Data Science - Assignment #1

Daniel Glants I.D:203267182 / Omri Ben Menahem I.D:204048771

```
knitr::opts_chunk$set(echo = TRUE)
library(reshape2)
library(magrittr)
library(DescTools)
library(tictoc)
library(dplyr)
```

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

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

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

```
library(ggplot2)
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:DescTools':
##
##     %like%

## The following objects are masked from 'package:reshape2':
##
##     dcast, melt
```

Question 1:

Part 1.a:

```r
# Write a function called my_aggregation:

my_aggregation <- function(x, is.truncated = FALSE) {
  if (is.truncated == FALSE){
    cat(paste("mean:",mean(x),"\nvar:",var(x),"\nmed:",median(x)))
  } else{
    cat(paste("mean:",mean(x,trim = 0.05),"\nvar:",var(Trim(x, trim = 0.05)),"\nmed:",
              median(Trim(x, trim = 0.05))))
  }
}
```

Part 1.b:

When do you expect that the truncated aggregates will be very different than the non-truncated aggregates?

Answer: We can expect the truncated aggregates to differ from the none-truncated values the more diffrense we've got among the observations.

We define Robust statistics as measures on which extreme observations have little effect.

The Median is a parameter that half of the observations are greater then him and half are lesser, thus is the most rubust and isn't effected by discarding the top and the bottom 5% of the obsservations.

```r
# Generate a vector of one million random variables each distributed Lognormal(1, 0.5):

set.seed(256)
a <- rlnorm(1000000,1, 0.5)
head(a)
```

```
## [1] 2.618684 1.642807 1.605792 3.454034 1.684118 3.001691
```

```r
# Compare the truncated mean, variance and median to the non-truncatNed aggregates.

# Truncated:
my_aggregation(a, is.truncated = TRUE)
```

```
## mean: 2.93277251932734
## var: 1.34386872850215
## med: 2.71702974002026
```

```r
# Non-truncated:
my_aggregation(a, is.truncated = FALSE)
```

```
## mean: 3.07662077845666
## var: 2.68637168471582
## med: 2.71702974002026
```

Part 1.c:

Compare the time takes R to compute the truncated mean with your function and with R base mean function. Why do you think they differ?

```r
# Truncated mean
tic("Runing Time:")
my_aggregation(a, is.truncated = TRUE)
```

```
## mean: 2.93277251932734
## var: 1.34386872850215
## med: 2.71702974002026
```

```r
toc()
```

```
## Runing Time:: 0.59 sec elapsed
```

```r
# Non-truncated:
tic("Runing Time:")
mean(a,trim = 0.05)
```

```
## [1] 2.932773
```

```r
toc()
```

```
## Runing Time:: 0.05 sec elapsed
```

The results differ mainly in the mean and the var, because we trimmed the outliers of the data set, as expected, the 'med' as the most robust veriable, hasn't changed

The difference between the two run times emerge from the code it self, computing the truncated values with the "my_aggregation" function steps through more levels then the R Base function thus takes more time.

Question 2

```
library(data.table)
db2 <- airquality
```

part 2.a:

```
variable_NA_val <- apply(is.na(db2),MARGIN=2,FUN=sum)
is.na_row <- apply(is.na(db2),MARGIN=1,FUN=any) #checks if row contains a True value for NA
```

When using apply, margin 1 indicates applying over columns(variable) and 2 over rows(observation). Alternatively, we could use colSums(is.na(db2)) instead of the apply function.

part 2.b:

```
NA_ind <- which(is.na(db2),arr.ind=TRUE) #matrix of indexes of NA values
month_needed <- db2[NA_ind[,1],5]
req <- cbind(NA_ind,month_needed)
airquality_imputed <- as.data.table(db2)
db2_copy <- db2
mean_db2_month <- airquality_imputed[,list(Ozone=mean(Ozone,na.rm = TRUE), Solar.R=mean(Solar.R,na.rm =
Wind=mean(Wind,na.rm = TRUE), Temp=mean(Temp,na.rm = TRUE)), by=Month]
mean_db2_month <- mean_db2_month[,c(2:5,1)]

values <- vector()
(unique(req[,2])) # see the different columns with NA values
```

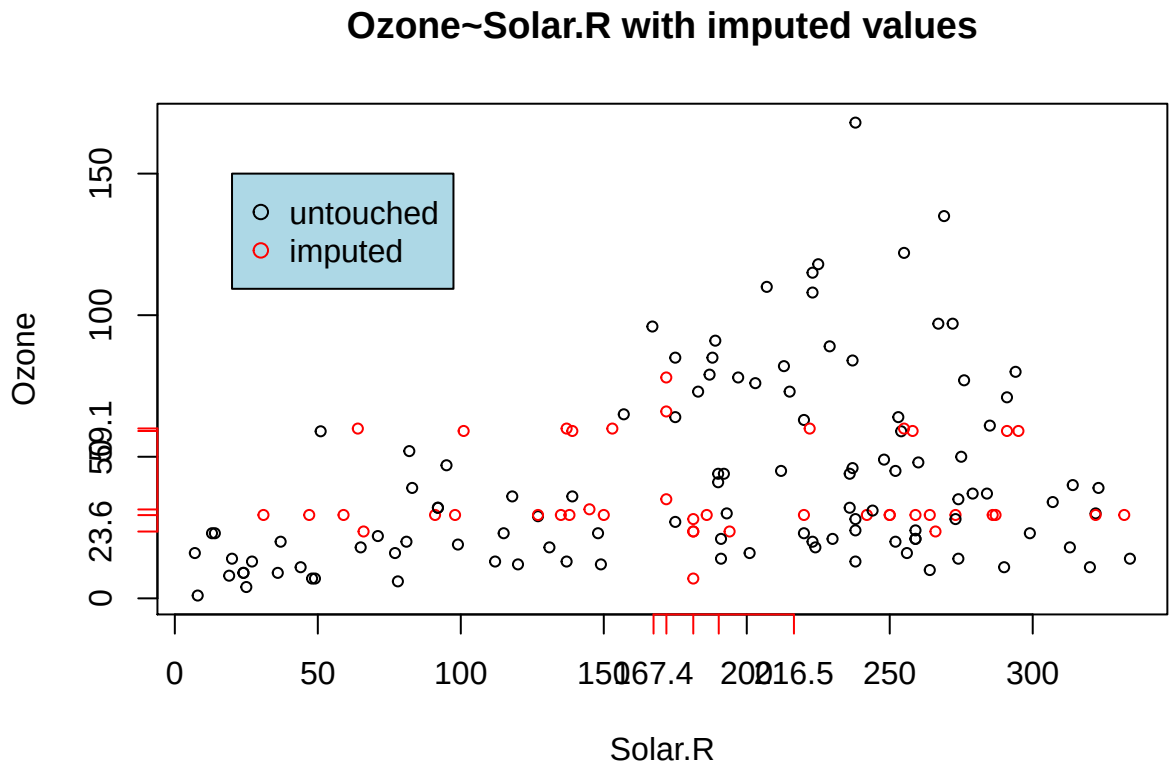
```
## [1] 1 2
```

```
for (i in 1:length(req[,1])){
  if (req[i,2] == 1)  #problem with integer type and Col Name
    value <- mean_db2_month[Month==req[i,3],Ozone,]
  else if (req[i,2] == 2)
    value <- mean_db2_month[Month==req[i,3],Solar.R,]
  else
    print("Unexpected value, no value added")
  values <- append(values,value)
}

db2_copy[NA_ind] <- values
airquality_imputed <- as.data.table(db2_copy)
```

Not very elegant but does the trick, after hours of unsuccessfully trying to get it to work properly with the power of data tables.

part 2.c:

```
plot(airquality_imputed$Ozone~airquality_imputed$Solar.R, xlab = 'Solar.R', ylab = 'Ozone',cex=0.7,col=
axis(2,at = round(mean_db2_month$Ozone,1),col = "red")
axis(1, at = round(mean_db2_month$Solar.R,1),col = "red")
title(main='Ozone~Solar.R with imputed values')
legend(x=20, y=150, legend=c("untouched", "imputed"), col=c("black","red"),pch=1, cex=1,bg='lightblue')
```



Ozone values ploted with Solar.R from the imputed data. The dots which were imputed are marked and so are their points on the axis.

