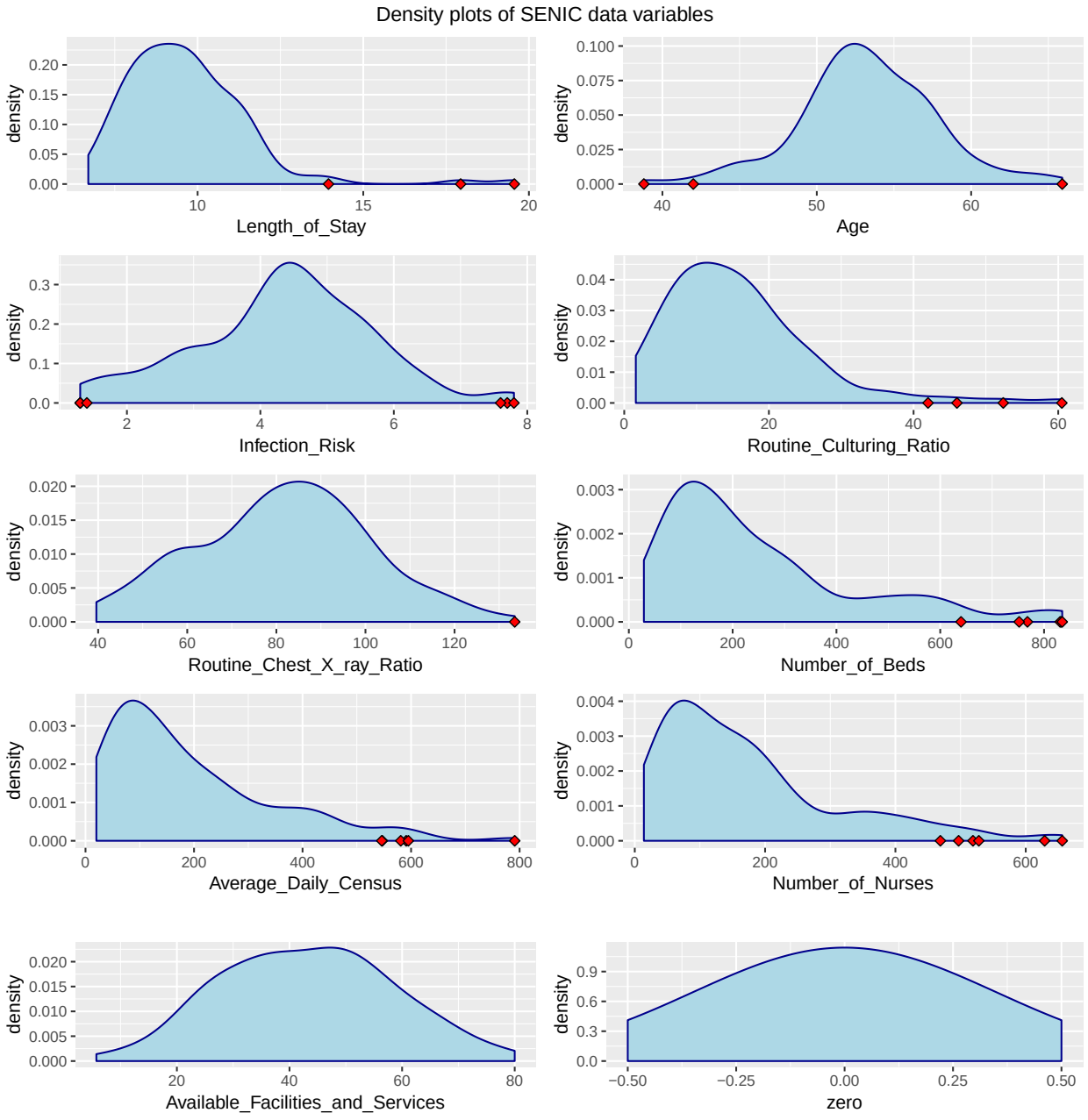


Density Plots

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")
```



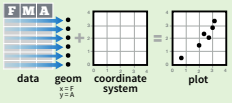
GGplot2 cheat sheet

Data Visualization with ggplot2 Cheat Sheet

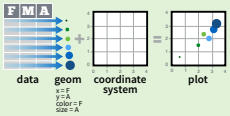
RStudio

Basics

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



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point"**)**
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, **aes(**x = cty, y = hwy**))**
Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

