Übung 2

Aufgaben 8 bis 13

[redacted]

21.10.2022

Aufgabe 8:

a)

```
epsilon <- 2^(-52)
1 + epsilon == 1
```

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

```
.Machine$double.eps == epsilon
```

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

b)

```
epsilon <- 2^(-53)#2^(-53) kann nicht mehr als Gleitkommazahl dargestellt werden
1 - epsilon == 1
```

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

```
.Machine$double.neg.eps == epsilon
```

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

c)

```
.Machine$double.eps/2 + 1 - 1
```

```
## [1] 0
```

```
##1 + .Machine$double.eps/2 == 1
.Machine$double.eps/2 + (1 - 1)
```

```
## [1] 1.110223e-16
```

```
##all.equal(.Machine$double.eps/2 + 1 - 1, .Machine$double.eps/2 + (1 - 1))
```

Es war gefragt, welcher Gesetz der Arithmetik verletzt wurde, sodass die zwei Zeilen im obigen beispiel nicht dasselbe Ergebnis liefern.

- Es geht uns hier um die Darstellung der Zahlen in der binären Darstellung, was heißt, dass genau 52 binäre Ziffer nach dem Komma dargestellt werden können. Wir müssen beachten, dass *.Machine$double.eps/2* eine Gleitkommazahl gleich 1.110223e-16 zurückgibt. Von dieser Zahl wird dann 1 subtrahiert. Wir müssen beachten, dass in R 1 + x = 1 -> x = 0 nicht immer stimmt. Genau in diesem Fall ist ***1 + .Machine$double.eps/2 == 1***, und wenn 1 davon abgezogen wird kommt eine Null raus. Im zweiten Beispiel wird zuerst die Klammer beachtet bevor zu ***.Machine$double.eps/2*** etwas addiert wird. Daher rechnet R zuerst mal den Wert von ***.Machine$double.eps/2*** und addiert 0 dazu danach. So kommt nur der wirkliche Wert von ***.Machine$double.eps/2*** raus.
- Wenn wir aber überprüfen wollen, ob diese zwei Werte in Wirklichkeit gleich sind, können wir Funktion *all.equal(x, y)* verwenden. Sie gibt uns in diesem Fall Auskünft darüber, dass diese zwei Zahlen **logisch** gesehen gleich sind.

Aufgabe 9:

a)

```
my_Sort <- function(x){
  for(j in (length(x) - 1):1){
    for(i in j:(length(x)-1)){
      if(x[i] > x[i + 1]){
        temp <- x[i]
        x[i] <- x[i + 1]
        x[i + 1] <- temp
      }
    }
  }
  return(x)
}
sum_for_sort <- function(x){
  sorted_vec <- sort(x)
  vec_sum <- 0
  if(length(x) >= 2){
    for(i in 1:length(sorted_vec)){
       vec_sum <- vec_sum + sorted_vec[i]
    }
  }
  return (vec_sum)
}

set.seed(1)
x <- rnorm(1e6)
print(sum_for_sort(x))
```

```
## [1] 46.90776
```

```
sum(x)
```

```
## [1] 46.90776
```

```
#install.packages("Rmpfr")
library("Rmpfr")
```

```
## Loading required package: gmp
```

```
##
## Attaching package: 'gmp'
```

```
## The following objects are masked from 'package:base':
##
##     %*%, apply, crossprod, matrix, tcrossprod
```

```
## C code of R package 'Rmpfr': GMP using 64 bits per limb
```

```
##
## Attaching package: 'Rmpfr'

## The following object is masked from 'package:gmp':
##
##     outer

## The following objects are masked from 'package:stats':
##
##     dbinom, dgamma, dnbinom, dnorm, dpois, dt, pnorm

## The following objects are masked from 'package:base':
##
##     cbind, pmax, pmin, rbind
```

```
sum(mpfr(x, 80))
```

```
## 1 'mpfr' number of precision  80   bits
## [1] 46.9077595333640895676844 69
```

- In meiner Ausarbeitung habe ich mich dafür enstchieden, Funktion zum Sortieren von Elementen, selbst zu implementieren, statt Built-In funktion sort() zu verwenden, da ich es aus der Angabe nicht klar lesen konnte, wie es zu implemenieren war. Jedoch hat der Bubble-Sort eine sehr lange Lauzeit bei 1e6 (1 000 000) Elementen. Daher ist es effektiver die Bulit-InFunktion aufzurufen. Von mir implementierter Bubble-Sort kann auf vekoten mit weniger Elementen angewendet werden.
- 80 Bit-Genaugkeit liefert ein präziseres Ergebnis, als von mir inplementierte Funktion und die Built-In Function sum(), da es hier darum geht, die Zahl möglichst genau darzustellen. Wir wählen hier 80 Bit-Genauigkeit, im Grunde genommen bedeutet, dass für unsere Zahl 80 Bit im speicher reserviert werden.