Question 3:

```
# Loading Data Set:
data("diamonds")
```

Part 3.a:

```
# The Dataset's dimantions are:

dim(diamonds)
```

```
## [1] 53940    10
```

```r
# The Dataset's Calss is:
class(diamonds)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

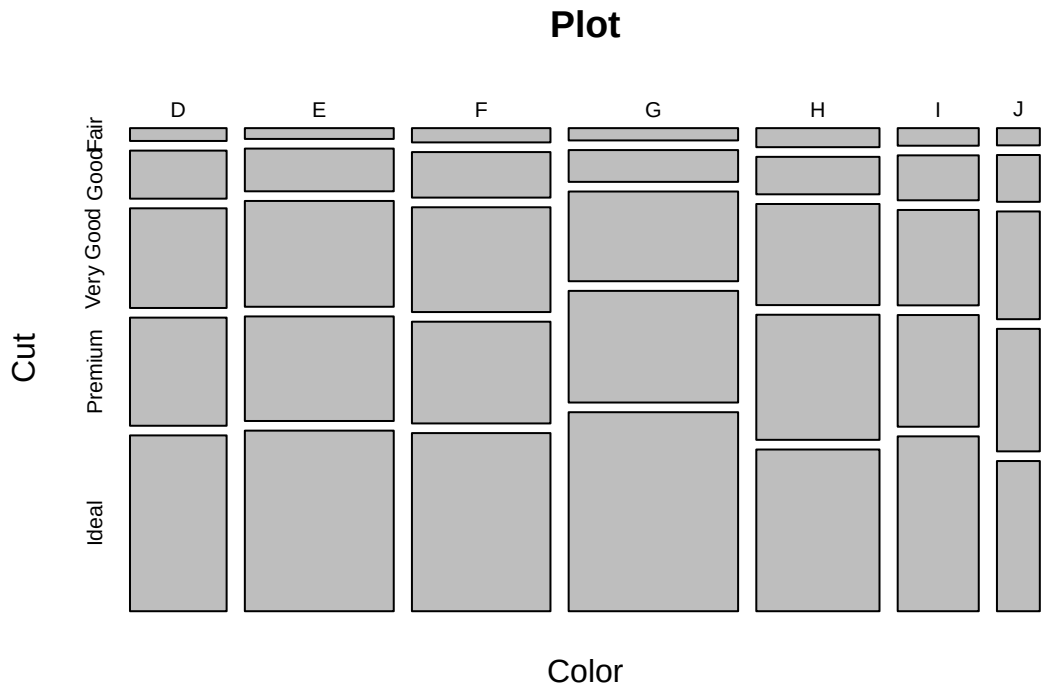
Part 3.b:

```r
# Take a random subset of size 25% of the database named d, and make it a data.table object.

d <- setDT(diamonds[sample(nrow(diamonds),0.25*nrow(diamonds)),])
class(d)
```

```
## [1] "data.table" "data.frame"
```

Part 3.c:

```r
# Use a mosaic plot to present the data distribution according to color and cut.

Plot<- table(d$color,d$cut)
mosaicplot(Plot,xlab = "Color", ylab = "Cut")
```



explanation:

We can see that most of the diamonds in the "d" data set are from Color 'G'. from them, the majority have 'Ideal' cut. "Fair" cut diamonds are very scarce in every color group while the 'j' color group is the smallest

Part 3.d:

creat the folowing feutures:

```
d$logp <- log10(d$price) #log base 10 of price
```

3.d.1:

```
d$v <- (d$x*d$y*d$z) #diamond volume
```

3.d.2:

```
# conditions:

cond_1 <- (d$cut == "Ideal")
cond_2 <- (d$depth < median(d$depth))
cond_3 <- (d$clarity != "I1")
cond_4 <- (d$color %in% c("D","E","F","G"))

cond_mat <- data.frame(cond_1,cond_2,cond_3, cond_4) #Creating cond Matrix according to demands
cond_mat <- cbind(cond_mat, total = rowSums(cond_mat)) #summing the Condition for every row

# Creating cond1 by the instructions
d$cond1 <- (cond_mat$total >= 3)
```

3.d.3:

Part 3.e:

```
d[,.("Mean logp"=mean(logp),"Mean v"=mean(v),
     "Var logp"=var(logp),"Var V"=var(v)),by=.("Condition"=cond1)]%>%print()
```

```
##    Condition Mean logp   Mean v  Var logp    Var V
## 1:     FALSE  3.447946 148.7127 0.2031097 7125.031
## 2:      TRUE  3.332007 114.8248 0.1819776 4483.657
```

Part 3.f:

```
color_cut <- setDT(unique(expand.grid(color = levels(d$color),cut = levels(d$cut))))
class(color_cut)
```

```
## [1] "data.table" "data.frame"
```

```
color_cut$some_feature <- rnorm(nrow(color_cut))

merged.dt <- merge(x=d, y = color_cut, by=c("cut","color"))
```

```
merged.dt[order(-clarity),.("Average some_feature"=mean(some_feature)),by=clarity]
```

3.f.i:

```
##    clarity Average some_feature
## 1:      IF          0.366292427
## 2:    VVS1          0.282376126
## 3:    VVS2          0.188898918
## 4:     VS1          0.146080204
## 5:     VS2          0.124006400
## 6:     SI1          0.128624473
## 7:     SI2          0.071900453
## 8:      I1         -0.008634671
```

```
merged.dt[some_feature>1][,.("Standard Deviation"=sd(price),"IQR"=IQR(price),"MAD"=mad(price)),by=.(cut
```

3.f.ii:

```
##          cut Standard Deviation     IQR      MAD
## 1:      Fair           3497.898 2396.50 1601.208
## 2: Very Good           4188.835 4777.50 3533.777
## 3:   Premium           3846.362 3779.25 1758.364
## 4:     Ideal           4304.840 5051.00 3033.400
```

```
merged.dt[,.("Cond i"=NROW(v[(1<carat)&(carat<2)]),"Cond ii"=NROW(v[(5000<price)&(price<10000)])),by=.(
```

3.f.iii:

```
##    color Cond i Cond ii
## 1:     D    314     204
## 2:     E    480     298
## 3:     F    623     368
## 4:     G    876     566
## 5:     H    787     446
## 6:     I    522     290
## 7:     J    294     171
```

Part 3.g:

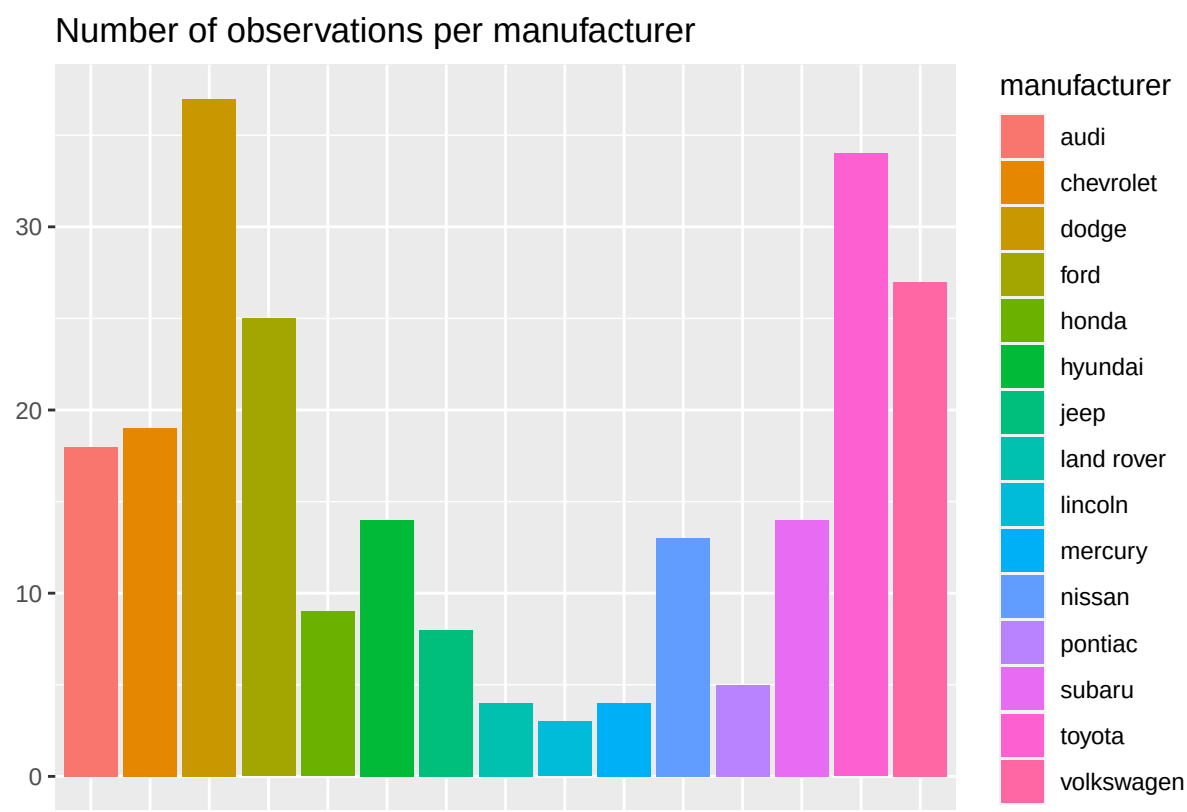
```
meld_dt <- melt(merged.dt,id.vars = c("cut","color"),measure.vars =c("price"))
res_mat <- acast(meld_dt,cut~color~variable,value.var = "value",fun.aggregate = mean)
round(res_mat)
```

