Exercise 2

Tasks 8 to 14

21.10.2023

Contents

Task 8:

8a: IEEE 754 +

a 64bit IEEE 754 floating point number consits of: - 1 bit sign (Vorzeichen) - 11 bit exponent - 52 bit fraction

For the smallest number we need to have no sign and no exponent. This leaves us with 52bit to work with. The smallest number is when ONLY the last bit is a 1. We can accomplish this with:

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

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

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

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

8b: IEEE 754 -

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

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

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

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

8c: IEEE 754 ()

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

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

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

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

As mentioned above there can only be 52 numbers after the comma. Since we are adding an additional number somewhere, we can't save the whole number anymore. Rounding errors and the finite precision of floating-point representations lead to small differences in results, even for simple arithmetic operations.

In the second example it 1 is subtracted by 1 first which results in 0. If you add 0 to something then the value won't change

Task 9:

9a: sum_for_sort:

```r
sum_for_sort <- function(input_vector) {
  if (length(input_vector) <= 1) {
    cat("Vector must have more than one element.\n")
    return(NULL)
  }

  # Sort the input vector in ascending order
  sorted_vector <- sort(input_vector)

  # Initialize a variable to store the sum
  sum_result <- 0

  # Use a for loop to compute the sum of the sorted vector
  for (i in 1:length(sorted_vector)) {
    sum_result <- sum_result + sorted_vector[i]
  }

  return(sum_result)
}

# Example usage:
set.seed(1)
x <- rnorm(1e6)

result <- sum_for_sort(x)
cat("Sorted vector sum:", result, "\n")
```

```
## Sorted vector sum: 46.90776
```

(since this part is super slow on my PC, I disabled the execution for it)

```r
#install.packages("Rmpfr")<
#library("Rmpfr")
#set.seed(1)
#x <- rnorm(1e6)
#sum(mpfr(x, 80))
```

Output: 1 ‘mpfr’ number of precision 80 bits [1] 46.907759533364089789729074

80bit accuracy results in a more detailed result. The last digit in the first example is rounded.

9b: binomial coefficient:

```r
binomial_coefficient <- function(n, k) {

  # Compute the binomial coefficient using prod()
  result <- prod((n - k + 1):n) / prod(1:k)

  return(result)
}

n <- 1000
k <- 500

cat("The binomial coefficient", n, "over", k, "is", binomial_coefficient(n, k), "\n")
```

```
## The binomial coefficient 1000 over 500 is NaN
```

The result cant be represented since the numbers are too big. In this case a NaN is displayed.

```r
binomial_coefficient_2 <- function(n, k) {
  # factorial of n - factorial of k - factorial from n - k
  log_result <- sum(log(1:n)) - sum(log(1:k)) - sum(log(1:(n - k)))

  # Compute the exponential to get the actual binomial coefficient
  exponential <- exp(log_result)

  return(exponential)
}



n <- 1000
k <- 500

cat("The binomial coefficient", n, "over", k, "is", binomial_coefficient_2(n, k), "\n")
```

```
## The binomial coefficient 1000 over 500 is 2.702882e+299
```

```r
result
```

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

9c: e^x approximation

```
approximate_exp_taylor <- function(x0, n) {
  result <- 1  # Initialize the result with the first term of the series
  term <- 1   # Initialize the term value with the first term (n=0)

  for (i in 1:n) {
    term <- term * (x0 / i)  # Calculate the next term based on the previous one
    result <- result + term  # Add the term to the result
  }

  return(result)
}



x0 <- 1  # Value at start with which to approximate e^x
terms <- 25  # Number of terms in the Taylor expansion

cat("Approximation of e^", x0, " using", terms, "terms:", approximate_exp_taylor(x0, terms), "\n")
```

```
## Approximation of e^ 1  using 25 terms: 2.718282
```

```
cat("Approximation of e^", x0, " using", terms, "terms:", exp(x0), "\n")
```

```
## Approximation of e^ 1  using 25 terms: 2.718282
```

```
x0 <- -25

cat("Approximation of e^", x0, " using", terms, "terms:", approximate_exp_taylor(x0, terms), "\n")
```

```
## Approximation of e^ -25  using 25 terms: -2834107793
```

```
cat("Approximation of e^", x0, " using", terms, "terms:", exp(x0), "\n")
```

```
## Approximation of e^ -25  using 25 terms: 1.388794e-11
```

There is no difference between using my function or exp() as long as the values are positive. But things change when the values are new the 0 coordinate.

The first result is a result of applying the Taylor series for e^x at x=0. The Taylor series expansion for e^x is centered at 0, and it converges most effectively near that point. Because of that, approximating e^-25 using this approach doesn't work well.

Taylor series expansions are most accurate near their center, and they converge less effectively as you move further away from that center. That's why the result of my function differs so much from built in one.

> Quote: "The exp() function in R accurately calculates the value of e raised to the power of x0, e^x0. This function is based on well-established and highly accurate numerical algorithms that provide a precise result. It is a direct and correct way to compute the exponential function for any real number, including e^-25."

Task 10:

10a: install fueleconomy

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

complete.cases(my_vehices) is a function that checks if there is missing data for every column in the dataset.

It seems like in our dataset is no data missing. If any data would be missing, then the the function would print FALSE. Since there are too many TRUEs, I commented the line out so that the document won't be that long

10b: optain types:

```
#sapply(vehices, class)
```

10c: unique valuse:

```
# Count the number of unique values for each variable
unique_class <- length(unique(vehicles$class))
unique_trans <- length(unique(vehicles$trans))
unique_drive <- length(unique(vehicles$drive))
unique_fuel <- length(unique(vehicles$fuel))

# Display the counts
cat("Number of unique values for 'class':", unique_class, "\n")
```

```
## Number of unique values for 'class': 34
```

```
cat("Number of unique values for 'trans':", unique_trans, "\n")
```

```
## Number of unique values for 'trans': 48
```

```
cat("Number of unique values for 'drive':", unique_drive, "\n")
```

```
## Number of unique values for 'drive': 7
```

```
cat("Number of unique values for 'fuel':", unique_fuel, "\n")
```

```
## Number of unique values for 'fuel': 13
```