```
ggplot(mpg, aes(hwy, cty)) +
 geom_point(aes(color = cyl)) +
 geom_smooth(method ="lm") +
 coord_cartesian() +
 scale_color_gradient() +
 theme_bw()
```

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

Add a new layer to a plot with a **geom_*()** or **stat_*()** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

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

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

a + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

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

a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

a + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

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

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

Graphical Primitives

c <- ggplot(map, aes(long, lat))

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

d <- ggplot(economics, aes(date, unemploy))

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

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

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat)**)**
x, xend, y, yend, alpha, color, linetype, size

e + geom_rect(aes(xmin = long, ymin = lat, xmax= long + delta_long, ymax = lat + delta_lat)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

Two Variables

Continuous X, Continuous Y

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

f + geom_blank()

f + geom_jitter()
x, y, alpha, color, fill, shape, size

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

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

f + geom_rug(sides = "bl"**)**
alpha, color, linetype, size

f + geom_smooth(model = lm**)**
x, y, alpha, color, fill, linetype, size, weight

f + geom_text(aes(label = cty)**)**
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Discrete X, Continuous Y

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

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

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

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

g + geom_violin(scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + geom_bin2d(binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + geom_density2d()
x, y, alpha, colour, linetype, size

i + geom_hex()
x, y, alpha, colour, fill size

Continuous Function

j <- ggplot(economics, aes(date, unemploy))

j + geom_area()
x, y, alpha, color, fill, linetype, size

j + geom_line()
x, y, alpha, color, linetype, size

j + geom_step(direction = "hv"**)**
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

l + 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))
m <- ggplot(seals, aes(long, lat))

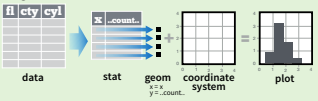
m + geom_contour(aes(z = z)**)**
x, y, z, alpha, colour, linetype, size, weight

m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

m + geom_tile(aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size

Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. a + geom_bar(stat = "bin")



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. stat_bin(geom="bar") does the same as geom_bar(stat="bin")

stat function | layer specific mappings | variable created by transformation

i + **stat_density2d**(aes(fill = ..level..), geom = "polygon", n = 100)

geom for layer | parameters for stat

1D distributions

a + **stat_bin**(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + **stat_bindot**(binwidth = 1, binaxis = "x")
x, y, | ..count.., ..ncount..
a + **stat_density**(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

2D distributions

f + **stat_bin2d**(bins = 30, drop = TRUE)
x, y, fill | ..count.., ..density..
f + **stat_binhex**(bins = 30)
x, y, fill | ..count.., ..density..
f + **stat_density2d**(contour = TRUE, n = 100)
x, y, color, size | ..level..

3 Variables

m + **stat_contour**(aes(z = z))
x, y, z, order | ..level..
m+ **stat_spoke**(aes(radius= z, angle = z))
angle, radius, x, xend, y, yend | ..x.., ..xend.., ..y.., ..yend..
m + **stat_summary_hex**(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..
m + **stat_summary2d**(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

Comparisons

g + **stat_boxplot**(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..outliers..
g + **stat_ydensity**(adjust = 1, kernel = "gaussian", scale = "area")
x, y | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

Functions

f + **stat_ecdf**(n = 40)
x, y | ..x.., ..y..
f + **stat_quantile**(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")
x, y | ..quantile.., ..x.., ..y..
f + **stat_smooth**(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

General Purpose

ggplot() + **stat_function**(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5))
x | ..y..
f + **stat_identity**()
ggplot() + **stat_qq**(aes(sample=1:100), distribution = qt, dparams = list(df=5))
sample, x, y | ..x.., ..y..
f + **stat_sum**()
x, y, size | ..size..
f + **stat_summary**(fun.data = "mean_cl_boot")
f + **stat_unique**()

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.

n <- b + **geom_bar**(aes(fill = fl))

n

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 any aesthetic:
alpha, color, fill, linetype, shape, size

scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values **as** visual values
scale_*_manual(values = c()) - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")) - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
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 <- b + **geom_bar**(aes(fill = fl))

n + **scale_fill_brewer**(palette = "Blues")
For palette choices: library(RColorBrewer) display.brewer.all()

n + **scale_fill_grey**(start = 0.2, end = 0.8, na.value = "red")

Continuous

o <- a + **geom_dotplot**(aes(fill = ..x..))

o + **scale_fill_gradient**(low = "red", high = "yellow")

o + **scale_fill_gradient2**(low = "red", hight = "blue", mid = "white", midpoint = 25)

o + **scale_fill_gradientn**(colours = terrain.colors(6))
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

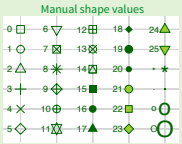
Shape scales

p <- f + **geom_point**(aes(shape = fl))

p + **scale_shape**(solid = FALSE)

p + **scale_shape_manual**(values = c(3:7))
Shape values shown in chart on right

Manual shape values



Size scales

q <- f + **geom_point**(aes(size = cyl))

q + **scale_size_area**(max = 6)
Value mapped to area of circle (not radius)

Coordinate Systems

r <- b + 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 extras and strains to the name of a window function.

z + **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**(. ~ x, **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 ^ .(x)))

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

s + **geom_bar**(**position = "stack"**)
Stack elements on top of one another

f + **geom_point**(**position = "jitter"**)
Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual **width** and **height** arguments

s + **geom_bar**(**position = position_dodge(width = 1)**)

Labels

t + **ggtitle**("New Plot Title")
Add a main title above the plot
t + **xlab**("New X label")
Change the label on the X axis
t + **ylab**("New Y label")
Change the label on the Y axis
t + **labs**(title =" New title", x = "New x", y = "New y")
All of the above

Use scale functions to update legend labels

Legends

t + **theme**(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"
t + **guides**(color = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)
t + **scale_fill_discrete**(name = "Title", labels = c("A", "B", "C"))
Set legend title and labels with a scale function.

Themes

r + **theme_bw**()
White background with grid lines

r + **theme_classic**()
White background no gridlines

r + **theme_grey**()
Grey background (default theme)

r + **theme_minimal**()
Minimal theme

ggthemes - Package with additional ggplot2 themes

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))

Plotly cheat sheet

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' )
```

Scatter Plots

```
plot_ly (
  x = c( 1, 2, 3 ),
  y = c( 5, 6, 7 ),
  type = 'scatter' ,
  mode = 'markers' )
```

Bar Charts

```
plot_ly (
  x = c( 1, 2, 3),
  y = c( 5, 6, 7),
  type = 'bar' ,
  mode = 'markers' )
```

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' )))
```

Heatmaps

```
plot_ly (
   z = volcano ,
   type = 'heatmap' )
```

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 Cheatsheet

Interactive Web Apps with shiny Cheat Sheet

learn more at shiny.rstudio.com

RStudio

Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)

Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

App template

Begin writing a new app with this template. Preview the app by running the code at the R command line.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output){}
shinyApp(ui = ui, server = server)
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines `ui` and `server` into a functioning app. Wrap with **runApp()** if calling from a sourced script or inside a function.

Share your app

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at **http://shinyapps.io**
2. Click the **Publish** icon in the RStudio IDE (>=0.99) or run: **rsconnect::deployApp**("<path to directory>")

Build or purchase your own Shiny Server

at www.rstudio.com/products/shiny-server/

Building an App - Complete the template by adding arguments to fluidPage() and a body to the server function.

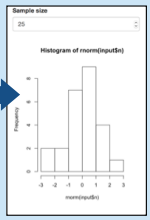
Add inputs to the UI with ***Input()** functions

Add outputs with ***Output()** functions

Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with **output$<id>**
2. Refer to inputs with **input$<id>**
3. Wrap code in a **render*()** function before saving to output

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

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



Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
```

```
# server.R
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

ui.R contains everything you would save to ui.

server.R ends with the function you would save to server.

No need to call **shinyApp()**.

Save each app as a directory that contains an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

- app-name
  - app.R — The directory name is the name of the app
  - global.R — (optional) defines objects available to both ui.R and server.R
  - DESCRIPTION — (optional) used in showcase mode
  - README — (optional) data, scripts, etc.
  - <other files> — (optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"

Launch apps with `runApp(<path to directory>)`

Outputs - render*() and *Output() functions work together to add R output to the UI

	works with	
DT::**renderDataTable**(expr, options, callback, escape, env, quoted)		**dataTableOutput**(outputId, icon, …)
renderImage(expr, env, quoted, deleteFile)		**imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
renderPlot(expr, width, height, res, …, env, quoted, func)		**plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)
renderPrint(expr, env, quoted, func, width)		**verbatimTextOutput**(outputId)
renderTable(expr,…, env, quoted, func)		**tableOutput**(outputId)
renderText(expr, env, quoted, func)		**textOutput**(outputId, container, inline)
renderUI(expr, env, quoted, func)		**uiOutput**(outputId, inline, container, …) & **htmlOutput**(outputId, inline, container, …)

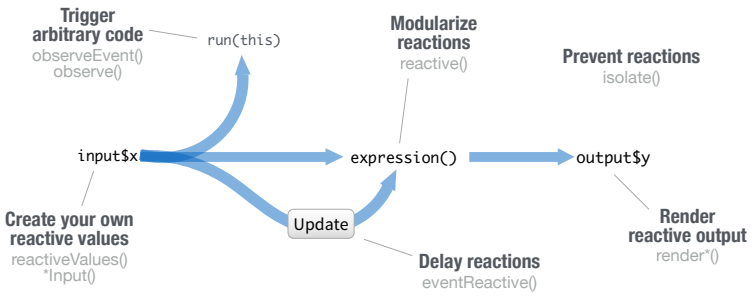
Inputs - collect values from the user

Access the current value of an input object with **input $<inputId>**. Input values are **reactive**.

- Action — **actionButton**(inputId, label, icon, …)
- Link — **actionLink**(inputId, label, icon, …)
- Choice 1, Choice 2, Choice 3 — **checkboxGroupInput**(inputId, label, choices, selected, inline)
- Check me — **checkboxInput**(inputId, label, value)
- **dateInput**(inputId, label, value, min, max, format, startview, weekstart, language)
- **dateRangeInput**(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)
- Choose File — **fileInput**(inputId, label, multiple, accept)
- 1 — **numericInput**(inputId, label, value, min, max, step)
- ••••••• — **passwordInput**(inputId, label, value)
- Choice A, Choice B, Choice C — **radioButtons**(inputId, label, choices, selected, inline)
- Choice 1, Choice 1, Choice 2 — **selectInput**(inputId, label, choices, selected, multiple, selectize, width, size) (also **selectizeInput()**)
- 0, 5, 10 — **sliderInput**(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)
- Apply Changes — **submitButton**(text, icon) (Prevents reactions across entire app)
- Enter text — **textInput**(inputId, label, value)

Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error `Operation not allowed without an active reactive context.`



Create your own reactive values