```
## , , price
##
##               D    E    F    G    H    I    J
## Fair       4664 3491 4002 4143 4918 4447 6143
## Good       3289 3591 3698 4152 4579 5478 4831
## Very Good 3522 3266 3904 3769 4472 5338 5402
## Premium    3650 3570 4432 4604 5217 6406 6464
## Ideal      2652 2542 3493 3774 3964 4028 5222
```

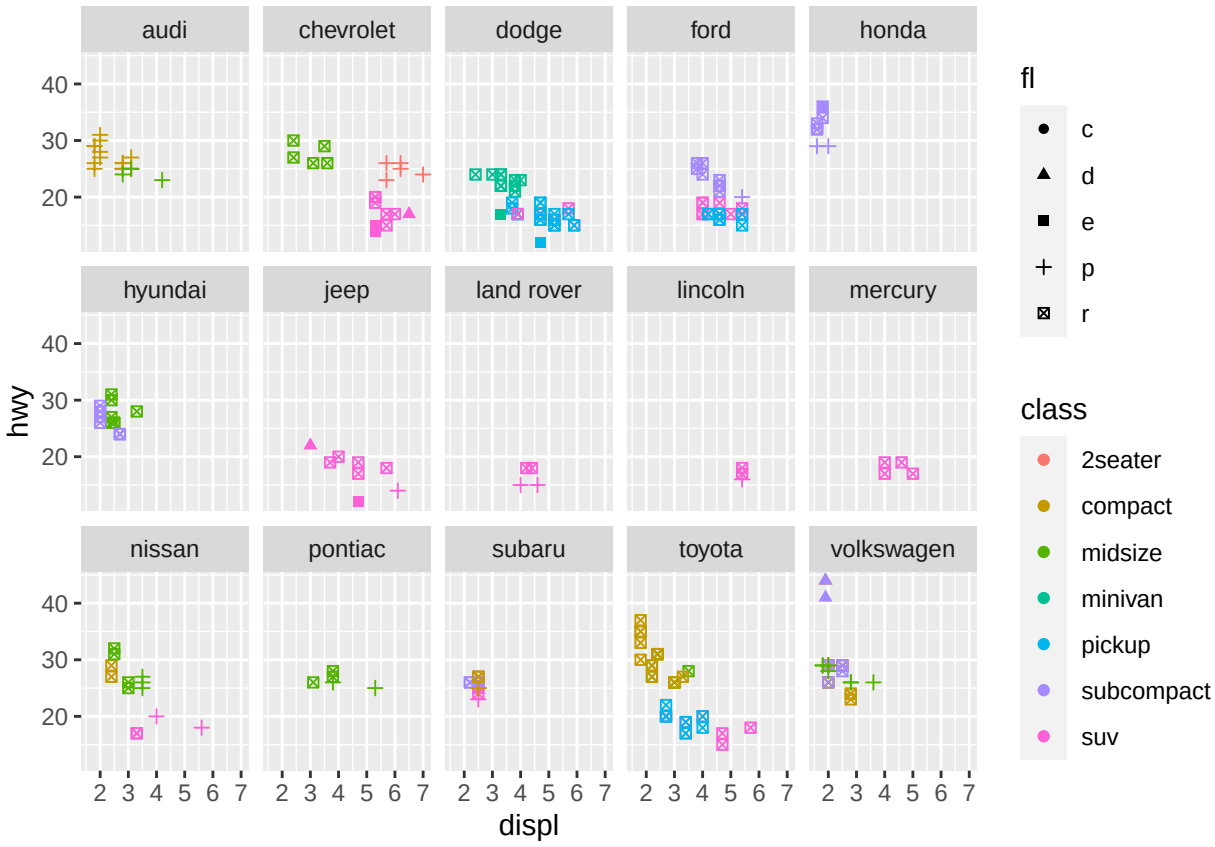
Question 4:

```
library(ggplot2)
db4 <- mpg
```

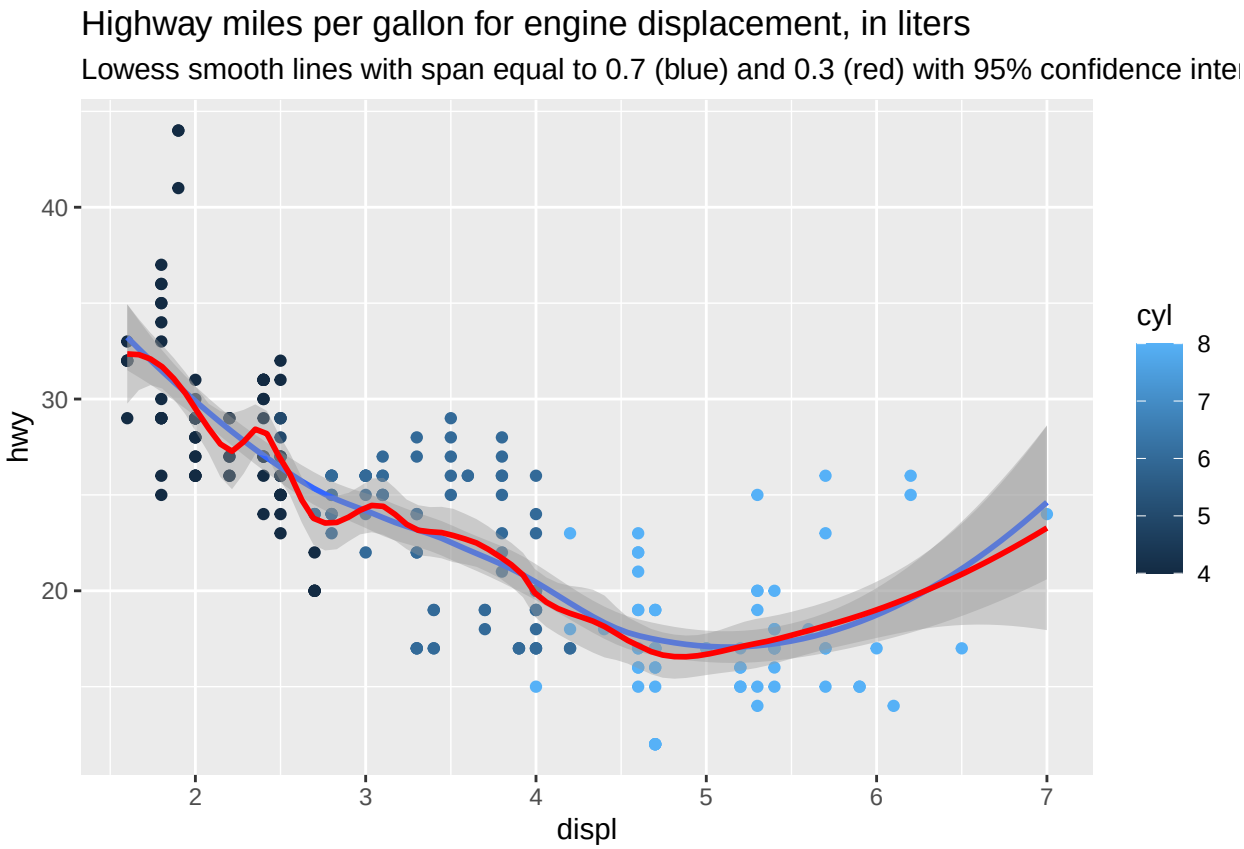
```
ggplot(data = db4, aes(x=manufacturer),frame.plot = F) + geom_bar(aes(fill=manufacturer))+
  ggtitle(label = "Number of observations per manufacturer")+xlab("")+ylab("")+
  theme(axis.text.x = element_blank(), axis.ticks.x = element_blank())
```



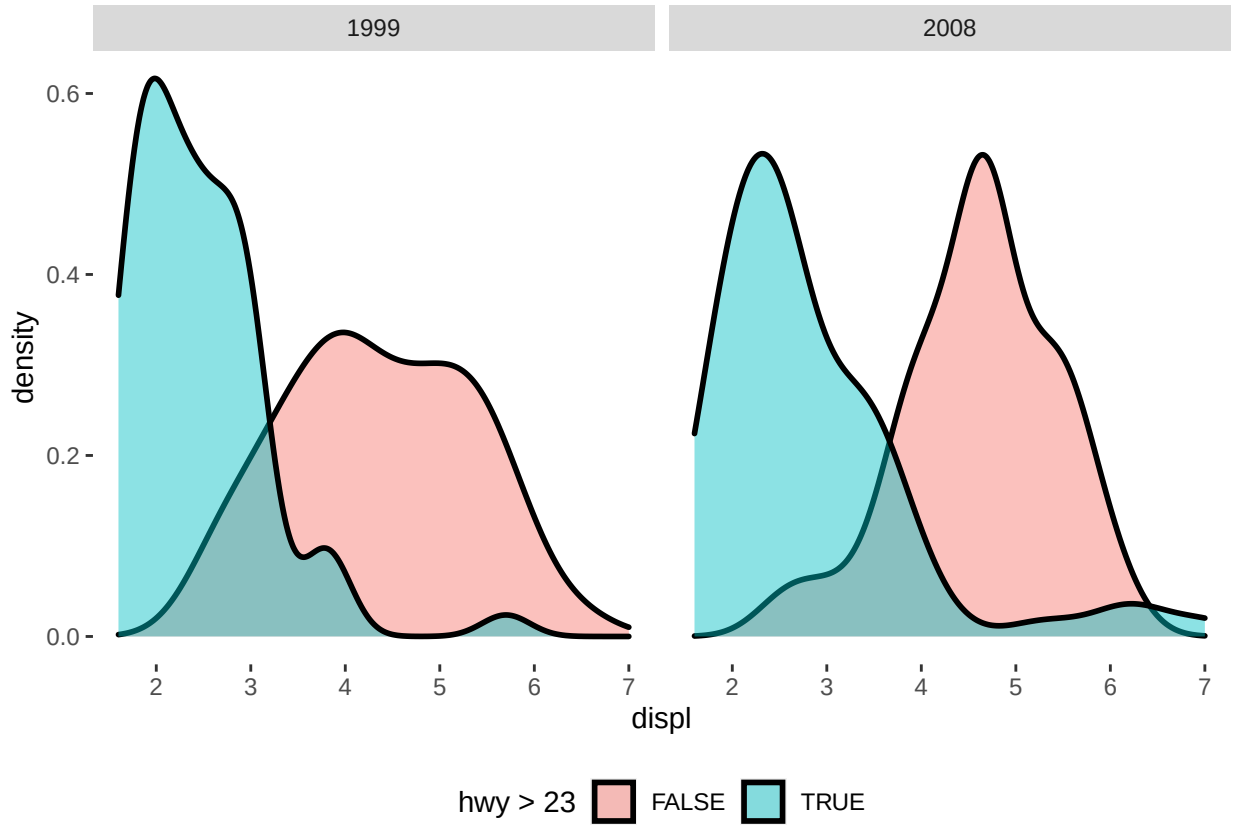
```
ggplot(data = db4, aes(x=displ, y=hwy))+geom_point(aes(color=class,shape=fl))+facet_wrap(~manufacturer,
```