10d: frequency of table:

```
# Create a frequency table of drive vs fuel
frequency_table <- table(vehicles$drive, vehicles$fuel)

total_count <- sum(frequency_table)
column_percentages <- prop.table(frequency_table, 2) * 100
row_percentages <- prop.table(frequency_table, 1) * 100
total_percentages <- (frequency_table / total_count) * 100

# Display the frequency table, total count, column percentages, and row percentages
print("Frequency Table (Drive vs. Fuel):")
```

```
## [1] "Frequency Table (Drive vs. Fuel):"
```

```
print(frequency_table)
```

```
##
##                               CNG Diesel Electricity Gasoline or E85
##   2-Wheel Drive                 0     72          14               0
##   4-Wheel Drive                 0     21           0              93
##   4-Wheel or All-Wheel Drive    2    225           0             147
##   All-Wheel Drive               0     30           0              97
##   Front-Wheel Drive            23    200          25             238
##   Part-time 4-Wheel Drive       0      0           0              28
##   Rear-Wheel Drive             33    326          16             440
##
##                              Gasoline or natural gas Gasoline or propane
##   2-Wheel Drive                                    0                   0
##   4-Wheel Drive                                    0                   0
##   4-Wheel or All-Wheel Drive                       4                   4
##   All-Wheel Drive                                  0                   0
##   Front-Wheel Drive                                5                   0
##   Part-time 4-Wheel Drive                          0                   0
##   Rear-Wheel Drive                                 9                   4
##
##                              Midgrade Premium Premium and Electricity
##   2-Wheel Drive                     0       1                       0
##   4-Wheel Drive                    12     223                       0
##   4-Wheel or All-Wheel Drive        0    1480                       0
##   All-Wheel Drive                   6     679                       0
##   Front-Wheel Drive                 0    2088                       0
##   Part-time 4-Wheel Drive           0       8                       0
##   Rear-Wheel Drive                 25    4138                       1
##
##                              Premium Gas or Electricity Premium or E85 Regular
##   2-Wheel Drive                                       0              0     420
##   4-Wheel Drive                                       0             18     332
##   4-Wheel or All-Wheel Drive                          0              1    4784
##   All-Wheel Drive                                     1             32     422
##   Front-Wheel Drive                                   5              0    9641
##   Part-time 4-Wheel Drive                             0              0      60
##   Rear-Wheel Drive                                    1             37    6963
##
##                              Regular Gas and Electricity
##   2-Wheel Drive                                      0
##   4-Wheel Drive                                      0
##   4-Wheel or All-Wheel Drive                         0
##   All-Wheel Drive                                    0
##   Front-Wheel Drive                                  8
##   Part-time 4-Wheel Drive                            0
##   Rear-Wheel Drive                                   0
```

```
cat("\nTotal Count (Grand Total):", total_count, "\n\n")
```

```
##
## Total Count (Grand Total): 33442
```

```
print("Column Percentages:")
```

```
## [1] "Column Percentages:"
```

```
print(column_percentages)
```

```
##
##                                      CNG       Diesel  Electricity
##   2-Wheel Drive               0.00000000   8.23798627  25.45454545
##   4-Wheel Drive               0.00000000   2.40274600   0.00000000
##   4-Wheel or All-Wheel Drive  3.44827586  25.74370709   0.00000000
##   All-Wheel Drive             0.00000000   3.43249428   0.00000000
##   Front-Wheel Drive          39.65517241  22.88329519  45.45454545
##   Part-time 4-Wheel Drive     0.00000000   0.00000000   0.00000000
##   Rear-Wheel Drive           56.89655172  37.29977117  29.09090909
##
##                             Gasoline or E85 Gasoline or natural gas
##   2-Wheel Drive                  0.00000000              0.00000000
##   4-Wheel Drive                  8.91658677              0.00000000
##   4-Wheel or All-Wheel Drive    14.09395973             22.22222222
##   All-Wheel Drive                9.30009588              0.00000000
##   Front-Wheel Drive             22.81879195             27.77777778
##   Part-time 4-Wheel Drive        2.68456376              0.00000000
##   Rear-Wheel Drive              42.18600192             50.00000000
##
##                             Gasoline or propane     Midgrade      Premium
##   2-Wheel Drive                      0.00000000   0.00000000   0.01160497
##   4-Wheel Drive                      0.00000000  27.90697674   2.58790762
##   4-Wheel or All-Wheel Drive        50.00000000   0.00000000  17.17535105
##   All-Wheel Drive                    0.00000000  13.95348837   7.87977254
##   Front-Wheel Drive                  0.00000000   0.00000000  24.23117094
##   Part-time 4-Wheel Drive            0.00000000   0.00000000   0.09283974
##   Rear-Wheel Drive                  50.00000000  58.13953488  48.02135314
##
##                             Premium and Electricity Premium Gas or Electricity
##   2-Wheel Drive                          0.00000000                 0.00000000
##   4-Wheel Drive                          0.00000000                 0.00000000
##   4-Wheel or All-Wheel Drive             0.00000000                 0.00000000
##   All-Wheel Drive                        0.00000000                14.28571429
##   Front-Wheel Drive                      0.00000000                71.42857143
##   Part-time 4-Wheel Drive                0.00000000                 0.00000000
##   Rear-Wheel Drive                     100.00000000                14.28571429
##
```

```
##                              Premium or E85     Regular
##   2-Wheel Drive                  0.00000000  1.85659977
##   4-Wheel Drive                 20.45454545  1.46759791
##   4-Wheel or All-Wheel Drive     1.13636364 21.14755548
##   All-Wheel Drive               36.36363636  1.86544072
##   Front-Wheel Drive              0.00000000 42.61780568
##   Part-time 4-Wheel Drive        0.00000000  0.26522854
##   Rear-Wheel Drive              42.04545455 30.77977190
##
##                              Regular Gas and Electricity
##   2-Wheel Drive                               0.00000000
##   4-Wheel Drive                               0.00000000
##   4-Wheel or All-Wheel Drive                  0.00000000
##   All-Wheel Drive                             0.00000000
##   Front-Wheel Drive                         100.00000000
##   Part-time 4-Wheel Drive                     0.00000000
##   Rear-Wheel Drive                            0.00000000
```

```r
cat("\n")
```

```r
print("Row Percentages:")
```

```
## [1] "Row Percentages:"
```

```r
print(row_percentages)
```

```
##
##                                       CNG       Diesel  Electricity
##   2-Wheel Drive               0.000000000 14.201183432  2.761341223
##   4-Wheel Drive               0.000000000  3.004291845  0.000000000
##   4-Wheel or All-Wheel Drive  0.030088762  3.384985708  0.000000000
##   All-Wheel Drive             0.000000000  2.367797948  0.000000000
##   Front-Wheel Drive           0.188016022  1.634921932  0.204365242
##   Part-time 4-Wheel Drive     0.000000000  0.000000000  0.000000000
##   Rear-Wheel Drive            0.275160510  2.718252314  0.133411157
##
##                              Gasoline or E85 Gasoline or natural gas
##   2-Wheel Drive                  0.000000000             0.000000000
##   4-Wheel Drive                 13.304721030             0.000000000
##   4-Wheel or All-Wheel Drive     2.211523996             0.060177524
##   All-Wheel Drive                7.655880032             0.000000000
##   Front-Wheel Drive              1.945557100             0.040873048
##   Part-time 4-Wheel Drive       29.166666667             0.000000000
##   Rear-Wheel Drive               3.668806804             0.075043776
##
##                              Gasoline or propane     Midgrade      Premium
##   2-Wheel Drive                      0.000000000  0.000000000  0.197238659
##   4-Wheel Drive                      0.000000000  1.716738197 31.902718169
##   4-Wheel or All-Wheel Drive         0.060177524  0.000000000 22.265683767
##   All-Wheel Drive                    0.000000000  0.473559590 53.591160221
##   Front-Wheel Drive                  0.000000000  0.000000000 17.068584975
##   Part-time 4-Wheel Drive            0.000000000  0.000000000  8.333333333
##   Rear-Wheel Drive                   0.033352789  0.208454932 34.503460352
##
##                              Premium and Electricity Premium Gas or Electricity
##   2-Wheel Drive                          0.000000000                0.000000000
##   4-Wheel Drive                         0.000000000               0.000000000
##   4-Wheel or All-Wheel Drive            0.000000000               0.000000000
##   All-Wheel Drive                       0.000000000               0.078926598
##   Front-Wheel Drive                     0.000000000               0.040873048
##   Part-time 4-Wheel Drive               0.000000000               0.000000000
##   Rear-Wheel Drive                      0.008338197               0.008338197
##
##                              Premium or E85      Regular
##   2-Wheel Drive                 0.000000000 82.840236686
##   4-Wheel Drive                 2.575107296 47.496423462
##   4-Wheel or All-Wheel Drive    0.015044381 71.972318339
##   All-Wheel Drive               2.525651144 33.307024467
##   Front-Wheel Drive             0.000000000 78.811411755
##   Part-time 4-Wheel Drive       0.000000000 62.500000000
##   Rear-Wheel Drive              0.308513299 58.058867673
##
##                              Regular Gas and Electricity
##   2-Wheel Drive                              0.000000000
##   4-Wheel Drive                              0.000000000
##   4-Wheel or All-Wheel Drive                 0.000000000
##   All-Wheel Drive                            0.000000000
##   Front-Wheel Drive                          0.065396877
##   Part-time 4-Wheel Drive                    0.000000000
##   Rear-Wheel Drive                           0.000000000
```

```r
print("Total Percentages:")
```

```
## [1] "Total Percentages:"
```

```r
print(total_percentages)
```

```
##
##                                      CNG       Diesel  Electricity
##   2-Wheel Drive                0.000000000  0.215298128  0.041863525
##   4-Wheel Drive                0.000000000  0.062795287  0.000000000
##   4-Wheel or All-Wheel Drive   0.005980504  0.672806650  0.000000000
##   All-Wheel Drive              0.000000000  0.089707553  0.000000000
##   Front-Wheel Drive            0.068775791  0.598050356  0.074756294
##   Part-time 4-Wheel Drive      0.000000000  0.000000000  0.000000000
##   Rear-Wheel Drive             0.098678309  0.974822080  0.047844028
##
##                              Gasoline or E85 Gasoline or natural gas
##   2-Wheel Drive                  0.000000000             0.000000000
##   4-Wheel Drive                  0.278093415             0.000000000
##   4-Wheel or All-Wheel Drive     0.439567012             0.011961007
##   All-Wheel Drive                0.290054423             0.000000000
##   Front-Wheel Drive              0.711679923             0.014951259
##   Part-time 4-Wheel Drive        0.083727050             0.000000000
##   Rear-Wheel Drive               1.315710783             0.026912266
##
##                              Gasoline or propane     Midgrade      Premium
##   2-Wheel Drive                      0.000000000  0.000000000  0.002990252
##   4-Wheel Drive                      0.000000000  0.035883021  0.666826147
##   4-Wheel or All-Wheel Drive         0.011961007  0.000000000  4.425572633
##   All-Wheel Drive                    0.000000000  0.017941511  2.030380958
##   Front-Wheel Drive                0.000000000  0.000000000  6.243645715
##   Part-time 4-Wheel Drive          0.000000000  0.000000000  0.023922014
##   Rear-Wheel Drive                 0.011961007  0.074756294 12.373661862
##
##                              Premium and Electricity Premium Gas or Electricity
##   2-Wheel Drive                            0.000000000                0.000000000
##   4-Wheel Drive                            0.000000000                0.000000000
##   4-Wheel or All-Wheel Drive               0.000000000                0.000000000
##   All-Wheel Drive                          0.000000000                0.002990252
##   Front-Wheel Drive                        0.000000000                0.014951259
##   Part-time 4-Wheel Drive                  0.000000000                0.000000000
##   Rear-Wheel Drive                         0.002990252                0.002990252
##
##                              Premium or E85      Regular
##   2-Wheel Drive                 0.000000000  1.255905747
##   4-Wheel Drive                 0.053824532  0.992763591
##   4-Wheel or All-Wheel Drive    0.002990252 14.305364512
##   All-Wheel Drive               0.095688057  1.261886251
##   Front-Wheel Drive             0.000000000 28.829017403
##   Part-time 4-Wheel Drive       0.000000000  0.179415107
##   Rear-Wheel Drive              0.110639316 20.821123139
##
##                              Regular Gas and Electricity
##   2-Wheel Drive                              0.000000000
##   4-Wheel Drive                              0.000000000
##   4-Wheel or All-Wheel Drive                 0.000000000
##   All-Wheel Drive                            0.000000000
##   Front-Wheel Drive                          0.023922014
##   Part-time 4-Wheel Drive                    0.000000000
##   Rear-Wheel Drive                           0.000000000
```