```
library(shiny)

ui <- fluidPage(
  textInput("a","")
)

server <-
function(input,output){
  rv <- reactiveValues()
  rv$number <- 5
}

shinyApp(ui, server)
```

***Input() functions**
(see front page)

reactiveValues(...)

Each input function creates a reactive value stored as **input$<inputId>**

reactiveValues() creates a list of reactive values whose values you can set.

Render reactive output

```
library(shiny)
ui <- fluidPage(
  textInput("a","")
)

server <-
function(input,output){
  output$b <-
    renderText({
      input$a
    })
}

shinyApp(ui, server)
```

render*() functions
(see front page)

Builds an object to display. Will rerun code in body to rebuild the object whenever a reactive value in the code changes.

Save the results to **output$<outputId>**

Prevent reactions

```
library(shiny)
ui <- fluidPage(
  textInput("a",""),
  textOutput("b")
)

server <-
function(input,output){
  output$b <-
    renderText({
      isolate({input$a})
    })
}

shinyApp(ui, server)
```

isolate(expr**)**

Runs a code block. Returns a **non-reactive** copy of the results.

Trigger arbitrary code

```
library(shiny)
ui <- fluidPage(
  textInput("a",""),
  actionButton("go", "")
)

server <-
function(input,output){
  observeEvent(input$go,
    print(input$a)
  )
}

shinyApp(ui, server)
```

observeEvent(eventExpr, handlerExpr, event.env, event.quoted, handler.env, handler.quoted, labe, suspended, priority, domain, autoDestroy, ignoreNULL**)**

Runs code in 2nd argument when reactive values in 1st argument change. See **observe()** for alternative.

Modularize reactions

```
library(shiny)
ui <- fluidPage(
  textInput("a",""),
  textInput("z", "")
)

server <-
function(input,output){
  re <- reactive({
    paste(input$a,input$b)
  output$b <- renderText({
    re()
  })
}

shinyApp(ui, server)
```

reactive(x, env, quoted, label, domain**)**

Creates a **reactive expression** that

- **caches** its value to reduce computation
- can be called by other code
- notifies its dependencies when it ha been invalidated

Call the expression with function syntax, e.g. `re()`

Delay reactions

```
library(shiny)
ui <- fluidPage(
  textInput("a",""),
  actionButton("go", "")
)

server <-
function(input,output){
  re <- eventReactive(
    input$go,{input$a})
  output$b <- renderText({
    re()
  })
}

shinyApp(ui, server)
```

eventReactive(eventExpr, valueExpr, event.env, event.quoted, value.env, value.quoted, label, domain, ignoreNULL**)**

Creates reactive expression with code in 2nd argument that only invalidates when reactive values in 1st argument change.

UI

An app's UI is an HTML document. Use Shiny's functions to assemble this HTML with R.

```
fluidPage(
  textInput("a","")
)
## <div class="container-fluid">
##   <div class="form-group shiny-input-container">
##     <label for="a"></label>
##     <input id="a" type="text"
##       class="form-control" value=""/>
##   </div>
## </div>
```

Returns HTML

Add static HTML elements with **tags**, a list of functions that parallel common HTML tags, e.g. **tags$a()**. Unnamed arguments will be passed into the tag; named arguments will become tag attributes.

tags$a	tags$data	tags$h6	tags$nav	tags$span
tags$abbr	tags$datalist	tags$head	tags$noscript	tags$strong
tags$address	tags$dd	tags$header	tags$object	tags$style
tags$area	tags$del	tags$hgroup	tags$ol	tags$sub
tags$article	tags$details	tags$hr	tags$optgroup	tags$summary
tags$aside	tags$dfn	tags$HTML	tags$option	tags$sup
tags$audio	tags$div	tags$i	tags$output	tags$table
tags$b	tags$dl	tags$iframe	tags$p	tags$tbody
tags$base	tags$dt	tags$img	tags$param	tags$td
tags$bdi	tags$em	tags$input	tags$pre	tags$textarea
tags$bdo	tags$embed	tags$ins	tags$progress	tags$tfoot
tags$blockquote	tags$eventsource	tags$kbd	tags$q	tags$th
tags$body	tags$fieldset	tags$keygen	tags$ruby	tags$thead
tags$br	tags$figcaption	tags$label	tags$rp	tags$time
tags$button	tags$figure	tags$legend	tags$rt	tags$title
tags$canvas	tags$footer	tags$li	tags$s	tags$tr
tags$caption	tags$form	tags$link	tags$samp	tags$track
tags$cite	tags$h1	tags$mark	tags$script	tags$u
tags$code	tags$h2	tags$map	tags$section	tags$ul
tags$col	tags$h3	tags$menu	tags$select	tags$var
tags$colgroup	tags$h4	tags$meta	tags$small	tags$video
tags$command	tags$h5	tags$meter	tags$source	tags$wbr

The most common tags have wrapper functions. You do not need to prefix their names with **tags$**

```
ui <- fluidPage(
  h1("Header 1"),
  hr(),
  br(),
  p(strong("bold")),
  p(em("italic")),
  p(code("code")),
  a(href="", "link"),
  HTML("<p>Raw html</p>")
)
```

To include a CSS file, use **includeCSS()**, or
1. Place the file in the **www** subdirectory
2. Link to it with

```
tags$head(tags$link(rel = "stylesheet",
  type = "text/css", href = "<file name>"))
```

To include JavaScript, use **includeScript()** or
1. Place the file in the **www** subdirectory
2. Link to it with