```
ggplot(data = db4, aes(x=displ, y=hwy),)+ geom_point(aes(color=cyl))+ geom_smooth(span=0.7)+geom_smooth
  ggtitle(label = "Highway miles per gallon for engine displacement, in liters", subtitle = "Lowess smo
```




```r
db4_1999_2008 <- subset(db4, db4$year == 1999 | db4$year == 2008) # only needed data
ggplot(data = db4_1999_2008, aes(x=displ, fill = hwy > 23)) + geom_density(size=1,alpha=0.45) +
  theme(panel.background = element_blank(),
  legend.position = "bottom")+facet_wrap(~year)
```



Question 5:

Part 5.a:

```
airq <- data.table(airquality,keep.rownames = TRUE)
stock <- data.table(EuStockMarkets,keep.rownames = TRUE)
```

Part 5.b:

```
#Create variable named "date" of class "Date" of the date of airq
for(i in 1:nrow(airq)){
 airq$date[i] <-  paste("2019",
                        toString(airq[,Month[i]]),
                        toString(airq[,Day[i]]),sep = "/")
}
airq$date <- airq$date %>% as.Date()

#generate new variable also named "date" which start ats "2019-01-01"
y <-c("2019/01/01")%>%as.Date()
for(i in 2:nrow(stock)){
  y[i] <- y[i-1]+1
```

```
}
stock$date <- y
```

part 5.c:

```
setkey(airq, date) #left
setkey(stock, date) #right

  joined <- stock[airq,,]
```

part 5.d:

```
min_date <- joined[joined$CAC ==min(joined$CAC)]$date %>% as.POSIXct()
max_date <- joined[joined$CAC ==max(joined$CAC)]$date %>% as.POSIXct()

difftime(max_date,min_date,units = "weeks")
```

```
## Time difference of 14.28571 weeks
```

```
difftime(max_date,min_date,units = "days")
```

```
## Time difference of 100 days
```

```
difftime(max_date,min_date,units = "hours")
```

```
## Time difference of 2400 hours
```

part 5.e:

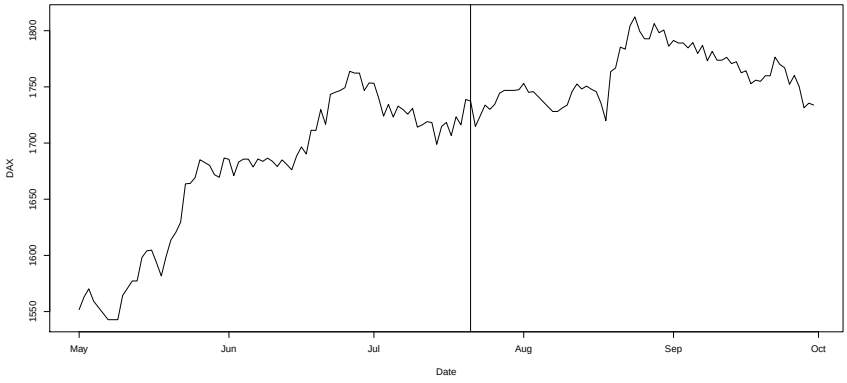
```
min <- joined[joined$CAC ==min(joined$CAC)]$date
max <- joined[joined$CAC ==max(joined$CAC)]$date

# the average temperature at this period is:
mean(joined$Temp[(joined$date>min)&(joined$date<max)])
```

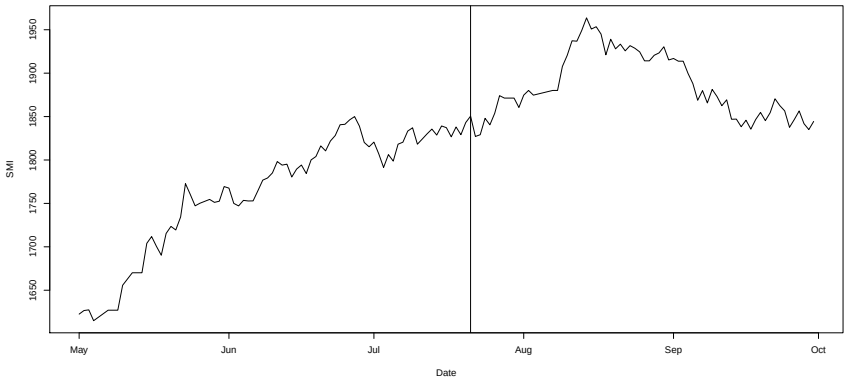
```
## [1] 78.0404
```

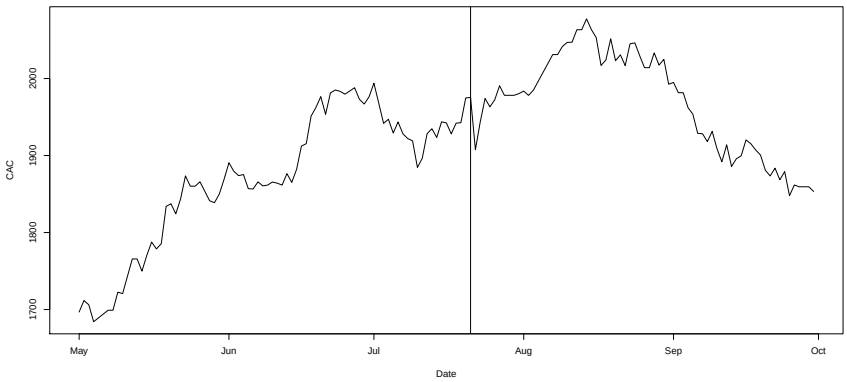
part 5.f:

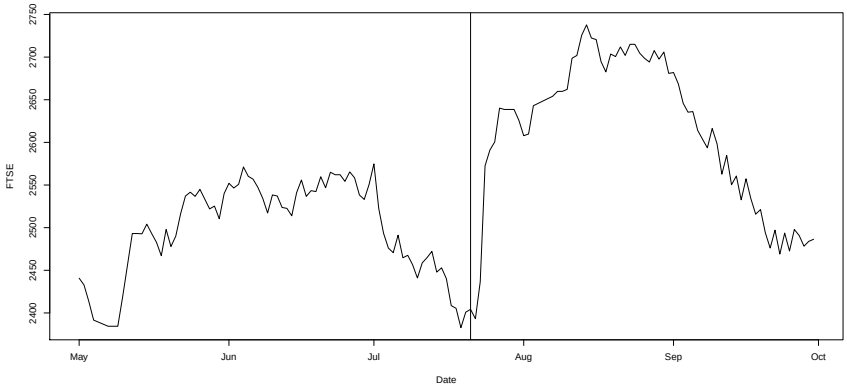
```
par(mfrow = c(4, 1))
df <- joined[!is.na(joined$Solar.R), ]
min_Solar.R <- df$date[df$Solar.R==min(df$Solar.R)]
```

```
count <- 1

for(stock in df[,1:4]) {
  plot(x = df$date, y = stock  ,type = "l", ylab = colnames(df)[count], xlab = "Date")
  abline(v = min_Solar.R)
  count <- count+1
}
```


DAX
1550
1600
1650
1700
1750
1800
May
Jun
Jul
Aug
Sep
Oct
Date


SMI
1650
1700
1750
1800
1850
1900
1950
May
Jun
Jul
Aug
Sep
Oct
Date

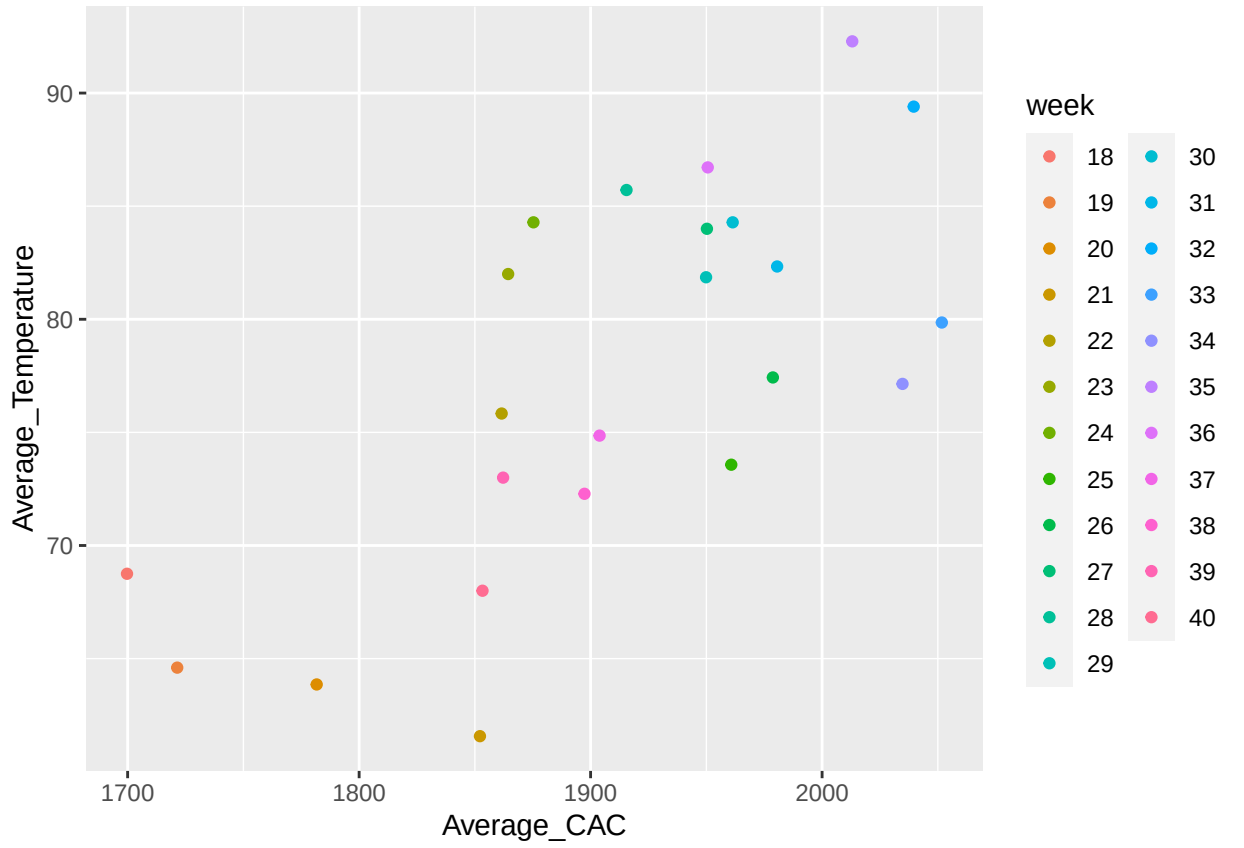

CAC
1700
1800
1900
2000
May
Jun
Jul
Aug
Sep
Oct
Date


FTSE
2400
2450
2500
2550
2600
2650
2700
2750
May
Jun
Jul
Aug
Sep
Oct
Date