10e: FED variable

```
# Create the FED data frame with selected variables and remove missing values
FED <- data.frame(
  cyl = vehicles$cyl,
  displ = vehicles$displ,
  hwy = vehicles$hwy,
  cty = vehicles$cty
)

# Compute the medians for each variable
medians <- data.frame(
  median_cyl = median(FED$cyl, na.rm = TRUE),
  median_displ = median(FED$displ, na.rm = TRUE),
  median_hwy = median(FED$hwy, na.rm = TRUE),
  median_cty = median(FED$cty, na.rm = TRUE)
)

# Display the medians
print(medians)
```

```
##   median_cyl median_displ median_hwy median_cty
## 1          6            3         23         17
```

10f: subtract median

```
# Subtract the medians from the variables in FED
FED_subtracted <- FED
FED_subtracted$cyl <- FED_subtracted$cyl - medians$median_cyl
FED_subtracted$displ <- FED_subtracted$displ - medians$median_displ
FED_subtracted$hwy <- FED_subtracted$hwy - medians$median_hwy
FED_subtracted$cty <- FED_subtracted$cty - medians$median_cty

# Display FED with medians subtracted:
#print(FED_subtracted) #I will comment this, because it added 400+ pages to the PDF lol
```

10g: trimmed means:

```
# Compute the trimmed means for displ by drive and fuel with a trim level of 0.1
trimmed_means <- by(vehicles$displ, list(vehicles$drive, vehicles$fuel), mean, trim = 0.1, na.rm = TRUE

# Display the trimmed means
print("Trimmed Means for displ by drive and fuel:")
```

```
## [1] "Trimmed Means for displ by drive and fuel:"
```

```
# print(trimmed_means) #I will comment this again, because it added another 400+ pages to the PDF
```

10f: factor DRIVE;

```
# Create the DRIVE factor variable based on conditions
DRIVE <- factor(ifelse(vehicles$drive %in% c("2-Wheel Drive", "Front-Wheel Drive", "Rear-Wheel Drive"),
#DRIVE

# Display the first few rows of the updated dataset to verify the DRIVE variable
head(vehicles)
```

```
##      id  make       model year           class           trans
## 1 13309 Acura 2.2CL/3.0CL 1997 Subcompact Cars Automatic 4-spd
## 2 13310 Acura 2.2CL/3.0CL 1997 Subcompact Cars    Manual 5-spd
## 3 13311 Acura 2.2CL/3.0CL 1997 Subcompact Cars Automatic 4-spd
## 4 14038 Acura 2.3CL/3.0CL 1998 Subcompact Cars Automatic 4-spd
## 5 14039 Acura 2.3CL/3.0CL 1998 Subcompact Cars    Manual 5-spd
## 6 14040 Acura 2.3CL/3.0CL 1998 Subcompact Cars Automatic 4-spd
##               drive cyl displ    fuel hwy cty
## 1 Front-Wheel Drive   4   2.2 Regular  26  20
## 2 Front-Wheel Drive   4   2.2 Regular  28  22
## 3 Front-Wheel Drive   6   3.0 Regular  26  18
## 4 Front-Wheel Drive   4   2.3 Regular  27  19
## 5 Front-Wheel Drive   4   2.3 Regular  29  21
## 6 Front-Wheel Drive   6   3.0 Regular  26  17
```

10i: REGULAR data frame

```
# Create the REGULAR data frame for "Regular" fuel type
#REGULAR <- subset(vehicles, fuel == "Regular", select = c(DRIVE, hwy, cty))

# Display the first few rows of the REGULAR data frame
#head(REGULAR)
```

I am commenting this block because of the extremely high knitting times this block produces

Task 11:

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

```
## Some larger datasets need to be installed separately, like senators and
## house_district_forecast. To install these, we recommend you install the
## fivethirtyeightdata package by running:
## install.packages('fivethirtyeightdata', repos =
## 'https://fivethirtyeightdata.github.io/drat/', type = 'source')
```

```
data(candy_rankings)

#candy_rankings
```

11a: univariate distribution:

```
# Compute the percentage of candies in each level of the 'chocolate' then 'carmel' and then 'bar' varia
chocolate_distribution <- prop.table(table(candy_rankings$chocolate))
caramel_distribution <- prop.table(table(candy_rankings$caramel))
bar_distribution <- prop.table(table(candy_rankings$bar))

print("Percentage distribution of chocolate:")
```

```
## [1] "Percentage distribution of chocolate:"
```

```
print(chocolate_distribution)
```

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

```
print("Percentage distribution of caramel:")
```

```
## [1] "Percentage distribution of caramel:"
```

```
print(caramel_distribution)
```

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

```
print("Percentage distribution of bar:")
```