```
tags$head(tags$script(src = "<file name>"))
```

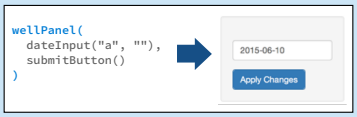
IMAGES

To include an image
1. Place the file in the **www** subdirectory
2. Link to it with `img(src="<file name>")`

Layouts

Combine multiple elements into a "single element" that has its own properties with a panel function, e.g.

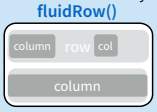
```
wellPanel(
  dateInput("a", ""),
  submitButton()
)
```



absolutePanel()	inputPanel()	tabPanel()
conditionalPanel()	mainPanel()	tabsetPanel()
fixedPanel()	navlistPanel()	titlePanel()
headerPanel()	sidebarPanel()	wellPanel()

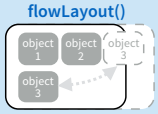
Organize panels and elements into a layout with a layout function. Add elements as arguments of the layout functions.

fluidRow()



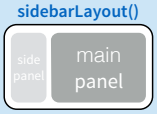

```
ui <- fluidPage(
  fluidRow(column(width = 4),
    column(width = 2, offset = 3)),
  fluidRow(column(width = 12))
)
```

flowLayout()



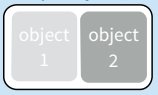

```
ui <- fluidPage(
  flowLayout( # object 1,
              # object 2,
              # object 3
  )
)
```

sidebarLayout()



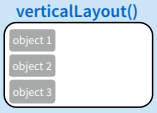

```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(),
    mainPanel()
  )
)
```

splitLayout()




```
ui <- fluidPage(
  splitLayout( # object 1,
               # object 2
  )
)
```

verticalLayout()




```
ui <- fluidPage(
  verticalLayout( # object 1,
                  # object 2,
                  # object 3
  )
)
```

Layer tabPanels on top of each other, and navigate between them, with:

```
ui <- fluidPage( tabsetPanel(
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  tabPanel("tab 3", "contents")))
```


```
ui <- fluidPage( navlistPanel(
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  tabPanel("tab 3", "contents")))
```


```
ui <- navbarPage(title = "Page",
  tabPanel("tab 1", "contents"),
  tabPanel("tab 2", "contents"),
  tabPanel("tab 3", "contents"))
```

Tidyr and Dplyr cheatsheet

Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio

Tidy Data - A foundation for wrangling in R

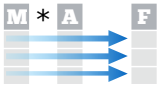
In a tidy data set:

Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own **row**

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

Converts data to tbl class. tbl's are easier to examine than data frames. R displays only the data that fits onscreen:

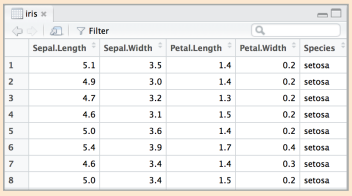
```
Source: local data frame [150 x 5]

   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
..          ...         ...          ...
Variables not shown: Petal.Width (dbl),
  Species (fctr)
```

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

View data set in spreadsheet-like display (note capital V).



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

dplyr::**%>%**

Passes object on left hand side as first argument (or . argument) of function on righthand side.

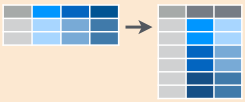
`x %>% f(y)` *is the same as* `f(x, y)`

`y %>% f(x, ., z)` *is the same as* `f(x, y, z )`

"Piping" with %>% makes code more readable, e.g.

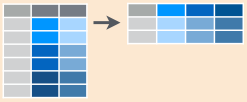
```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Reshaping Data - Change the layout of a data set



tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.



tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

dplyr::**data_frame(a = 1:3, b = 4:6)**

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

Order rows by values of a column (low to high).

dplyr::**arrange(mtcars, desc(mpg))**

Order rows by values of a column (high to low).

dplyr::**rename(tb, y = year)**

Rename the columns of a data frame.

Subset Observations (Rows)

dplyr::**filter(iris, Sepal.Length > 7)**

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

dplyr::**sample_frac(iris, 0.5, replace = TRUE)**

Randomly select fraction of rows.

dplyr::**sample_n(iris, 10, replace = TRUE)**

Randomly select n rows.

dplyr::**slice(iris, 10:15)**

Select rows by position.

dplyr::**top_n(storms, 2, date)**

Select and order top n entries (by group if grouped data).

Logic in R - ?Comparison, ?base::Logic

<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

Subset Variables (Columns)

dplyr::**select(iris, Sepal.Width, Petal.Length, Species)**

Select columns by name or helper function.

Helper functions for select - ?select

select(iris, contains("."))
Select columns whose name contains a character string.

select(iris, ends_with("Length"))
Select columns whose name ends with a character string.

select(iris, everything())
Select every column.

select(iris, matches(".t."))
Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))
Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))
Select columns whose names are in a group of names.

select(iris, starts_with("Sepal"))
Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)
Select all columns except Species.

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Summarise Data

dplyr::**summarise(iris, avg = mean(Sepal.Length))**
Summarise data into single row of values.
dplyr::**summarise_each(iris, funs(mean))**
Apply summary function to each column.
dplyr::**count(iris, Species, wt = Sepal.Length)**
Count number of rows with each unique value of variable (with or without weights).

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

dplyr::**first**
First value of a vector.
dplyr::**last**
Last value of a vector.
dplyr::**nth**
Nth value of a vector.
dplyr::**n**
# of values in a vector.
dplyr::**n_distinct**
# of distinct values in a vector.
IQR
IQR of a vector.
min
Minimum value in a vector.
max
Maximum value in a vector.
mean
Mean value of a vector.
median
Median value of a vector.
var
Variance of a vector.
sd
Standard deviation of a vector.

Group Data

dplyr::**group_by(iris, Species)**
Group data into rows with the same value of Species.
dplyr::**ungroup(iris)**
Remove grouping information from data frame.

iris %>% group_by(Species) %>% summarise(…)
Compute separate summary row for each group.

Make New Variables

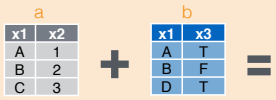
dplyr::**mutate(iris, sepal = Sepal.Length + Sepal. Width)**
Compute and append one or more new columns.
dplyr::**mutate_each(iris, funs(min_rank))**
Apply window function to each column.
dplyr::**transmute(iris, sepal = Sepal.Length + Sepal. Width)**
Compute one or more new columns. Drop original columns.