```
rm(count)
```

part 5.g:

```
splitByWeek <- cbind(df$date, isoweek(df$date))
splitByWeek <- splitByWeek %>% as.data.table()
splitByWeek$date <-  df$date



splitByWeek$week <- splitByWeek$V2
splitByWeek <- splitByWeek[,!c("V1","V2")]

setkey(splitByWeek,date)
setkey(df,date)
df1 <- df[splitByWeek,,]

df2 <- df1[,.("Average_CAC"=mean(CAC),"Average_Temperature"=mean(Temp)),by = week]


df2$week <- as.factor(df2$week)

p <- ggplot(data =df2, aes(x =Average_CAC, y= Average_Temperature,colour=week))
p<-p+geom_point()
print(p)
```



Question 6:

part 6.a:

```
library(kableExtra)
library(dplyr)
```

```
db6 <- read.csv("2017.csv")
#db6 <- read.csv('https://www.kaggle.com/unsdsn/world-happiness#2017.csv')
names(db6) <- gsub('..', '-', names(db6), fixed = T)
```

```
knitr::kable(head(db6), booktabs = T) %>%
  kable_styling(latex_options =c("striped", "scale_down"),font_size = 15)
```

Country	Happiness.Rank	Happiness.Score	Whisker.high	Whisker.low	Economy-GDP.per.Capita.	Family	Health-Life.Expectancy.	Freedom	Generosity	Trust-Government.Corruption.	Dystopia.Residual
Norway	1	7.537	7.594445	7.479556	1.616463	1.533524	0.7966665	0.6354226	0.3620122	0.3159638	2.277027
Denmark	2	7.522	7.581728	7.462272	1.482383	1.551122	0.7925655	0.6260067	0.3552805	0.4007701	2.313707
Iceland	3	7.504	7.622031	7.385970	1.480633	1.610574	0.8335521	0.6271626	0.4755402	0.1535266	2.322715
Switzerland	4	7.494	7.561772	7.426228	1.564980	1.516912	0.8581313	0.6200706	0.2905493	0.3670073	2.276716
Finland	5	7.469	7.527542	7.410458	1.443572	1.540247	0.8091577	0.6179509	0.2454828	0.3826115	2.430182
Netherlands	6	7.377	7.427426	7.326574	1.503945	1.428939	0.8106961	0.5853845	0.4704898	0.2826618	2.294804

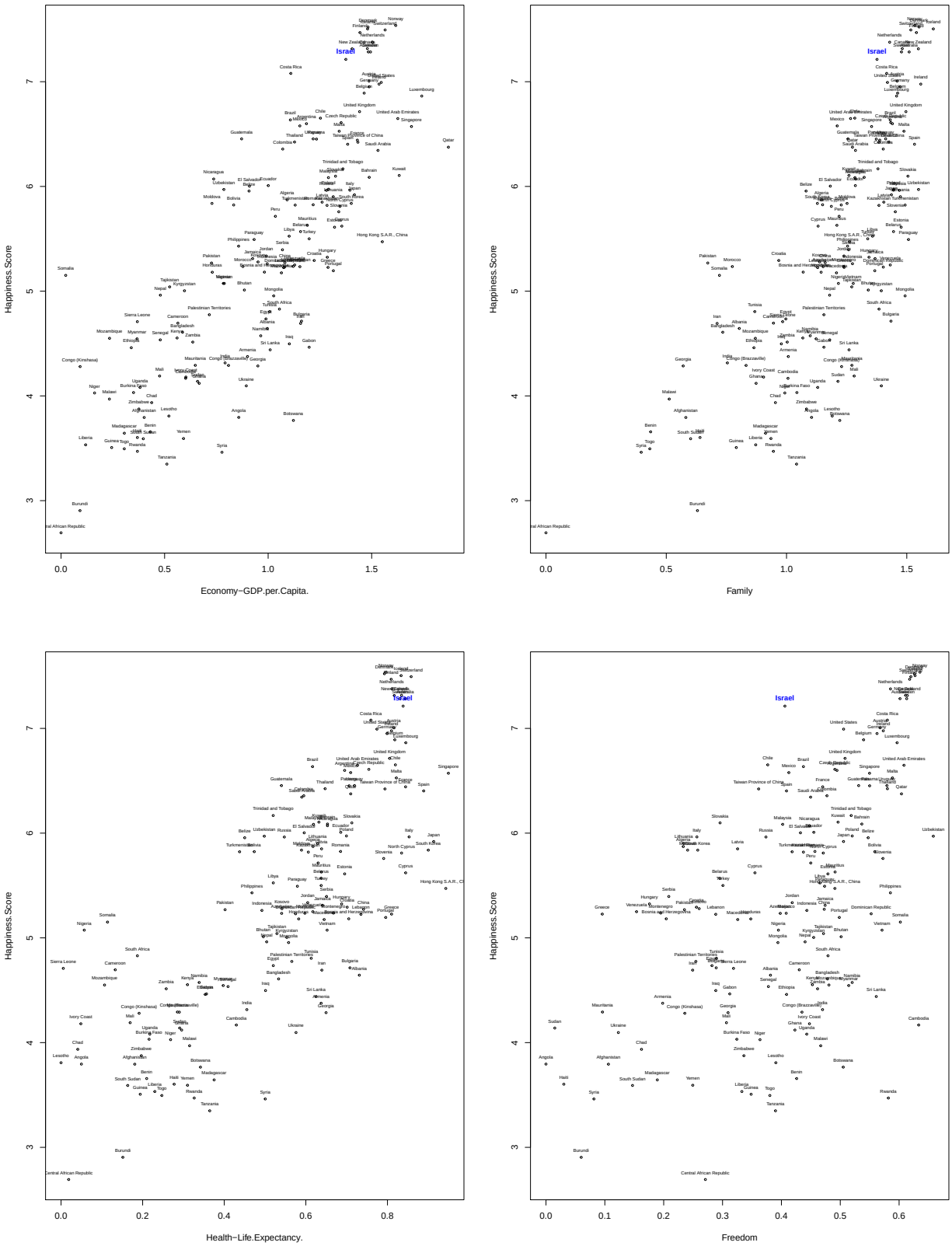
Modifies the databases column names, by replacing '..' with '-'. '.' is recognized by r as a char, therefore we use fixed = True in order for it to be recognized as the character period and not just *any* char. The header of the table is presented while using kable package for nicer presentation.

part 6.b:

```
db6_isr <- db6[which(db6$Country=='Israel'),]
db6_rest <- db6[which(db6$Country!='Israel'),]
x_lab <- c("Economy-GDP.per.Capita.","Family","Health-Life.Expectancy.","Freedom")
```

Seperate Israel's data from rest for easier coding.

```
par(mfrow = c(2, 2))
for (i in 6:9){
  plot(db6$Happiness.Score~db6[,i],ylab ="Happiness.Score",xlab = x_lab[i-5], lty = "solid",cex = .5)+
    graphics::text(db6_isr$Happiness.Score~db6_isr[,i],
                   labels=db6_isr[,1], cex = 0.8, pos = 3,col = "blue", font = 2)+
    graphics::text(db6_rest$Happiness.Score~db6_rest[,i],
                   labels=db6_rest[,1], pos = 3, cex = 0.5)
}
```



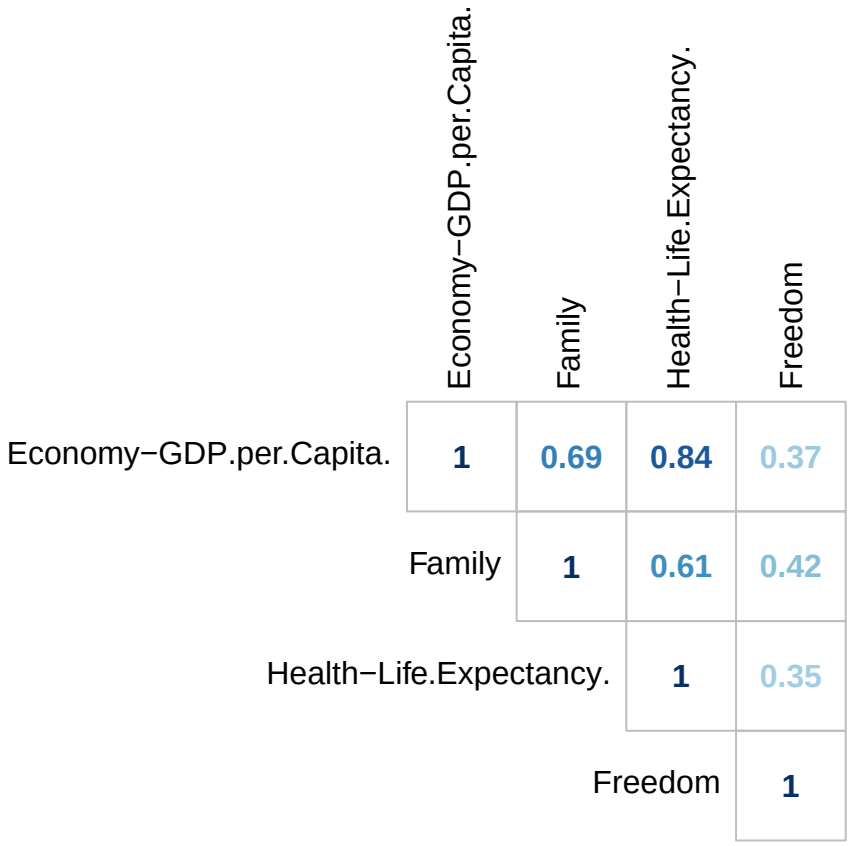
Very clustered due to many countries and long names. Shows Happiness.Score vs different measurements for

every country, with Israel bold and in blue. Done with a loop for better modularity.

part 6.c:

```
library(corrplot)
```

```
happy_corr <- cor(db6$Happiness.Score,db6[,6:12])
cor_names <- names(happy_corr[,happy_corr>0.5 | happy_corr < -0.5])
db6_c <- db6[cor_names]
M <- cor(db6_c)
corrplot(M, method = "number", type = "upper",cl.pos = "n",tl.col = "black")
```



A very clean and simple correlation plot which shows the correlation between the different factors which are correlated with "Happiness.Score". At first we created a vector containing the interaction each factor had with "Happiness.Score" and afterwards we filtered only those with values larger than 0.5 or smaller than -0.5. These values apply as a rule of thumb for correlation. Afterwards we took a subset of our data which consisted only the factors above and plotted their interaction with one another. We decided to remove unnecessary information and focus on having the plot clean since by its nature this plot is quite simple to read and there's no need for extra information which helps reading much larger plots which can often be difficulty to read.

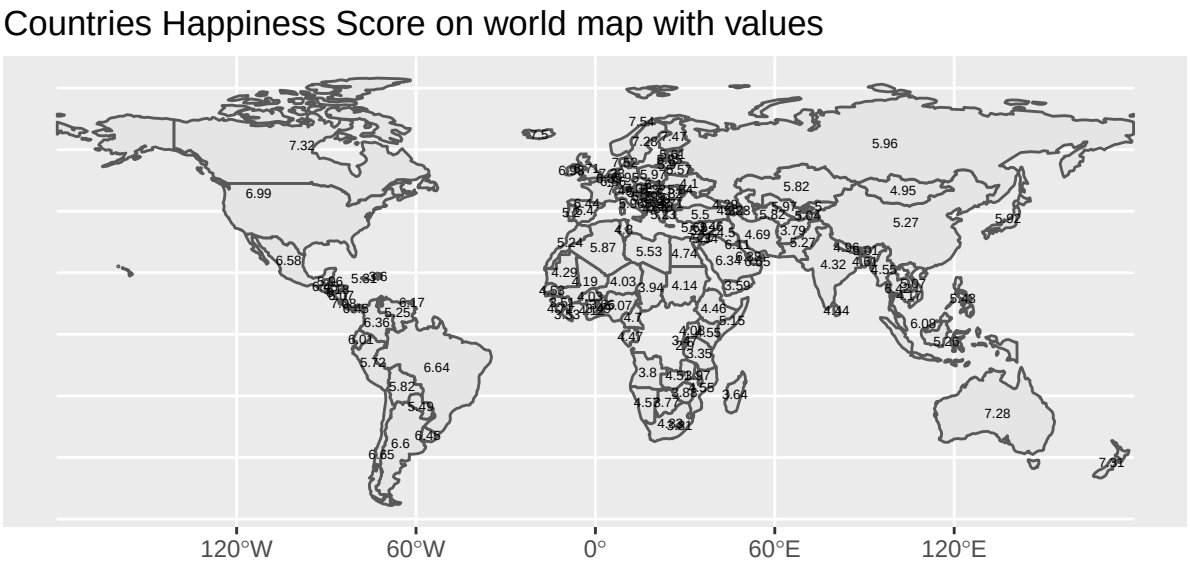
part 6.d:

```
library("ggplot2")
library("sf")
library("rnaturalearth")
library("rnaturalearthdata")
library("rgeos")
library("rworldmap")
library("RColorBrewer")
```

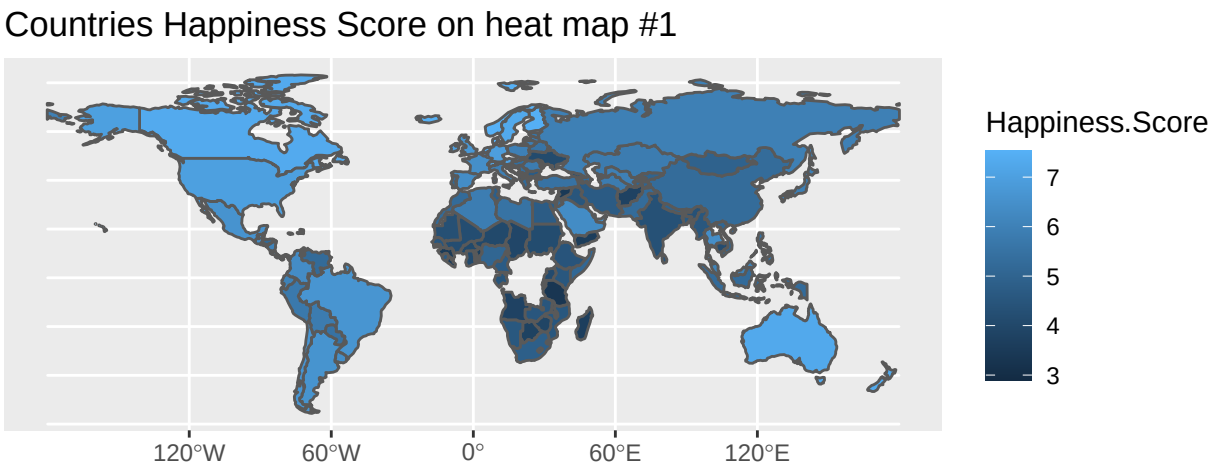
```
db6_d <- db6[,c(1,3)]
db6_d$Happiness.Score <- round(db6_d$Happiness.Score,2)
```

```
world <- ne_countries(returnclass = "sf")
names(world)[names(world) == "name"] <- "Country"
world_happy <- merge(world,db6_d,by="Country")
world_points<- st_centroid(world_happy)
world_points <- cbind(world_happy, st_coordinates(st_centroid(world_happy$geometry)))
```

```
(gg_map1 <- ggplot(data = world_happy) +geom_sf() +
  geom_text(data= world_points,aes(x=X, y=Y, label=Happiness.Score),check_overlap = FALSE, size = 1.8)+
  ggtitle(label = "Countries Happiness Score on world map with values")+labs(x="",y=""))
```



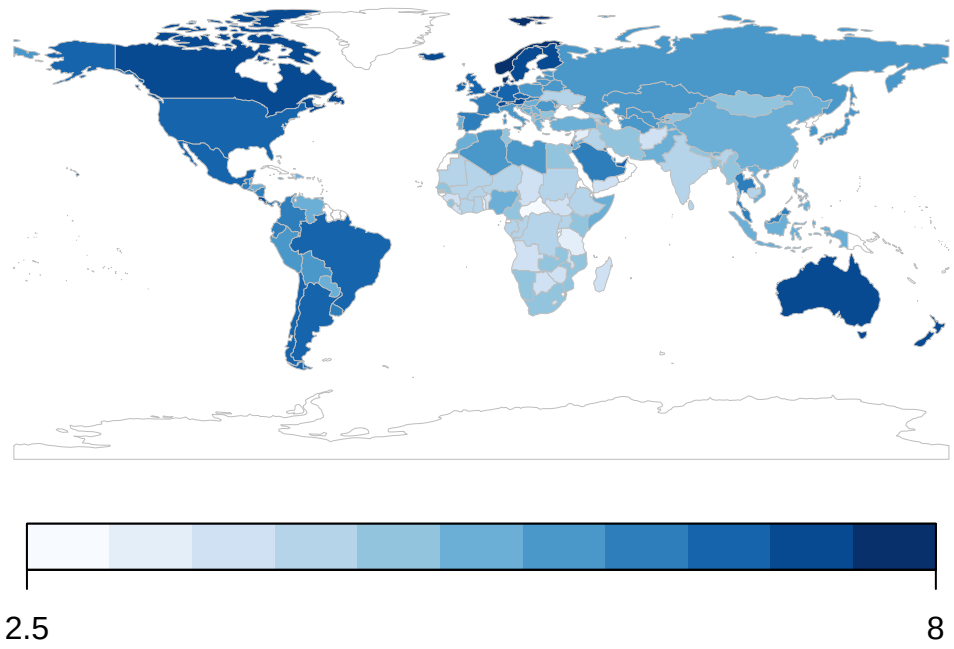
```
(gg_map2 <- ggplot(data = world_happy) +
  geom_sf(aes(fill = Happiness.Score))+ggtitle(label = "Countries Happiness Score on heat map #1")) # s
```




```
colors <- RColorBrewer::brewer.pal(9,'Blues')

matched <- joinCountryData2Map(db6_d, joinCode="NAME", nameJoinColumn="Country")
```