b)

```
compute_binomial_coefficients <- function(n, k){
  n_factorial <- prod(n : 1)
  k_factorial <- prod(k : 1)
  kn_factorial <- prod((n-k) : 1)

  bin_coeff <- n_factorial / (k_factorial * kn_factorial)
  return (bin_coeff)
}

print(compute_binomial_coefficients(1000, 500))
```

```
## [1] NaN
```

```
compute_binomial_coefficients2 <-function(n, k){
  n_factorial <- sum(log(1:n))
  k_factorial <- sum(log(1:k))
  kn_factorial <- sum(log(1:(n-k)))

  bin_coeff <- exp(n_factorial - (k_factorial + kn_factorial))
  return (bin_coeff)
}

print(compute_binomial_coefficients2(1000, 500))
```

```
## [1] 2.702882e+299
```

- **Anmerkung:** Im ersten Fall wird NaN zurück gegeben, da es durch ständige Produktbildung zu große Werte repräsentiert werden sollen. Diese Zahl ist nich definiert und deshalb wird NaN zurückgeliefert.

c)

```
calculate_appromaxination <- function(x0, n){
  approximation <- numeric(n + 1)
  approximation[1] <- 1
  for(i in 1:n){
    approximation[i + 1] <- (x0^i)/factorial(i)
  }
  return(sum(approximation))

}


print(calculate_appromaxination(1, 25))
```

```
## [1] 2.718282
```

```
exp(1)
```

```
## [1] 2.718282
```

```r
print(calculate_appromaxination(-25, 25))
```

```
## [1] -2834107793
```

```r
exp(-25)
```

```
## [1] 1.388794e-11
```

- Im Entwicklungspunkt 0 gilt:

$$\exp(x) = 1 + x + \frac{x^2}{2} + \ldots + \frac{x^n}{n!} \text{ -}$$

- Das betrifft den ersten Fall den wir betrachten (Entwicklungspunkt x0 = 0 und bis zur n-ten Ableitung der Taylor Entwicklung n = 25). In diesem Fall liefert die Aproximattion genau den Wert, den wir auch von der exp()-Funktion zurück bekommen. Im zweiten Fall ist das aber etwas komplizierter, da der Wert für den wir die Approximation berechnen sollen sehr klein ist. Es wird hier eine sehr kleine (aber größer NUll!) Zahl erwartet. Die Approimationsfunktion, die ich geschrieben habe, stellt aber fest, dass es zu vielen Abzugen führt, die einen negativen Wert ergeben. Gleich am anfang wird von dem Anfangswert 25 abgezogen und jedes mal wo wir einen ungeraden Exponent erwarten kommt es zu Abzüge. Hier spricht man auch von Rundungseffekten.

- Das wäre bi kleineren Werten nicht der Fall. Zum Beispiel wäle man *x0 = - 5*:

  ```r
  print(calculate_appromaxination(-5, 25))
  ```

  ```
  ## [1] 0.006737944
  ```

  ```r
  exp(-5)
  ```

  ```
  ## [1] 0.006737947
  ```

Aufgabe 10:

a)

```
#install.packages("fueleconomy")
library(fueleconomy)
my_vehices <- data.frame(vehicles)
#complete.cases(my_vehices)
```

Complite.Cases() ist eine Funktion, die für jede Zeile in unserem Data Frame angibt, ob Informationen fehlen. Unser datensatz ist hier vollständig und es fehlen keine Daten. Zur Übersichtlickeit habe ich diese zeile kommentiert, weil wir einen datensatz mit 33442 Zeilen haben, was uns insgesamt auch so viel logische Werte vorweist. Für jede Zeile, wo eine eine Information fehlt wird **FALSE** angezeigt, wenn die Zeile aber vollständig ist wird **TRUE** zrückgegeben.

b.

```
sapply(my_vehices, class)
```

```
##          id        make       model        year       class       trans
##   "numeric" "character" "character"   "numeric" "character" "character"
##       drive         cyl       displ        fuel         hwy         cty
## "character"   "numeric"   "numeric" "character"   "numeric"   "numeric"
```

c.

```
#?unique

#Anzahl der inzigartigen Werte aus der Spalte class

length(unique(my_vehices$class))
```

```
## [1] 34
```

```
#Anzahl der inzigartigen Werte aus der Spalte trans
length(unique(my_vehices$trans))
```

```
## [1] 48
```

```
#Anzahl der inzigartigen Werte aus der Spalte fuel
length(unique(my_vehices$fuel))
```

```
## [1] 13
```

d.

```
tab <- table(my_vehices$drive, my_vehices$fuel)
total_precentage <- prop.table(tab)#total precentages
row_precentages <- prop.table(tab, 1)
column_precentages <- prop.table(tab, 2)

#tab
#total_percentage
#row_percentage
#column_percentage
```