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

dplyr::**lead**
Copy with values shifted by 1.
dplyr::**lag**
Copy with values lagged by 1.
dplyr::**dense_rank**
Ranks with no gaps.
dplyr::**min_rank**
Ranks. Ties get min rank.
dplyr::**percent_rank**
Ranks rescaled to [0, 1].
dplyr::**row_number**
Ranks. Ties got to first value.
dplyr::**ntile**
Bin vector into n buckets.
dplyr::**between**
Are values between a and b?
dplyr::**cume_dist**
Cumulative distribution.
dplyr::**cumall**
Cumulative `all`
dplyr::**cumany**
Cumulative `any`
dplyr::**cummean**
Cumulative `mean`
cumsum
Cumulative `sum`
cummax
Cumulative `max`
cummin
Cumulative `min`
cumprod
Cumulative `prod`
pmax
Element-wise `max`
pmin
Element-wise `min`

iris %>% group_by(Species) %>% mutate(…)
Compute new variables by group.

Combine Data Sets



Mutating Joins

dplyr::**left_join(a, b, by = "x1")**
Join matching rows from b to a.

dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.

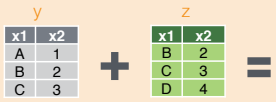
dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

dplyr::**full_join(a, b, by = "x1")**
Join data. Retain all values, all rows.

Filtering Joins

dplyr::**semi_join(a, b, by = "x1")**
All rows in a that have a match in b.

dplyr::**anti_join(a, b, by = "x1")**
All rows in a that do not have a match in b.



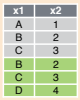
Set Operations

dplyr::**intersect(y, z)**
Rows that appear in both y and z.

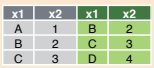
dplyr::**union(y, z)**
Rows that appear in either or both y and z.

dplyr::**setdiff(y, z)**
Rows that appear in y but not z.

Binding



dplyr::**bind_rows(y, z)**
Append z to y as new rows.



dplyr::**bind_cols(y, z)**
Append z to y as new columns.
Caution: matches rows by position.

devtools::install_github("rstudio/EDAWR") for data sets

Learn more with browseVignettes(package = c("dplyr", "tidyr")) • dplyr 0.4.0• tidyr 0.2.0 • Updated: 1/15

Demo of plotly

```
#Demo(package="plotly")
```

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=TRUE),
  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"))
```

Shiny Oleg examples

```
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)
```

Shiny adding graph example

```
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()
   })
}
```

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

Shiny widget

```r
library(shiny)

# Define UI for dataset viewer app ----
ui <- fluidPage(

  # App title ----
  titlePanel("More Widgets"),

  # Sidebar layout with input and output definitions ----
  sidebarLayout(

    # Sidebar panel for inputs ----
    sidebarPanel(

      # Input: Select a dataset ----
      selectInput("dataset", "Choose a dataset:",
                  choices = c("rock", "pressure", "cars")),

      # Input: Specify the number of observations to view ----
      numericInput("obs", "Number of observations to view:", 10),

      # Include clarifying text ----
      helpText("Note: while the data view will show only the specified",
               "number of observations, the summary will still be based",
               "on the full dataset."),

      # Input: actionButton() to defer the rendering of output ----
      # until the user explicitly clicks the button (rather than
      # doing it immediately when inputs change). This is useful if
      # the computations required to render output are inordinately
      # time-consuming.
      actionButton("update", "Update View")

    ),

    # Main panel for displaying outputs ----
    mainPanel(

      # Output: Header + summary of distribution ----
      h4("Summary"),
      verbatimTextOutput("summary"),

      # Output: Header + table of distribution ----
      h4("Observations"),
      tableOutput("view")
    )
```

```
  )
)

# Define server logic to summarize and view selected dataset ----
server <- function(input, output) {

  # Return the requested dataset ----
  # Note that we use eventReactive() here, which depends on
  # input$update (the action button), so that the output is only
  # updated when the user clicks the button
  datasetInput <- eventReactive(input$update, {
    switch(input$dataset,
           "rock" = rock,
           "pressure" = pressure,
           "cars" = cars)
  }, ignoreNULL = FALSE)

  # Generate a summary of the dataset ----
  output$summary <- renderPrint({
    dataset <- datasetInput()
    summary(dataset)
  })

  # Show the first "n" observations ----
  # The use of isolate() is necessary because we don't want the table
  # to update whenever input$obs changes (only when the user clicks
  # the action button)
  output$view <- renderTable({
    head(datasetInput(), n = isolate(input$obs))
  })

}

# Create Shiny app ----
shinyApp(ui, server)
```

Good header for knitr

```
# include this to add pdf to your document

# title: '732A98: Visualization - Cheatbook'
# author: "Anubhav Dikshit (anudi287)"
# date: "02 Nov 2018"
# output:
#   pdf_document:
#     fig_caption: yes
#     toc: yes
#   html_document:
#     number_sections: yes
#     theme: readable
#     toc: yes
```

```
# always_allow_html: yes
# header-includes:
#   - \usepackage{booktabs}
#   - \usepackage{sectsty} \sectionfont{\centering}
#   - \renewcommand{\contentsname}{}\vspace{-2cm}
#   - \usepackage{pdfpages}