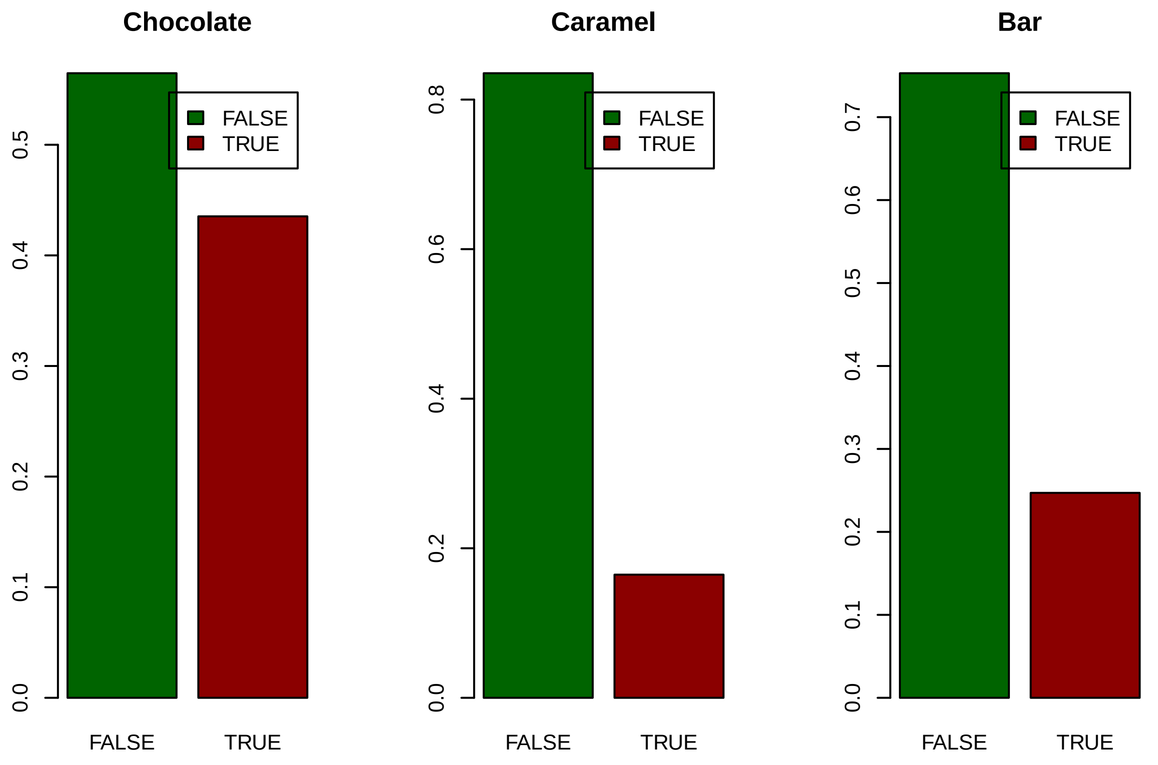
```
## [1] "Percentage distribution of bar:"
```

```
print(bar_distribution)
```

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

11b: Bar-Plot

```r
# Create a layout with three plots in one row
par(mfrow = c(1, 3))

# Create a barplot for 'chocolate' then 'carmel' and then 'bar' with percentages and title
barplot(chocolate_distribution, main = "Chocolate", legend = rownames(chocolate_distribution), col = c(
barplot(caramel_distribution, main = "Caramel", legend = rownames(caramel_distribution), col = c("darkg
barplot(bar_distribution, main = "Bar", legend = rownames(bar_distribution), col = c("darkgreen", "dark
```



```r
# Reset the layout
par(mfrow = c(1, 1))
```

11c: number of candies

```
# Select the specified variables
selected_vars <- c("chocolate", "caramel", "bar", "fruity", "peanutyalmondy", "crispedricewafer")

# Generate a table of the number of candies for each factor level combination
candy_counts <- aggregate(count ~ ., data = data.frame(candy_rankings[selected_vars], count = 1), FUN =

# Order the resulting table in decreasing order by the number of candies in each group
candy_counts <- candy_counts[order(candy_counts$count, decreasing = TRUE), ]

# Display the table
print(candy_counts)
```

```
##    chocolate caramel   bar fruity peanutyalmondy crispedricewafer count
## 7      FALSE   FALSE FALSE   TRUE          FALSE            FALSE    36
## 2       TRUE   FALSE FALSE  FALSE          FALSE            FALSE     7
## 1      FALSE   FALSE FALSE  FALSE          FALSE            FALSE     6
## 5       TRUE   FALSE  TRUE  FALSE          FALSE            FALSE     6
## 11      TRUE   FALSE FALSE  FALSE           TRUE            FALSE     6
## 3      FALSE    TRUE FALSE  FALSE          FALSE            FALSE     3
## 6       TRUE    TRUE  TRUE  FALSE          FALSE            FALSE     3
## 13      TRUE   FALSE  TRUE  FALSE           TRUE            FALSE     3
## 16      TRUE   FALSE  TRUE  FALSE          FALSE             TRUE     3
## 4       TRUE    TRUE FALSE  FALSE          FALSE            FALSE     2
## 14      TRUE    TRUE  TRUE  FALSE           TRUE            FALSE     2
## 17      TRUE    TRUE  TRUE  FALSE          FALSE             TRUE     2
## 8       TRUE   FALSE FALSE   TRUE          FALSE            FALSE     1
## 9      FALSE    TRUE FALSE   TRUE          FALSE            FALSE     1
## 10     FALSE   FALSE FALSE  FALSE           TRUE            FALSE     1
## 12     FALSE   FALSE  TRUE  FALSE           TRUE            FALSE     1
## 15      TRUE   FALSE FALSE  FALSE          FALSE             TRUE     1
## 18      TRUE    TRUE  TRUE  FALSE           TRUE             TRUE     1
```

Notes for the Übung:

"aggregate()": This is a function in R used for aggregating data. It takes a dataset, groups the data by specified factors or variables, and then applies a given function (in this case, sum) to the grouped data.

"count ~ ." specifies the formula for aggregation. It tells aggregate() to aggregate the variable named "count" by all other variables (indicated by the dot .). In this context, "count" is a new variable that will be created to store the counts for each factor level combination.

FUN = sum specifies that sum function that is being applied during aggregation

11d: create table: NOTE: this is wrong. I don't know what I have to do there…

```r
# srot by count
candy_counts <- candy_counts[order(candy_counts$count, decreasing = TRUE), ]

# Select the specified variables
# Extract the factor levels (variable names) for each row
taste_profiles <- apply(candy_counts[, -ncol(candy_counts)], 1, function(row) {
  factors <- names(row)[as.logical(row)]
  if (length(factors) == 0) {
    return("")
  } else {
    return(paste(factors, collapse = ", "))
  }
})

# Create a new data frame with the taste profile and the number of observations
result_df <- data.frame(TasteProfile = taste_profiles, Observations = candy_counts$count)

# Display the result data frame
print("Taste Profiles and Observations:")
```

```
## [1] "Taste Profiles and Observations:"
```

```r
print(result_df)
```

```
##                                                  TasteProfile Observations
## 7                                                      fruity           36
## 2                                                   chocolate            7
## 1                                                                        6
## 5                                              chocolate, bar            6
## 11                                 chocolate, peanutyalmondy            6
## 3                                                     caramel            3
## 6                                     chocolate, caramel, bar            3
## 13                            chocolate, bar, peanutyalmondy            3
## 16                          chocolate, bar, crispedricewafer            3
## 4                                          chocolate, caramel            2
## 14                   chocolate, caramel, bar, peanutyalmondy            2
## 17                 chocolate, caramel, bar, crispedricewafer            2
## 8                                           chocolate, fruity            1
## 9                                             caramel, fruity            1
## 10                                             peanutyalmondy            1
## 12                                        bar, peanutyalmondy            1
## 15                                chocolate, crispedricewafer            1
## 18 chocolate, caramel, bar, peanutyalmondy, crispedricewafer            1
```

11e: inspect winpercent variable

```r
# Inspect the top 5 candies based on winpercent
top_5_candies <- head(candy_rankings[order(candy_rankings$winpercent, decreasing = TRUE), ], 5)
#Top 5 Candies:
print(top_5_candies)
```

```
##               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
```

```r
# Inspect the bottom 5 candies based on winpercent
bottom_5_candies <- head(candy_rankings[order(candy_rankings$winpercent), ], 5)
#Bottom 5 Candies:
print(bottom_5_candies)
```

```
##       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
```

```r
# Calculate the mean, median, and standard deviation of winpercent
mean_winpercent <- mean(candy_rankings$winpercent)
median_winpercent <- median(candy_rankings$winpercent)
std_dev_winpercent <- sd(candy_rankings$winpercent)

print(paste("Mean of winpercent:", mean_winpercent))
```

```
## [1] "Mean of winpercent: 50.3167638117647"
```