e.

```
FED <- data.frame(cyl = my_vehices$cyl, displ = my_vehices$displ,
                  hwy = my_vehices$hwy, cty = my_vehices$cty)

#FED
mean(FED$cyl, na.rm = TRUE)
```

```
## [1] 5.771867
```

```
mean(FED$displ, na.rm = TRUE)
```

```
## [1] 3.352557
```

```
mean(FED$hwy, na.rm = TRUE)
```

```
## [1] 23.55128
```

```
mean(FED$cty, na.rm = TRUE)
```

```
## [1] 17.491
```

f.

```
substrected_means <- FED - colMeans(FED, na.rm = TRUE)[col(FED[])]
```

g.

```
trimmed_means <- by(my_vehices$displ,
                    list(my_vehices$drive, my_vehices$fuel), mean,
                    trim = 0.1, na.rm = TRUE)
#trimmed_means
```

h.

```
DRIVE <- factor(ifelse(my_vehices$drive %in% c("2-Wheel Drive",
                                               "Front-Wheel Drive",
                                               "Rear-Wheel Drive"), "2WD", "4WD"))
#DRIVE
```

i.

```
my_vehices$DRIVE <-DRIVE #DRIVE wird an unseren Dataframe angehängt

REGULAR <- subset(my_vehices, fuel == 'Regular',
                  select = c(DRIVE, hwy, cty))
#REGULAR
```

Aufgabe 11:

Vorbereitung:

```
#install.packages("fivethirtyeight")
library(fivethirtyeight)
data(candy_rankings)
```

a.

```
choco_percentages <- prop.table(table(candy_rankings$chocolate))
choco_percentages
```

```
##
##     FALSE      TRUE
## 0.5647059 0.4352941
```

```
caramel_percentages <- prop.table(table(candy_rankings$caramel))
caramel_percentages
```

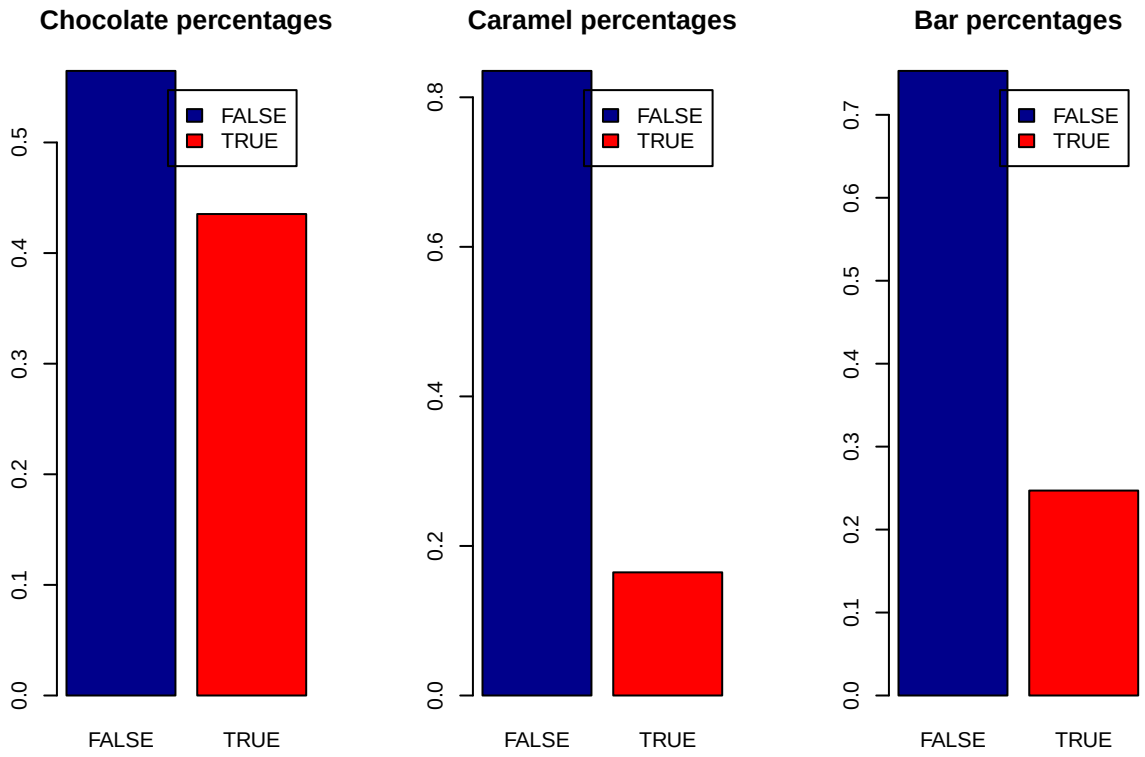
```
##
##     FALSE      TRUE
## 0.8352941 0.1647059
```

```
bar_percentages <- prop.table(table(candy_rankings$bar))
bar_percentages
```

```
##
##     FALSE      TRUE
## 0.7529412 0.2470588
```