```
## 151 codes from your data successfully matched countries in the map
## 4 codes from your data failed to match with a country code in the map
## 92 codes from the map weren't represented in your data
```

```
gg_map3 <- mapCountryData(matched,
                          nameColumnToPlot="Happiness.Score",
                          mapTitle="Countries Happiness Score on heat map #2",
                          catMethod = "pretty",
                          colourPalette = colors)
```



The First map is a map which plots the Happiness.Score numeric values on the world map, while the second map is a heat map which possibly is slightly easier to read. Both maps use quite a few libraries therefore we created a 3rd map which uses rworldmap library and code wise is cleaner and simpler.

Question 7:

```
setwd("C:/Users/Daniel/Desktop/BGU/Year 4/Semester B/Data Science/HW")
auto <- fread('autos.csv')
```

```
## Warning in fread("autos.csv"): Found and resolved improper
## quoting out-of-sample. First healed line 5263: <<2016-03-29
## 16:46:46,"_SPARDOSE"______Polo_1_4___6N1___60PS___5Tuerer____FESTPREIS,privat,Angebot,
## 500,control,limousine,1999,manuell,60,polo,150000,12,benzin,volkswagen,ja,
## 2016-03-25 00:00:00,0,59581,2016-03-30 11:46:58>>. If the fields are not quoted
## (e.g. field separator does not appear within any field), try quote="" to avoid
## this warning.
```

part 7.a:

```
mazda <- auto[grep("mazda",auto$name, ignore.case=TRUE)]%>%setDT #we also ask the matching to be case s

paste(nrow(mazda),"cars in this dataset have "Mazda" in their name")
```

```
## [1] "5463 cars in this dataset have "Mazda" in their name"
```

part 7.b:

```
library(stringr)
mazda$is_3 <- str_detect(mazda$model,"3")
```

part 7.c:

```
mazda$dateCreated <- mazda$dateCreated%>%as.POSIXct(format = "%Y-%m-%d %H:%M:%S", tz = "UTC")

mazda$lastSeen <- mazda$lastSeen%>%as.POSIXct(format = "%Y-%m-%d %H:%M:%S", tz = "UTC")

mazda$time_dif <- mazda[,difftime(mazda$lastSeen,mazda$dateCreated,units = "hours"),]

table <- ftable(mazda$fuelType,mazda$is_3)

res <- mazda[,.("Avarege Time"=mean(time_dif),"Number of Cars"=NROW(time_dif)),by=is_3]

res$diesel_type <- prop.table(table, margin = 2)[5,]

print(res)
```

```
##     is_3   Avarege Time Number of Cars diesel_type
## 1:  TRUE 211.6534 hours           1282   0.2085625
## 2: FALSE 229.7084 hours           4181   0.1138846
```

Question 8:

part 8.a:

```
q8a <- function(d) {   # frame of zeroes around matrix
  M <- matrix(1,nrow = d ,ncol= d)
  M[2:(d-1),2:(d-1)] <- 0
  return(M)
}
```

For example calling the function with size $= 4$ will give an output of a matrix of size 4×4, with the following result:

```
q8a(4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    1
## [2,]    1    0    0    1
## [3,]    1    0    0    1
## [4,]    1    1    1    1
```

part 8.b:

```
q8b <- function(vec1,vec2) {
if (length(vec1) != length(vec2)) # make sure not go 'out of bound'
  return (FALSE)
for (i in length(vec1)){
  if (vec1[i] != vec2[i])
      return (FALSE)
}
return (TRUE)
}
```

Function that checks if 2 vectors are identical, with loop. For example:

```
q8b(c(1,2,4),c(1,2,4))
```

```
## [1] TRUE
```

```
q8b(c(1,2,4),c(1,4,2))
```

```
## [1] FALSE
```

part 8.c:

```
q8c <- function(b, a) {
  a_no_b <- gsub(b,"",a)
  return (nchar(a) - nchar(a_no_b)) #Difference in length equals b appearances
}
```

A function that counts a chars appearances in a string. For example, for the word gOOood and the letter o we expected it bring back 2.

```
q8c('o',"gOOood")
```

```
## [1] 2
```

part 8.d:

```
library(lubridate)
```

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

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

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year

## The following objects are masked from 'package:dplyr':
##
##     intersect, setdiff, union

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

```
q8d <- function(bday) {   # prints info about your birth day
  today <- Sys.Date()   #date today
  daysPassed <- difftime(today,bday, units =  "day")
  age <- year(today)-year(bday)

  if (today < bday+years(age)) #no difftime with year units
    age <- age-1
  next_bday <- bday + years(age+1)
  daysTillbday <- difftime(next_bday,today, units =  "day")

  days <- c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday","Friday", "Saturday")
  bday2 <- as.POSIXlt(bday)
  dayOfWeek <- days[bday2$wday + 1]# requires POSIXlt class

  paste0("You were born on a ", dayOfWeek,", ", daysPassed, " days ago, and your next birthday is in "
         ,daysTillbday, " days.")
}
```

Function that gives information about our birthday, for example for my birthday: *8/14/1991* - You were born on a Wednesday, 10490 days ago, and your next birthday is in 103 days.

Question 9:

part 9.a:

```
cor(diamonds[sapply(diamonds, is.numeric)])
```

```
##            carat       depth      table      price           x           y
## carat 1.00000000  0.02822431  0.1816175  0.9215913  0.97509423  0.95172220
## depth 0.02822431  1.00000000 -0.2957785 -0.0106474 -0.02528925 -0.02934067
## table 0.18161755 -0.29577852  1.0000000  0.1271339  0.19534428  0.18376015
## price 0.92159130 -0.01064740  0.1271339  1.0000000  0.88443516  0.86542090
## x     0.97509423 -0.02528925  0.1953443  0.8844352  1.00000000  0.97470148
## y     0.95172220 -0.02934067  0.1837601  0.8654209  0.97470148  1.00000000
## z     0.95338738  0.09492388  0.1509287  0.8612494  0.97077180  0.95200572
##                z
## carat 0.95338738
```

```
## depth 0.09492388
## table 0.15092869
## price 0.86124944
## x     0.97077180
## y     0.95200572
## z     1.00000000
```

part 9.b:

cut and color aren't numeric variabls there for cannot be computed

part 9.c:

```
print(prop.table(ftable(diamonds[,c("cut","color")]), margin = 2))
```

```
##           color          D          E          F          G          H          I          J
## cut
## Fair            0.02405904 0.02286414 0.03269755 0.02780730 0.03648844 0.03227591 0.04237892
## Good            0.09771218 0.09523323 0.09526305 0.07713425 0.08453757 0.09627444 0.10933048
## Very Good       0.22332103 0.24497295 0.22678684 0.20359547 0.21965318 0.22205828 0.24145299
## Premium         0.23660517 0.23854241 0.24428841 0.25894439 0.28420039 0.26337145 0.28774929
## Ideal           0.41830258 0.39838726 0.40096416 0.43251860 0.37512042 0.38601992 0.31908832
```

part 9.d:

```
diamonds$color <- diamonds$color%>%as.integer()
cor(diamonds$color,diamonds$carat)
```

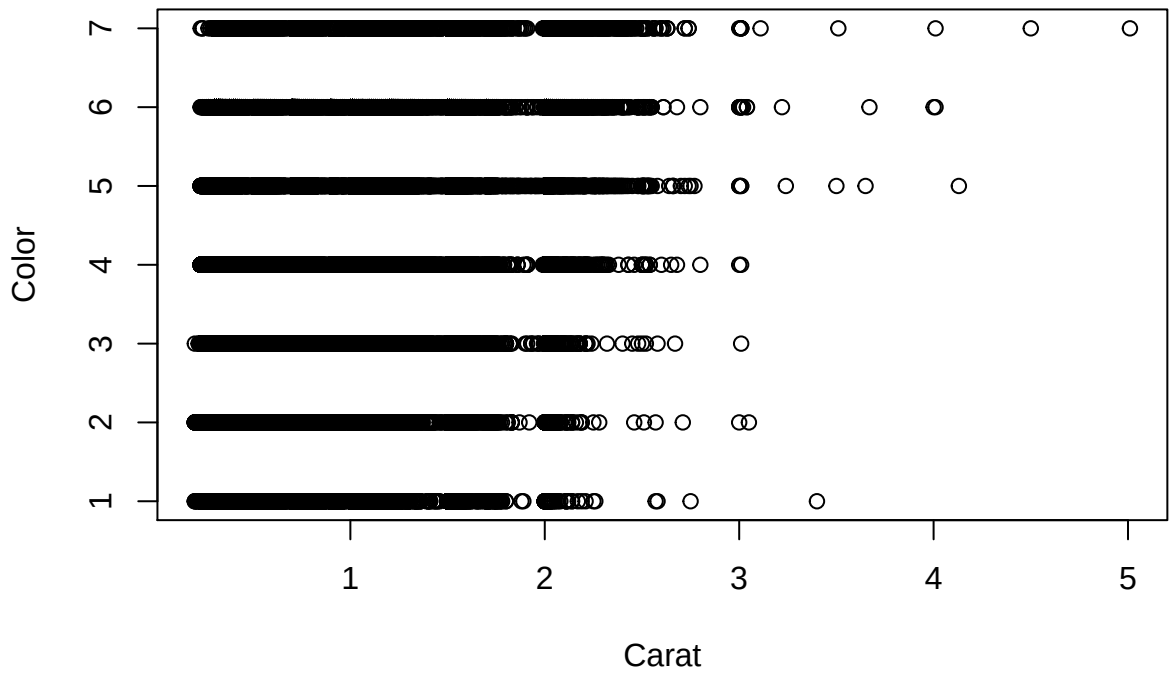
```
## [1] 0.2914368
```

the problem of this value is that color has no mathematical hirarchy in himself ther for the corralation between the color and carat has absolutly no segnificans.

part 9.e:

A better way to display the relationship between color and carat might be to plot it in to a scatter plot to get abetter understanding about the proportions of the two veriablse.