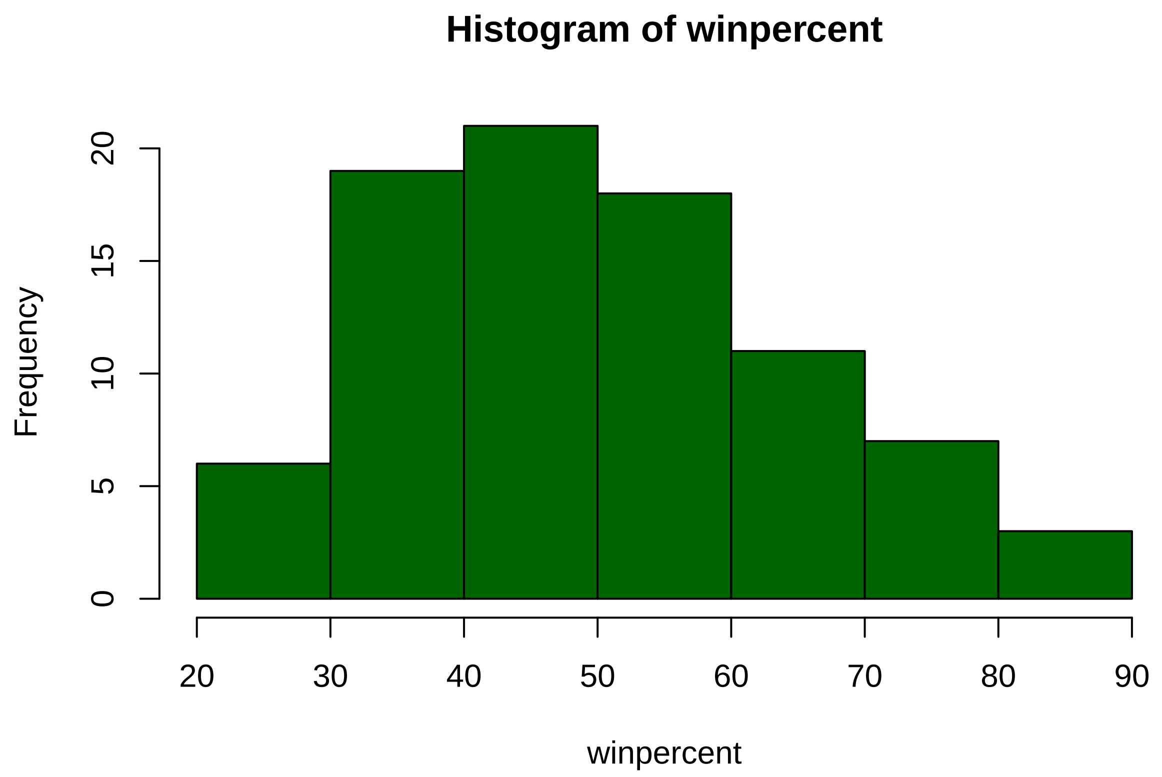
```r
print(paste("Median of winpercent:", median_winpercent))
```

```
## [1] "Median of winpercent: 47.829754"
```

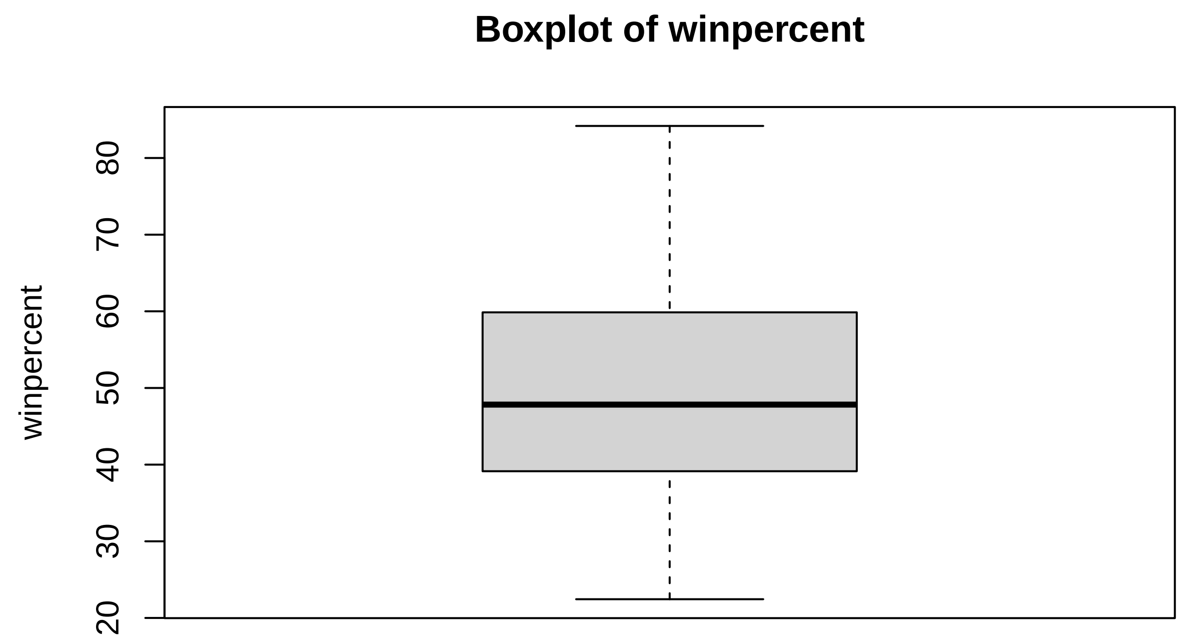
```r
print(paste("Standard Deviation of winpercent:", std_dev_winpercent))
```

```
## [1] "Standard Deviation of winpercent: 14.7143574134078"
```

```r
# Create a histogram and boxplot of winpercent:
hist(candy_rankings$winpercent, main = "Histogram of winpercent", xlab = "winpercent", col = "darkgreen
```



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



11f: compute correlation

```r
# Select the quantitative variables
quantitative_vars <- candy_rankings[, c("sugarpercent", "pricepercent", "winpercent")]

# Compute the correlation matrix for the quantitative variables
correlation_matrix <- cor(quantitative_vars)

# Display the correlation matrix
print("Correlation Matrix:")
```

```
## [1] "Correlation Matrix:"
```

```r
print(correlation_matrix)
```

```
##              sugarpercent pricepercent winpercent
## sugarpercent    1.0000000    0.3297064  0.2291507
## pricepercent    0.3297064    1.0000000  0.3453254
## winpercent      0.2291507    0.3453254  1.0000000
```

```r
# Discuss the correlation
cat("\nCorrelation between sugarpercent and pricepercent:", correlation_matrix["sugarpercent", "pricepe
```

```
##
## Correlation between sugarpercent and pricepercent: 0.3297064
```