b.

```
par(mfrow = c(1,3))
barplot(choco_percentages,
        main ="Chocolate percentages",
        col = c("darkblue", "red"), legend = rownames(choco_percentages))

barplot(caramel_percentages,
        main ="Caramel percentages",
        col = c("darkblue", "red"), legend = rownames(caramel_percentages))

barplot(bar_percentages, main ="Bar percentages",
        col = c("darkblue", "red"), legend = rownames(bar_percentages))
```


Chocolate percentages
FALSE
TRUE
0.0
0.1
0.2
0.3
0.4
0.5
FALSE
TRUE
Caramel percentages
FALSE
TRUE
0.0
0.2
0.4
0.6
0.8
FALSE
TRUE
Bar percentages
FALSE
TRUE
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
FALSE
TRUE

c.

```
tab_combinations <- table(candy_rankings$chocolate, candy_rankings$caramel,
                     candy_rankings$bar, candy_rankings$fruity,
                     candy_rankings$peanutyalmondy, candy_rankings$crispedricewafer)
#tab_combinations
```

e.

```
highest_five <- candy_rankings[order(candy_rankings$winpercent, decreasing = TRUE),]
highest_five <- head(highest_five, 5)
highest_five
```

```
##               competitorname chocolate fruity caramel peanutyalmondy nougat
## 53 Reese's Peanut Butter cup      TRUE  FALSE   FALSE           TRUE  FALSE
## 52        Reese's Miniatures      TRUE  FALSE   FALSE           TRUE  FALSE
## 80                      Twix      TRUE  FALSE    TRUE          FALSE  FALSE
## 29                   Kit Kat      TRUE  FALSE   FALSE          FALSE  FALSE
## 65                  Snickers      TRUE  FALSE    TRUE           TRUE   TRUE
##    crispedricewafer  hard   bar pluribus sugarpercent pricepercent winpercent
## 53            FALSE FALSE FALSE    FALSE        0.720        0.651   84.18029
## 52            FALSE FALSE FALSE    FALSE        0.034        0.279   81.86626
## 80             TRUE FALSE  TRUE    FALSE        0.546        0.906   81.64291
## 29             TRUE FALSE  TRUE    FALSE        0.313        0.511   76.76860
## 65            FALSE FALSE  TRUE    FALSE        0.546        0.651   76.67378
```

```
lowest_five <-candy_rankings[order(candy_rankings$winpercent),]
lowest_five <- head(lowest_five, 5)
lowest_five
```

```
##        competitorname chocolate fruity caramel peanutyalmondy nougat
## 45          Nik L Nip     FALSE   TRUE   FALSE          FALSE  FALSE
## 8  Boston Baked Beans     FALSE  FALSE   FALSE           TRUE  FALSE
## 13           Chiclets     FALSE   TRUE   FALSE          FALSE  FALSE
## 73       Super Bubble     FALSE   TRUE   FALSE          FALSE  FALSE
## 27         Jawbusters     FALSE   TRUE   FALSE          FALSE  FALSE
##    crispedricewafer  hard   bar pluribus sugarpercent pricepercent winpercent
## 45            FALSE FALSE FALSE     TRUE        0.197        0.976   22.44534
## 8             FALSE FALSE FALSE     TRUE        0.313        0.511   23.41782
## 13            FALSE FALSE FALSE     TRUE        0.046        0.325   24.52499
## 73            FALSE FALSE FALSE    FALSE        0.162        0.116   27.30386
## 27            FALSE  TRUE FALSE     TRUE        0.093        0.511   28.12744
```

```
mean(candy_rankings$winpercent)
```

```
## [1] 50.31676
```