```
data("diamonds")
Plot1<- table(diamonds$color,diamonds$carat)
plot(x = diamonds$carat, y = diamonds$color,ylab = "Color", xlab = "Carat")
```



Question 10:

part 10.a:

```
q10a <- function(x, c = 1.4826){  #calculate MAD
  MAD <- c*median(abs(x-median(x)))
}
```

Function that gets a vector 'x' and compute it's MAD.

##Reminder:

$$MAD(x) := c|x - x_{0.5}|_{0.5}$$

$$S(x) := \sqrt{\frac{\sum_{x=1}^{N}(x_i - \overline{x})^2}{N-1}}$$

part 10.b:

```
set.seed(256)   #given instruction
vec_10b <- rnorm(n = 10, mean = 1, sd = 1)
sd_10b <- sd(vec_10b)
mad_10b <- q10a(vec_10b)
```

Computes a sample of 10 observation's MAD and standard deviation of $X \sim \mathsf{norm}(\mu = 1, \sigma = 1)$.

part 10.c:

```
set.seed(256)
vec_10c <- rexp(n = 10, rate = 1)
sd_10c <- ((sd(vec_10c)))
mad_10c <- (q10a(vec_10c))
```

Computes a sample of 10 observation's MAD and standard deviation of $X \sim \mathsf{exp}(\lambda = 1)$.

part 10.d:

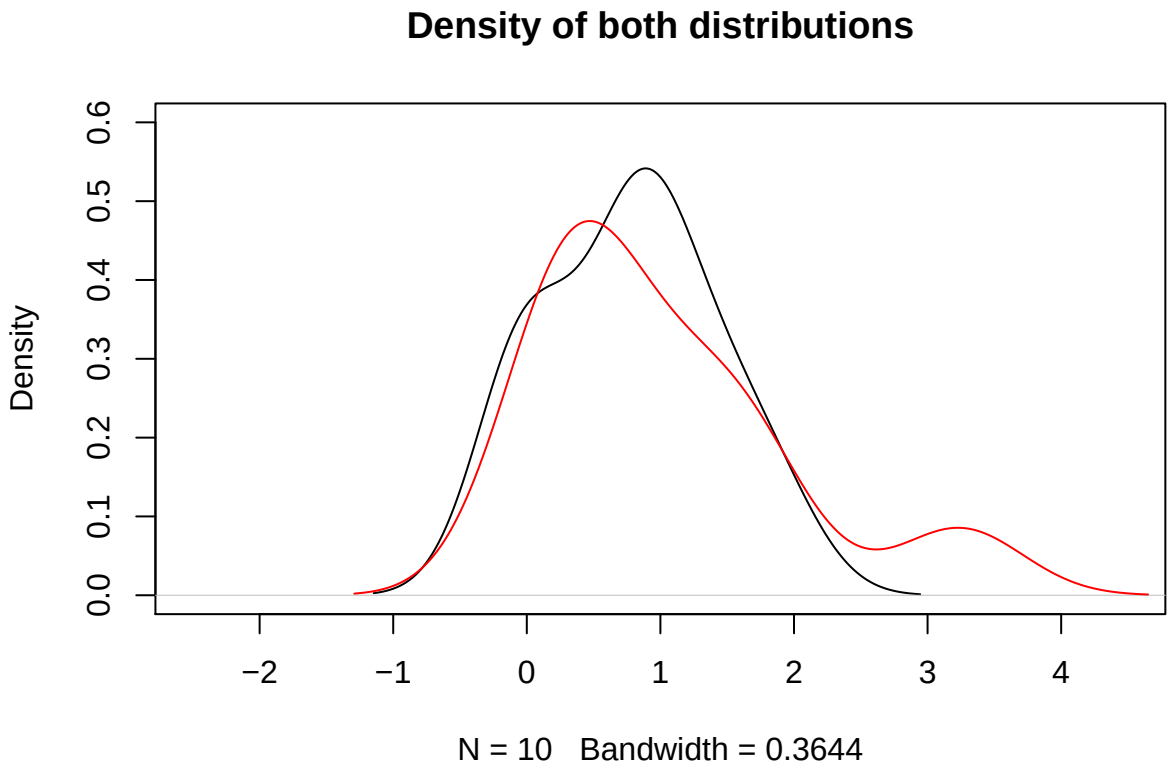
```
(diff_10b <- abs(sd_10b-mad_10b))
```

```
## [1] 0.1152187
```

```
(diff_10c <- abs(sd_10c-mad_10c))
```

```
## [1] 0.1606609
```

```
plot(density(vec_10b), xlim = c(-2.5,4.5), ylim = c(0,0.6), main = "Density of both distributions")
lines(density(vec_10c), col = "red")
```



The MAD is a robust statistic, being more resilient to outliers in a data set than the standard deviation. In the standard deviation, the distances from the mean are squared, so large deviations are weighted more heavily, and thus outliers can heavily influence it. In the MAD, the deviations of a small number of outliers are irrelevant, due to their small impact on the median. In addition, the normal distribution is symmetric in case it is not skewed, and as a result we expect the mean and median to be close to each other, unlike the exponential distribution, which has a hump shape by nature. Therefore if the values are less spread out and the distance from the median and mean are similiar, MAD and SD will be closer, which is the case in the normal distribution.

part 10.e:

```
set.seed(256)
rep <- t(replicate(1000,rnorm(10,1,1)))
sd_vec <- apply(rep,1,sd) # 1 means for rows
mad_vec <- apply(rep,1,q10a)
diff_1000normvec <- abs(sd_vec-mad_vec)
(avg_1000normvec <- mean(diff_1000normvec))
```

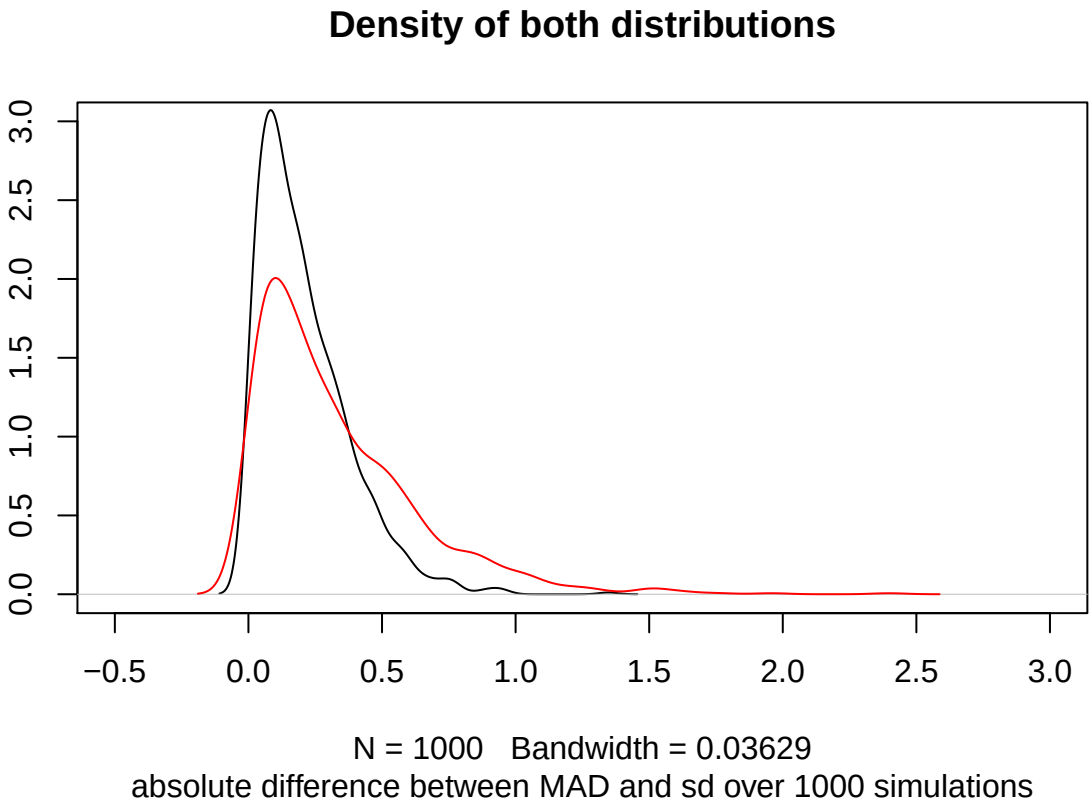
```
## [1] 0.2086996
```

```
set.seed(256)
rep2 <- t(replicate(1000,rexp(10,1)))
sd_vec2 <- apply(rep2,1,sd) # 1 means for rows
mad_vec2 <- apply(rep2,1,q10a)
diff_1000expvec <- abs(sd_vec2-mad_vec2)
(avg_1000expvec <- mean(diff_1000expvec))
```

```
## [1] 0.3282354
```

part 10.f:

```
plot(density(diff_1000normvec), xlim = c(-0.5,3), ylim = c(0,3), main = "Density of both distributions"
lines(density(diff_1000expvec), col = "red")
```



The values calculated in (d) constitute as a single observation of the simulation created in (e). We can think of (e) as a normal distribution of 1000 observations of samples like the observations in (b) and (c). Over a single sample we wouldn't necessarily see a significant result as expected between the differences between the MAD and standard deviation, but over the course of 1000 runs we expect the overall standard deviation to decrease drastically and for this reason to get a significant result regarding the difference between the statistics in both distributions.