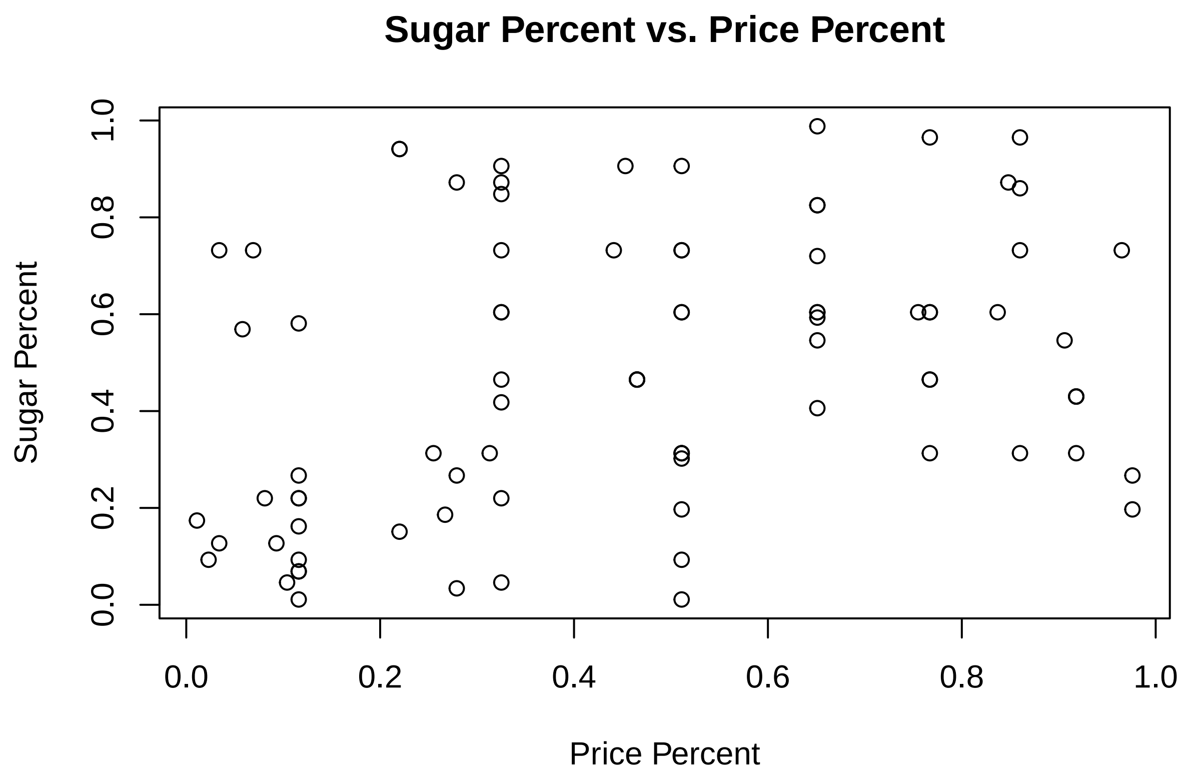
```
cat("Correlation between winpercent and sugarpercent:", correlation_matrix["winpercent", "sugarpercent"
```

```
## Correlation between winpercent and sugarpercent: 0.2291507
```

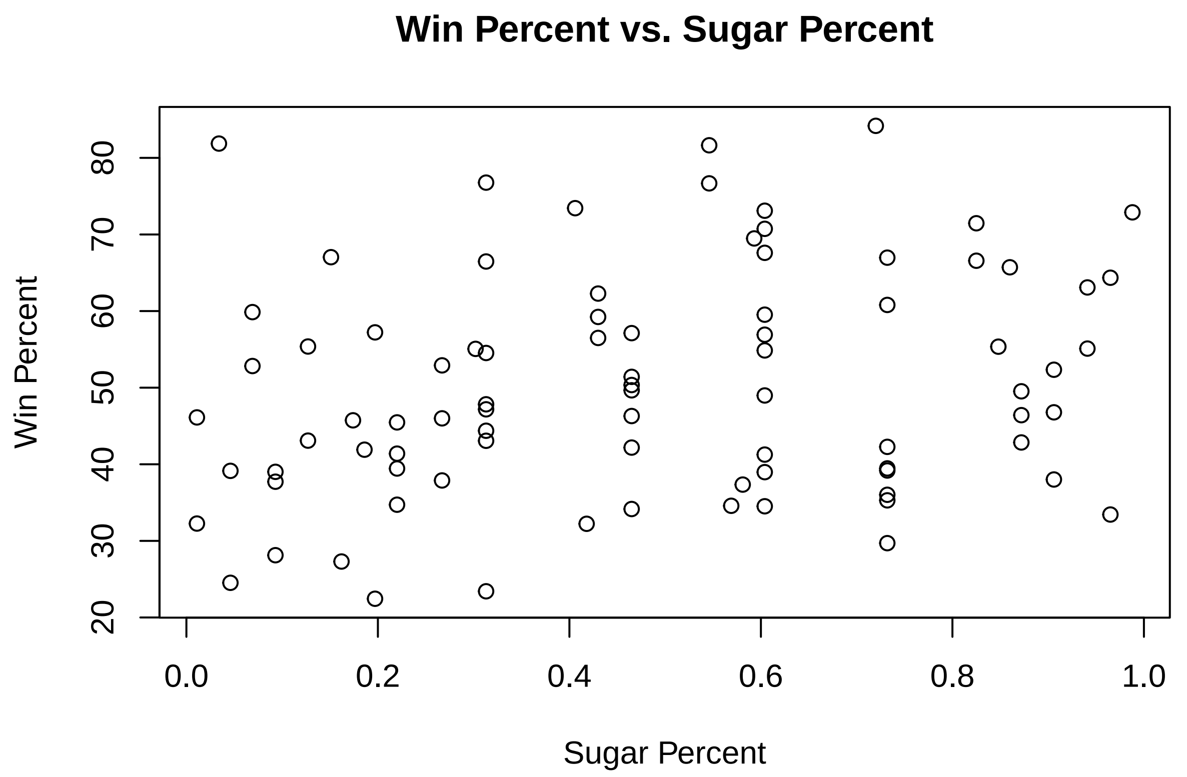
```
cat("Correlation between winpercent and pricepercent:", correlation_matrix["winpercent", "pricepercent"
```

```
## Correlation between winpercent and pricepercent: 0.3453254
```

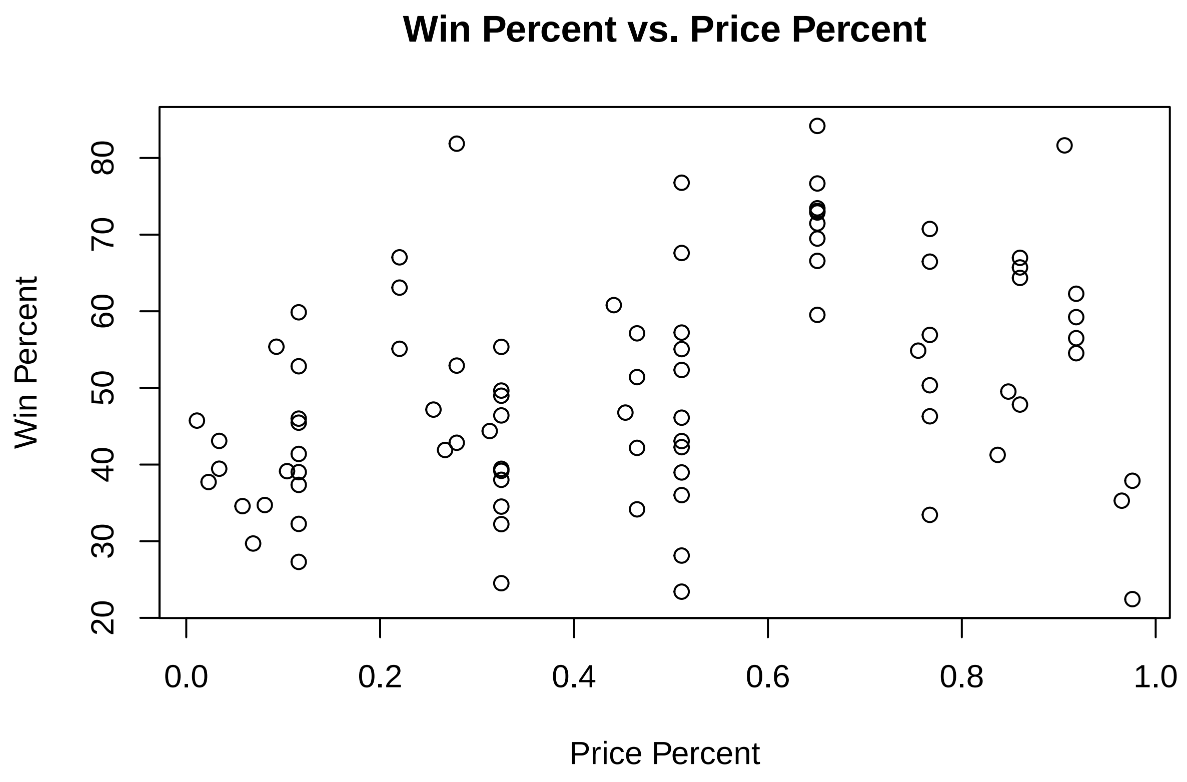
```
# Generate a scatterplot for winpercent vs. sugarpercent
plot(candy_rankings$pricepercent, candy_rankings$sugarpercent,
     xlab = "Price Percent", ylab = "Sugar Percent", main = "Sugar Percent vs. Price Percent")
```



```
# Generate a scatterplot for winpercent vs. sugarpercent
plot(candy_rankings$sugarpercent, candy_rankings$winpercent,
     xlab = "Sugar Percent", ylab = "Win Percent", main = "Win Percent vs. Sugar Percent")
```



```r
# Generate a scatterplot for winpercent vs. pricepercent
plot(candy_rankings$pricepercent, candy_rankings$winpercent,
     xlab = "Price Percent", ylab = "Win Percent", main = "Win Percent vs. Price Percent")
```



The correlation value between sugarpercent and pricepercent is approximately 0.330. This suggests a weak positive linear relationship between sugarpercent and pricepercent. As the sugarpercent of candies increases, winpercent increases a bit.