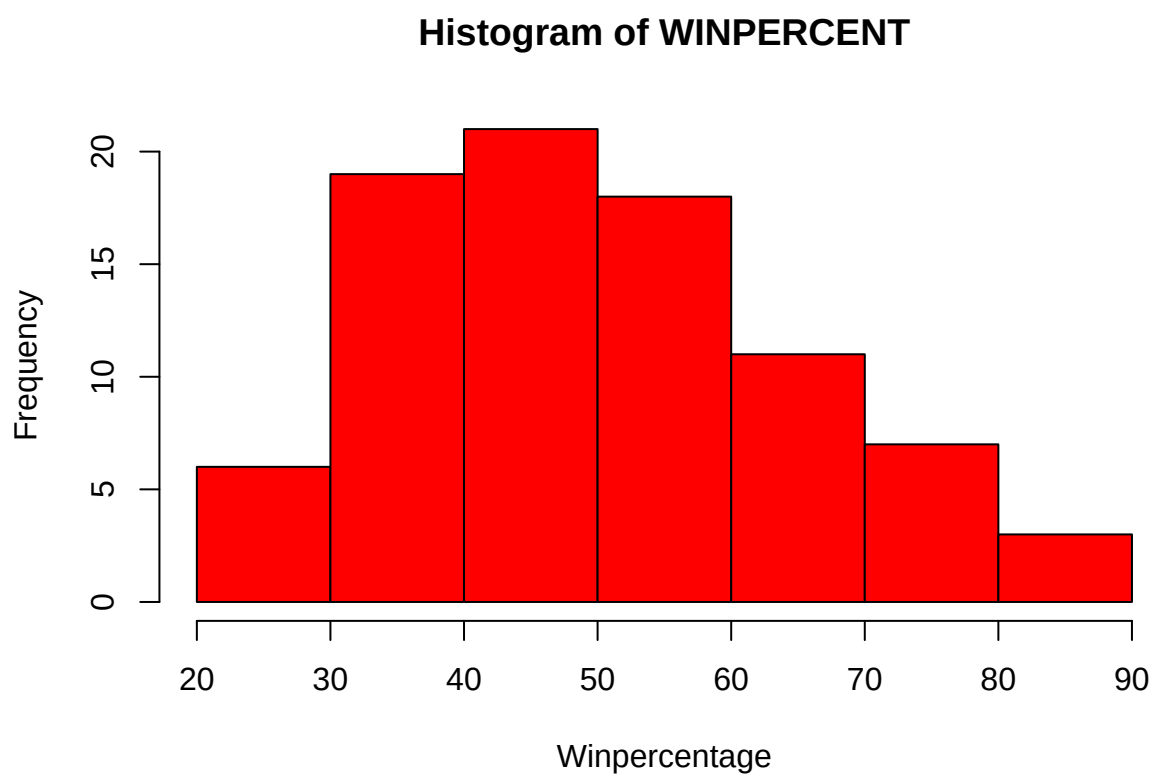
```
median(candy_rankings$winpercent)
```

```
## [1] 47.82975
```

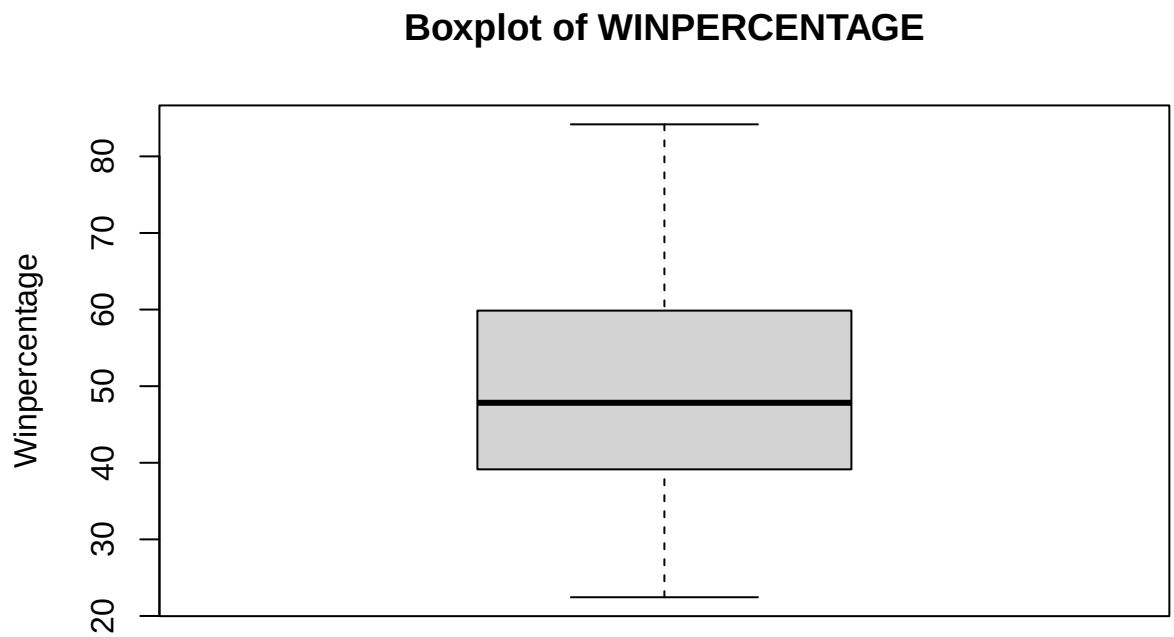
```r
sd(candy_rankings$winpercent)
```

```
## [1] 14.71436
```

```r
hist(candy_rankings$winpercent,
     main = "Histogram of WINPERCENT",
     col = "red", xlab = "Winpercentage")
```




```r
boxplot(candy_rankings$winpercent, main ="Boxplot of WINPERCENTAGE",
        ylab ="Winpercentage")
```



f.

```
#Correlation coefficient between sugar and price_percentage
cor(candy_rankings$sugarpercent, candy_rankings$pricepercent)
```

```
## [1] 0.3297064
```

```
cor(candy_rankings$sugarpercent, candy_rankings$winpercent)
```

```
## [1] 0.2291507
```

```
cor(candy_rankings$pricepercent, candy_rankings$winpercent)
```

```
## [1] 0.3453254
```