#\includepdf[pages={-}]{oleg_lectures_combined.pdf}
```

Appendix Code

```
# ```{r, ref.label=knitr::all_labels(), echo=TRUE, eval=FALSE}
# ```
```

Oleg Lectures Compilation

Lecture Notes

Interaction Operators

- **Navigation:** user controls for altering the position of the camera and for scaling the view (what gets mapped to the screen) such as panning, rotating, and zooming.
- **Selection:** user controls for identifying an object, a collection of objects, or regions of interest to be the subject of some operation, such as highlighting, deleting, and modifying.
- **Filtering:** User controls for reducing the size of the data being mapped to the screen, either by eliminating records, dimensions, or both.
- **Reconfiguring:** User controls for changing the way data is mapped to graphical entities or attributes, such as reordering the data or layouts, thereby providing a different way of viewing a data subset.
- **Encoding:** User controls for changing the graphical attributes, such as point size or line color, to potentially reveal different features.
- **Connecting:** User controls for linking different views or objects to show related items.
- **Abstracting/elaborating**: User controls for modifying the level of detail.

Interaction Operands

1. **Screen Space(Pixels):** Screen space consists of the pixels of the display
   - **Navigation**: Navigation in screen space typically consists of actions such as panning, zooming, and rotation. Note that in each case, no new data is used;
   - **Selection**: The selection can be performed on individual pixels, rectangles or circles of pixels, or on arbitrarily shaped regions that the user specifies.
   - **Abstraction**: Distortion of image (fisheye)

   - **Filtering**: Removing some pixels
2. **Data Value Space (Multivariate Data Values):** Operations on data space are applied directly to the data, rather than to the screen.
   - **Navigation**: Using the data values as a mechanism for view specification. The analogous operations for panning and zooming would be to change the data values being displayed; panning

would shift the start of the value range to be shown, while zooming would decrease the size of this range.
    - **Selection**: This can be performed via direct manipulation, as in the data-driven brushing reported in or via sliders or other query specification mechanisms Selection may involve a single value, or one or more ranges of values.
    - **Filtering**: Sample the data, sample dimensions
    - **Reconfiguring**: Sorting observations, dimensions, nonlinear transformations.
3. **Data structure space**: Data can be structured in a number of ways, such as lists, tables, grids,hierarchies, and graphs. For each structure, one can develop interaction mechanisms to indicate what portions of the structure will be manipulated, and how this manipulation will be manifested.