The correlation value between winpercent and sugarpercent is approximately 0.223. This suggests an even weaker positive linear relationship between sugarpercent and pricepercent. As the sugarpercent of candies increases, winpercent increases a lttle.

The correlation value between winpercent and pricepercent is approximately 0.223. This suggests a stronger positive linear relationship between winpercent and pricepercent. As the pricepercent of candies increases, the winpercent increases a bit more.

11g: which taste the best is

```
# The most tasty flavour is:
print(candy_counts[1, ])
```

```
##   chocolate caramel   bar fruity peanutyalmondy crispedricewafer count
## 7     FALSE   FALSE FALSE   TRUE          FALSE            FALSE    36
```

```
#"The most untasty flavour is:
print(candy_counts[18, ])
```

```
##    chocolate caramel  bar fruity peanutyalmondy crispedricewafer count
## 18      TRUE    TRUE TRUE  FALSE           TRUE             TRUE     1
```

We can see from theresults that apparently the most tasty flavor is “fruity” and the least tasty is every flavor, except fruity, mixed.

Task 12:

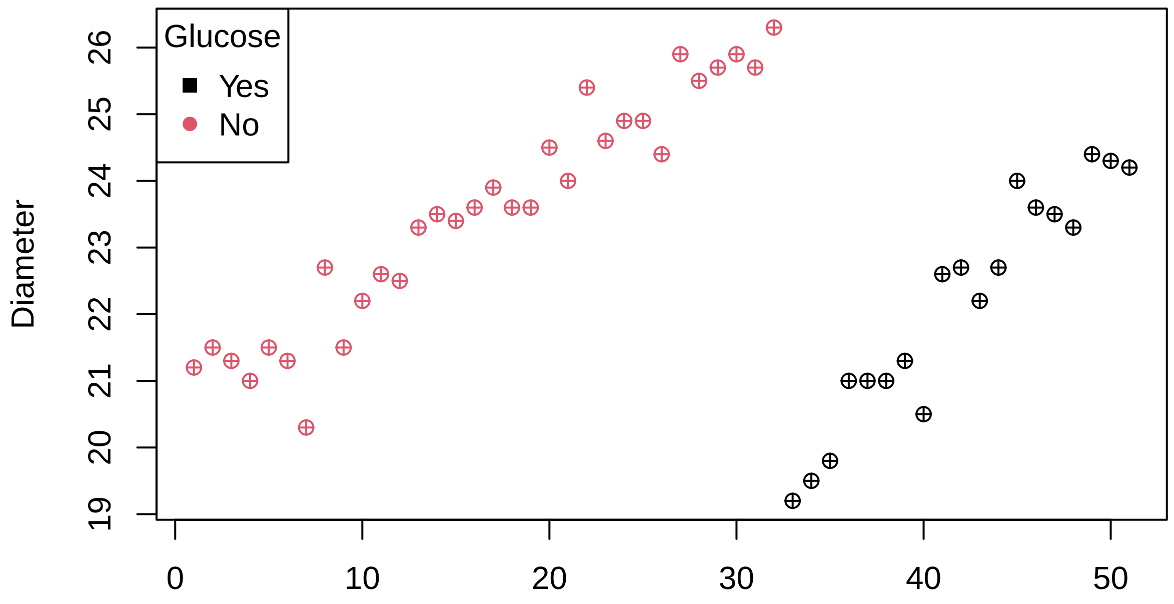
12a: installation:

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

There is a lot of free space here. But I wanted to have the code with the corrisponding box plot all on one page

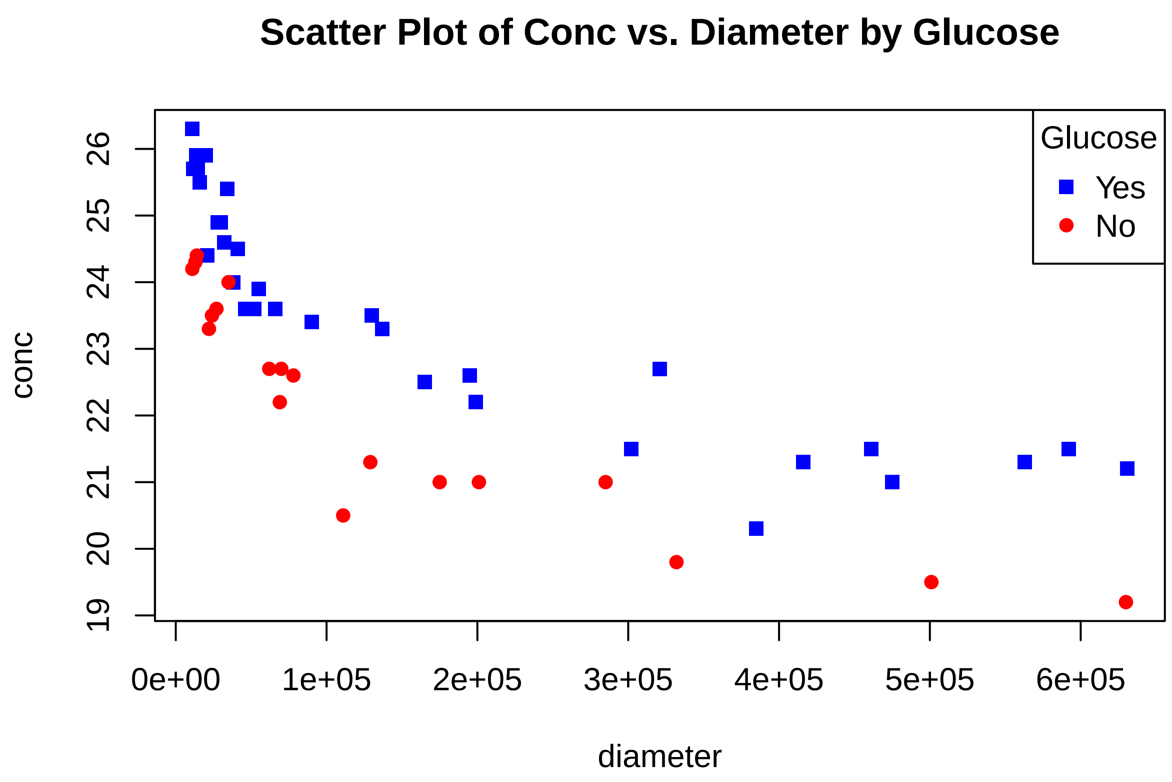
12b: add GLUCOSE to dataframe:

```
# Define a function to label glucose as "Yes" or "No" based on a threshold
GLUCOSE <- factor(ifelse(hellung$glucose == 1, "Yes", "No"))

# Add the GLUCOSE variable to the data frame
hellung$GLUCOSE <- GLUCOSE
#hellung

#create a scatterplot
plot (hellung$diameter,
      pch = 10,
      col = GLUCOSE,
      ylab ="Diameter",
      xlab ="")

legend("topleft",
       legend = c("Yes", "No"),
       col = factor(levels(GLUCOSE)),
       pch = c(15, 16),
       title = "Glucose")
```