Aufgabe 12:

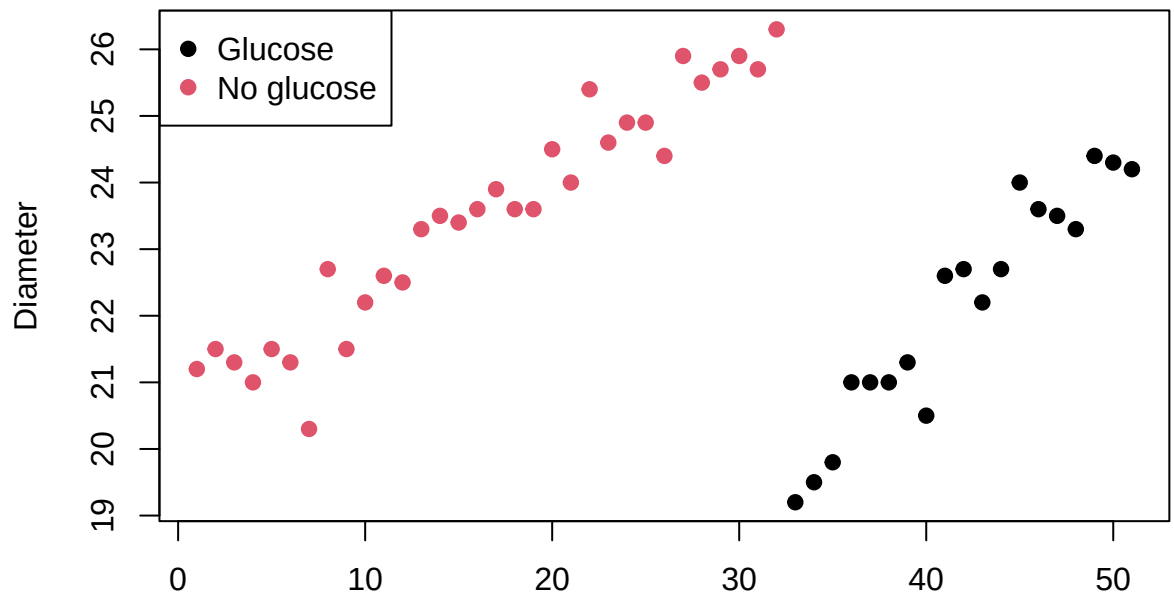
a.

```
#install.packages("ISwR")
library(ISwR)
data(hellung)
?hellung
```

```
## starting httpd help server ... done
```

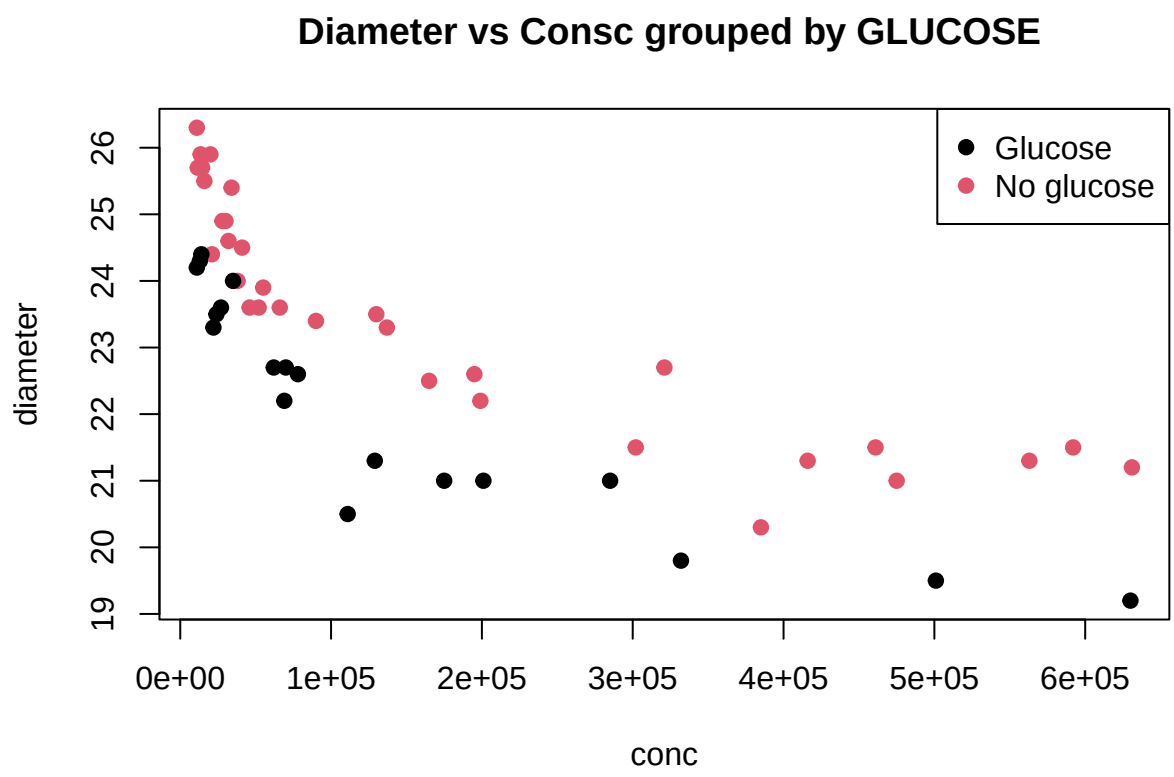
b.

```
GLUCOSE <- factor(ifelse(hellung$glucose == 1, "Yes", "No"))
hellung$GLUCOSE <- GLUCOSE
#hellung

plot (hellung$diameter,
      pch = 19,col = GLUCOSE,
      ylab ="Diameter", xlab ="")

legend("topleft",
       legend = c("Glucose", "No glucose"),
       pch = 19,
       col = factor(levels(GLUCOSE)))
```