    - **Navigation**:Navigation in data structurespace involves moving the view specification along the structure, as in showing sequential groups of records, or moving down or up a hierarchical structure
    - **Selection**: Selection in data structure space generally involves displaying the structure and allowing the user to identify regions of interest within it
    - **Filtering**: Filtering is often performed in data structure space to reduce the amount of information on the display.
    - **Abstraction/Elaboration**: histogram with zooming - recompute bars?
4. **Attribute Space (Components of Graphical Entities)**: In attribute space, operators are focused on one or more of the attributes associated with the graphical entity being used to convey information. Such attributes could include color, size, shape, or any other of the eight visual variables.
    - **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)
5. **Visualization Structure Space**: A visualization consists of a structure that is relatively independent of the values, attributes, and structure of data.
    - **Navigation**: Navigation in visualization structure space might include moving through pages in a spreadsheet-style visualization tool or zooming in on an individual plot in a scatterplot matrix.
    - **Selection**: For selection, typical operations would include choosing components to hide, move, or rearrange. For example, one might select an axis in parallel coordinates and drag it to a new location to discover different relationships among the data dimensions.

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)

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 features

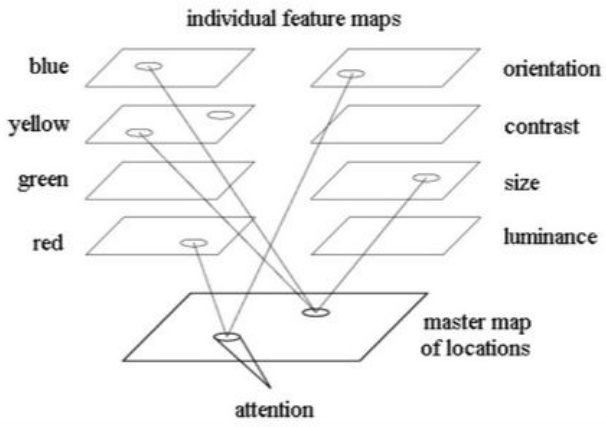
- Length
- Width
- Size
- Curvature (shape)
- Hue
- Intensity
- Flicker
- Direction of motion
- 3D depth
- Lighting direction

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



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
  - Level of details

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

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

Analysis:

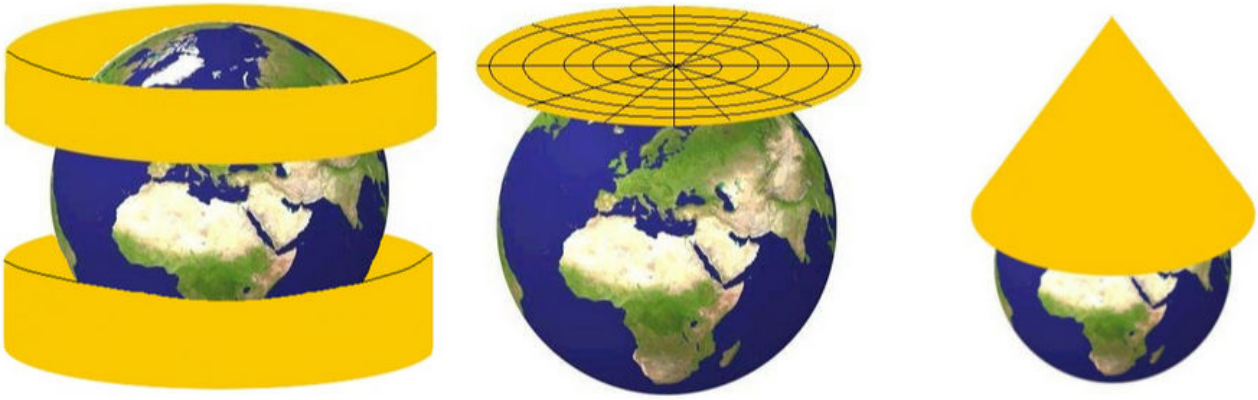
- Direction (if monotonic, decreasing or increasing; if not monotonic, which parts increasing, which decreasing)
- Density (dense areas, sparse areas)
- Outliers
- Clusters

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)
  - …

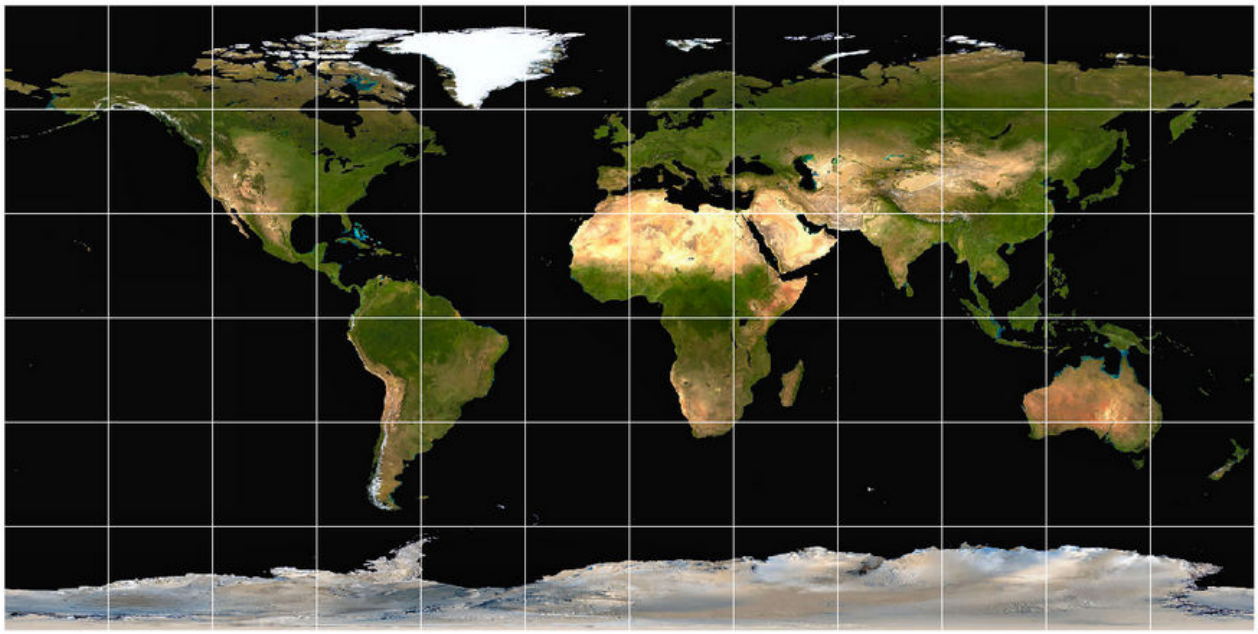
What is map?

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



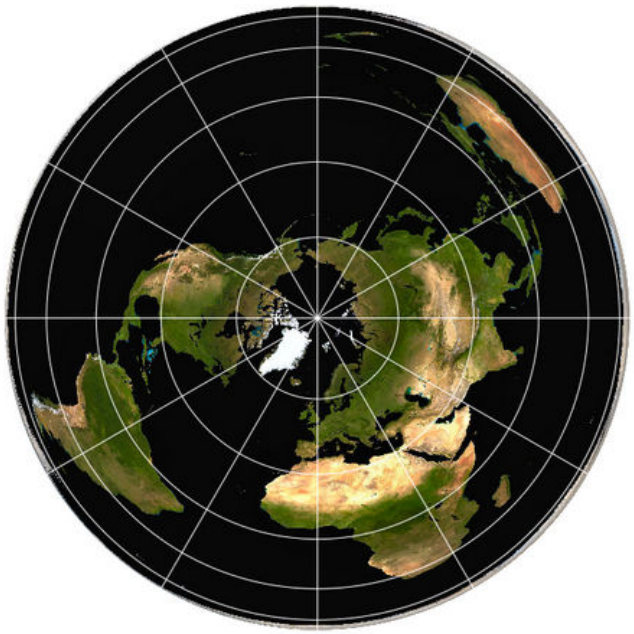
Cylindrical projection

- 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



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

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

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

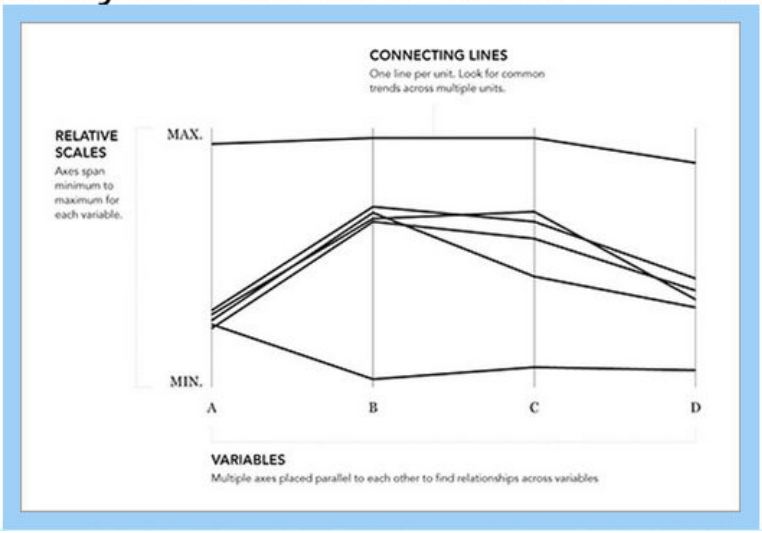