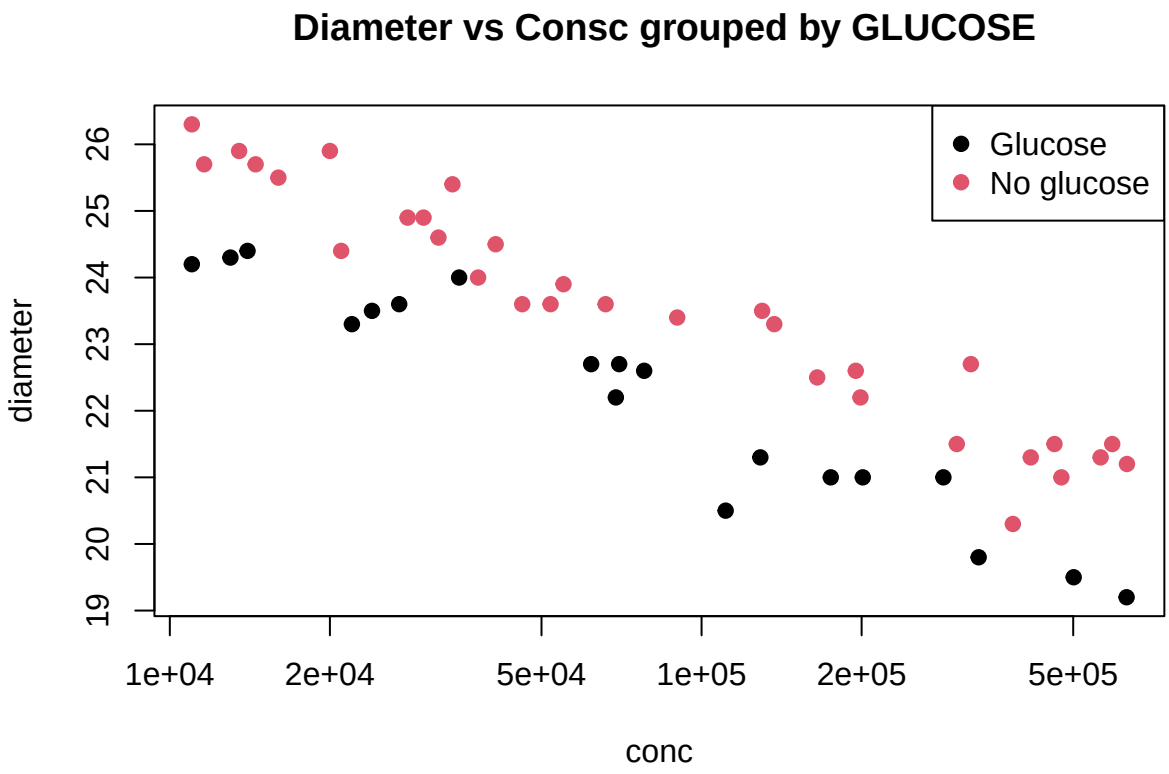
c.

```
plot(hellung$conc,
     hellung$diameter, pch = 19, col = hellung$GLUCOSE,
     main =" Diameter vs Consc grouped by GLUCOSE",
     xlab = "conc", ylab = "diameter")
legend("topright",
       legend = c("Glucose", "No glucose"),
       pch = 19,
       col = factor(levels(GLUCOSE)))
```



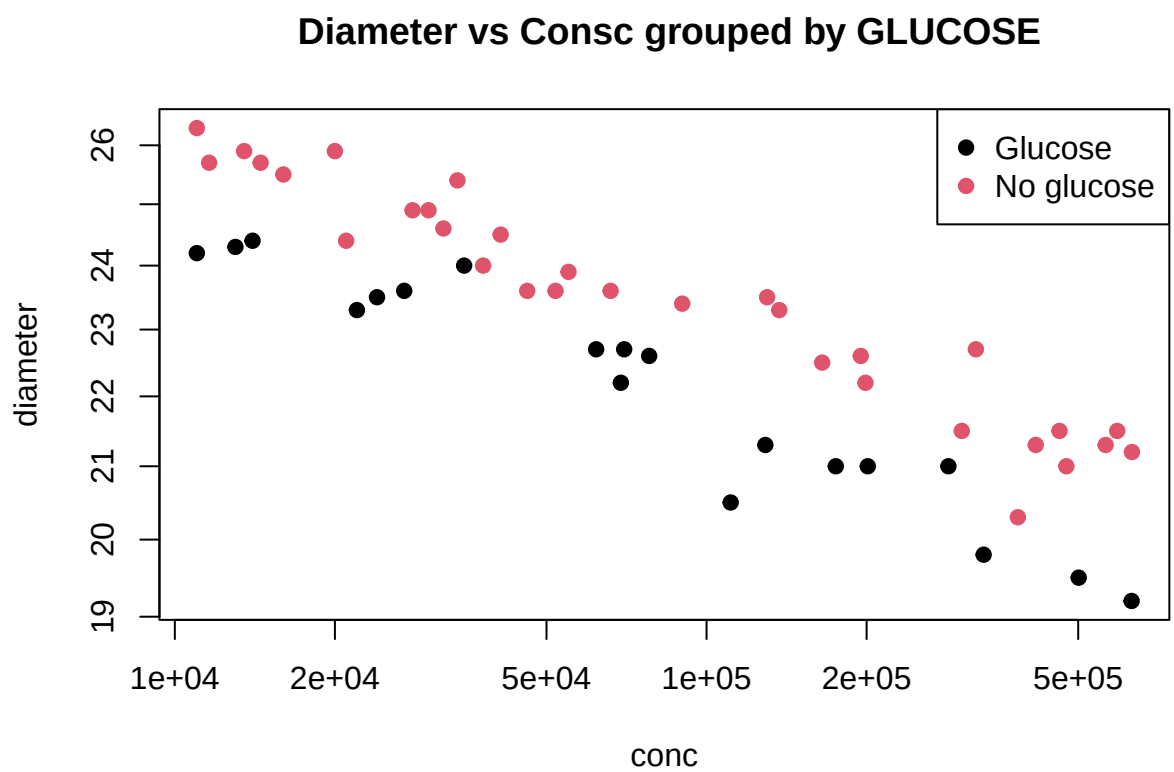
d.

```
plot(hellung$conc,
     hellung$diameter, log = "x", pch = 19, col = hellung$GLUCOSE,
     main =" Diameter vs Consc grouped by GLUCOSE",
     xlab = "conc", ylab = "diameter")
legend("topright",
       legend = c("Glucose", "No glucose"),
       pch = 19,
       col = factor(levels(GLUCOSE)))
```


Diameter vs Consc grouped by GLUCOSE
Glucose
No glucose
diameter
19
20
21
22
23
24
25
26
1e+04
2e+04
5e+04
1e+05
2e+05
5e+05
conc

e.

```
plot(hellung$conc,
     hellung$diameter, log = "xy", pch = 19, col = hellung$GLUCOSE,
     main =" Diameter vs Consc grouped by GLUCOSE",
     xlab = "conc", ylab = "diameter")
legend("topright",
       legend = c("Glucose", "No glucose"),
       pch = 19,
       col = factor(levels(GLUCOSE)))
```



f.

```
pdf(file = "MarijanaPetojevic.pdf",
    width = 7,
    height = 7)

colors <- c("#FDAE61",
            "#D9EF8B")

plot(hellung$conc,
     hellung$diameter, log = "xy", type ="b", pch = 19,
     col = colors[hellung$GLUCOSE],
     main =" Diameter vs Consc grouped by GLUCOSE",
     xlab = "conc", ylab = "diameter", mai = c(0,0,0,0))
legend("topright",
```

```r
       legend = c("Glucose", "No glucose"),
       pch = 19,
       col = colors,
       title ="Groups")

dev.off()
```

```
## pdf
##   2
```