- A few countries available through plotly
- Downloading map of a country:
  - Finding a country map http://gadm.org/
  - Decide what level of detalization is needed (region, county,…)
  - Download R(sf) file.
  - Load the file to R using readRDS function
    - e.g. rds <- readRDS('filename.rds')
  -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

Parallel coordinates

Construction:

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

Analysis: - Clusters - Outliers - Correlated variables



Parallel coordinates

Analysis

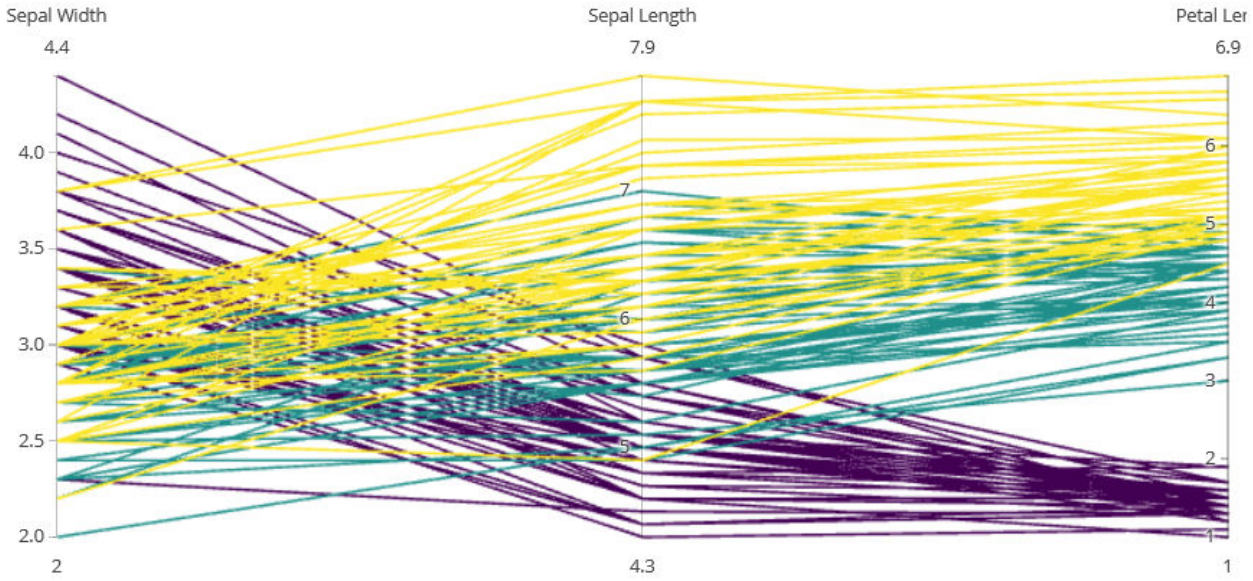
- Positive correlation between two adjacent variables: almost all segments are parallel to each other
- Clusters in some variable space: several trace lines that are near each other and have similar pattern
- Outliers: trace lines that have unusual pattern and/or fall out outside the common plot area

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

Parallel coordinates

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



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

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)

Radar charts

If juxtaposed, analyse:

- Clusters
- Outliers
- Outlying directions

If superimposed,

- Comparing variable length
- Seeing similar and outlying observations

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

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)
- …

Interaction operators

Navigation operator:

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

Selection operators

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

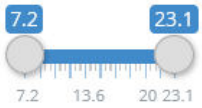
Connection operators

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

Filtering operators

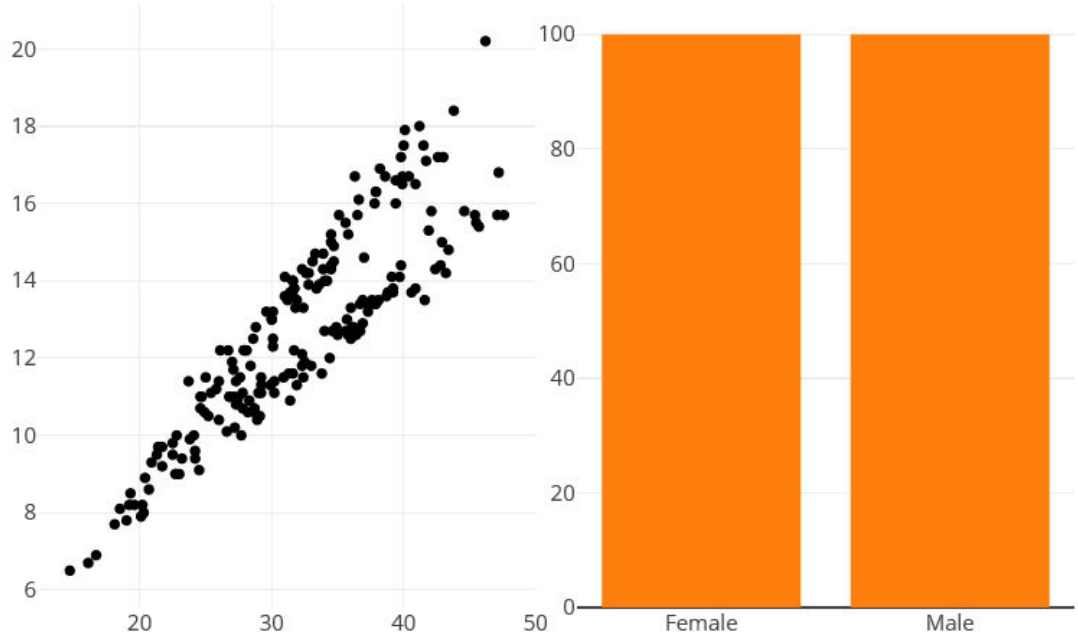
- Reducing data acc. to specifications

Frontal Lobe



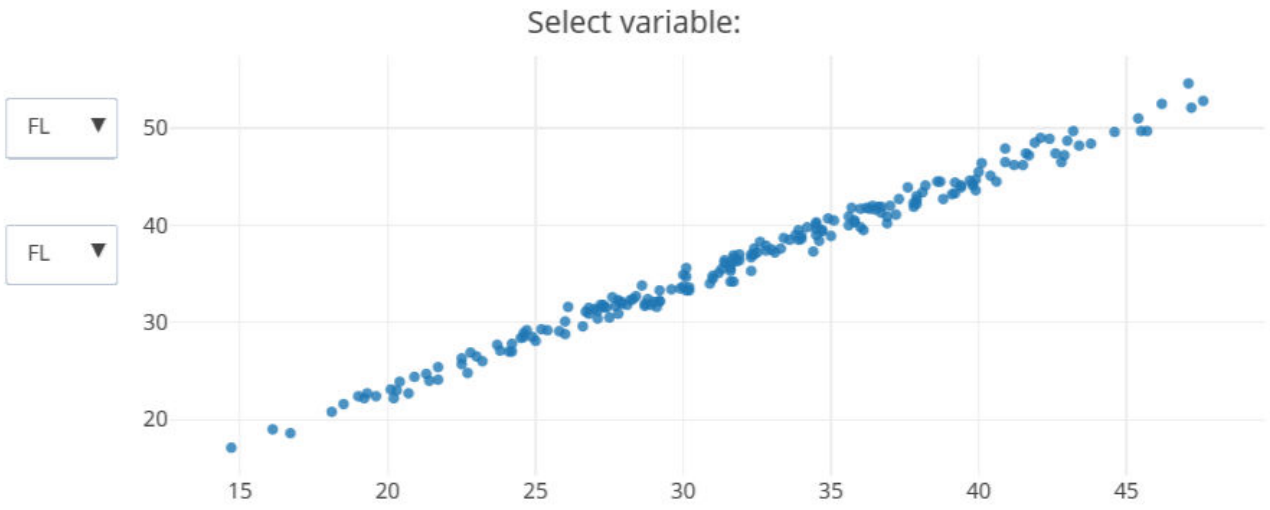
Brush color

rgba(228,26,28,1)



Reconfiguring operators

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

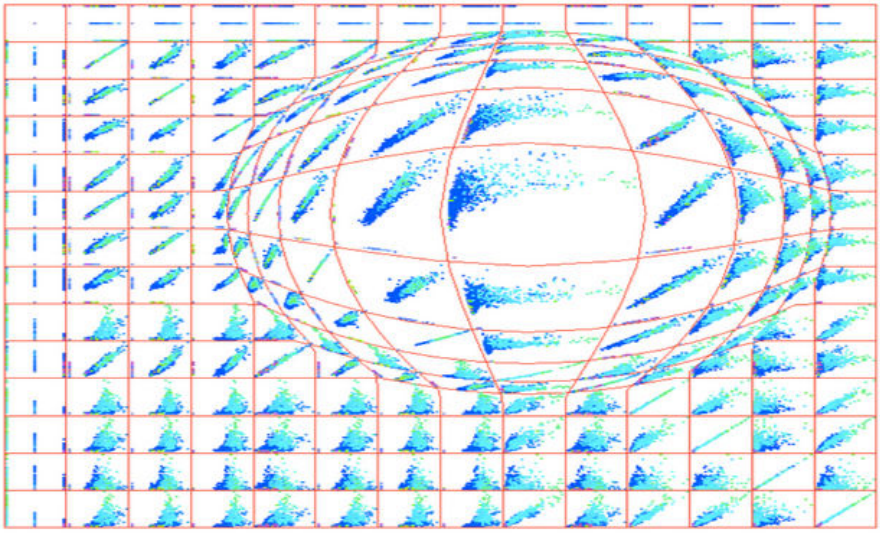


Encoding operators

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

Abstraction operators

- Distoring objects locally or globally



Interaction operands

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

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

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

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

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

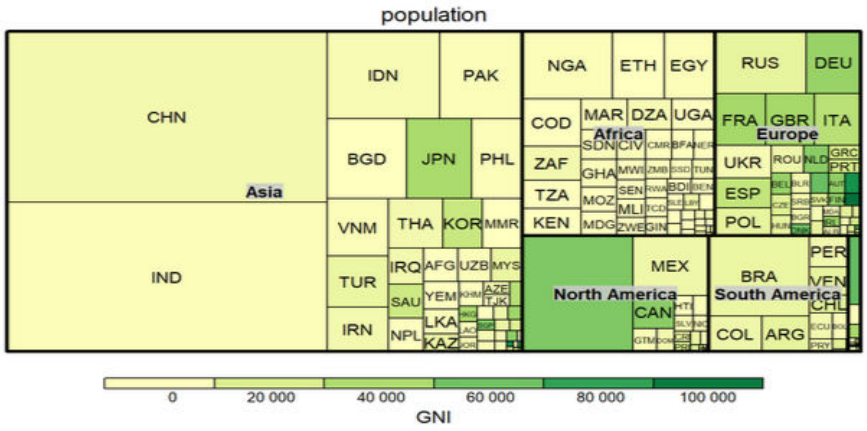
Analysis:

- 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

Treemaps

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

Example: World population



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

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

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.

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