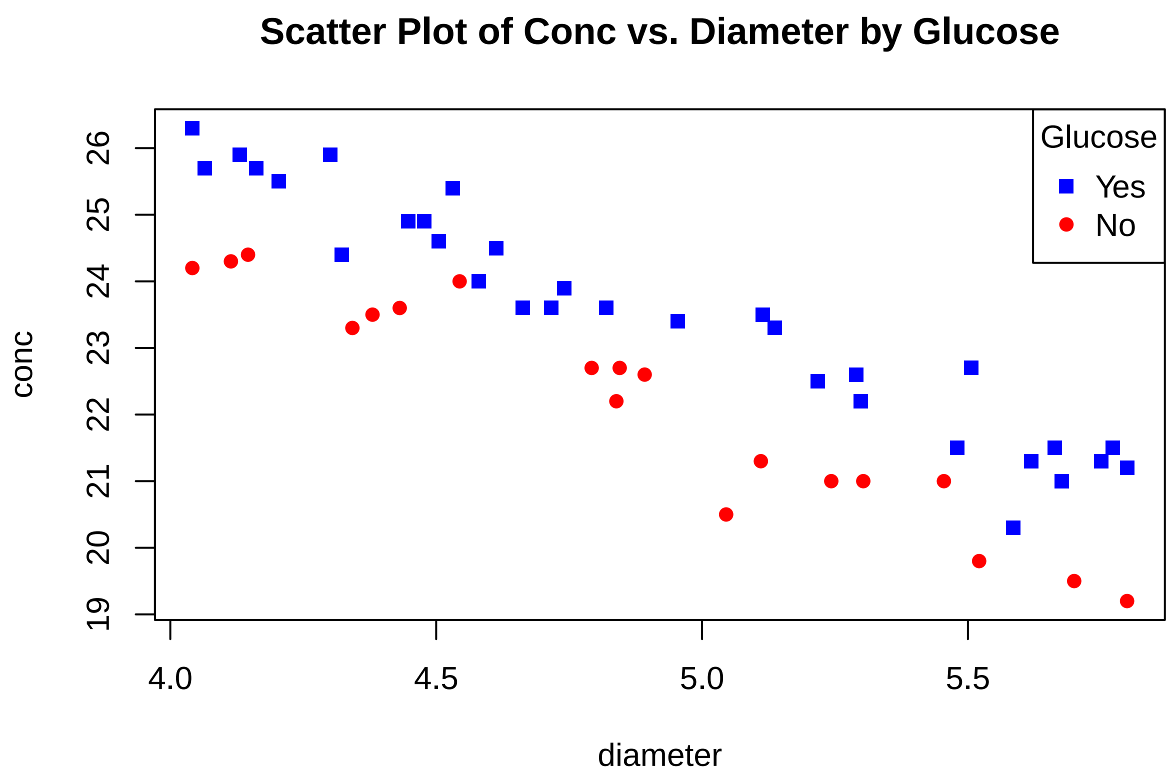
12c: plot conc against diameter

```r
# Create a vector of colors for "glucose" (Yes: blue, No: red)
glucose_colors <- ifelse(hellung$GLUCOSE == "Yes", "blue", "red")

# Create a vector of plotting symbols for "glucose" (Yes: square, No: circle)
glucose_symbols <- ifelse(hellung$GLUCOSE == "Yes", 15, 16)

# Create a scatter plot with different colors and symbols
plot(hellung$conc,
     hellung$diameter,
     col = glucose_colors,
     pch = glucose_symbols,
     xlab = "diameter",
     ylab = "conc",
     main = "Scatter Plot of Conc vs. Diameter by Glucose")

# Add a custom legend
legend("topright",
       legend = c("Yes", "No"),
       col = c("blue", "red"),
       pch = c(15, 16),
       title = "Glucose")
```



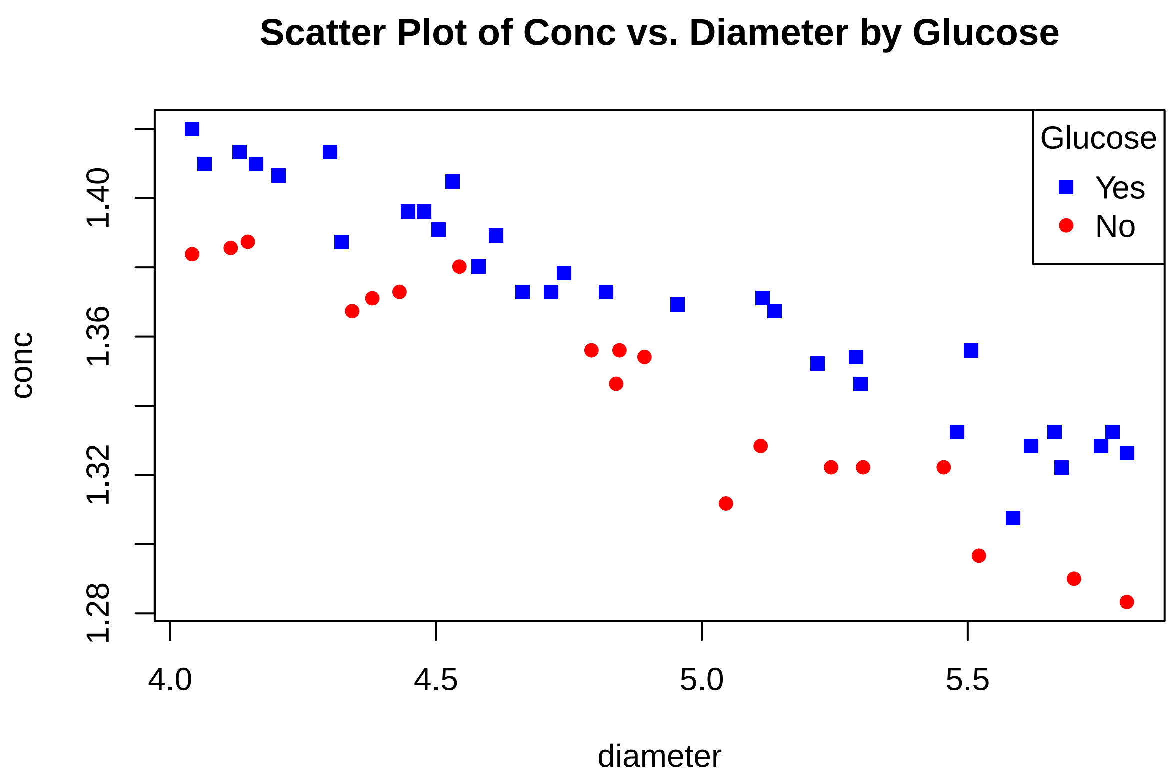
12d: logarithmic scale for X-Axis

```r
plot(log10(hellung$conc),
     hellung$diameter,
     col = glucose_colors,
     pch = glucose_symbols,
     xlab = "diameter",
     ylab = "conc",
     main = "Scatter Plot of Conc vs. Diameter by Glucose")

# Add a custom legend
legend("topright",
       legend = c("Yes", "No"),
       col = c("blue", "red"),
       pch = c(15, 16),
       title = "Glucose")
```



12e: logarithmic scale for both Axis

```r
#create plot
plot(log10(hellung$conc),
     log10(hellung$diameter),
     col = glucose_colors,
     pch = glucose_symbols,
     xlab = "diameter",
     ylab = "conc",
     main = "Scatter Plot of Conc vs. Diameter by Glucose")

# Add a custom legend
legend("topright",
       legend = c("Yes", "No"),
       col = c("blue", "red"),
       pch = c(15, 16), title = "Glucose")
```



12f: create PDF:

```r
# Specify the PDF file name and dimensions
pdf("ChristopherScherling.pdf", width = 7, height = 7)

# Create the scatter plot with logarithmic scales and custom labels
plot(hellung$conc, hellung$diameter, col = glucose_colors, pch = glucose_symbols,
     xlab = "Log(Diameter)",
     ylab = "Log(Conc)",
     main = "Scatter Plot of Conc vs. Diameter by Glucose")

# Add a custom legend
legend("topright",
       legend = c("Yes", "No"),
       col = c("blue", "red"),
       pch = c(15, 16),
       title = "Glucose")

# Close the PDF device to save the plot
dev.off()
```

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

Task 13: find_optimal_cutoff function

```r
find_optimal_cutoff <- function(x, y) {
  #Unique cutoff values in x #Extract all unique elements from the vector and
  # then creates a numeric vector with a length equal to the number of unique
  #cutoff values.
  unique_cutoffs <- unique(x)
  misclassification_rates <- numeric(length(unique_cutoffs))


  #The loop effectively goes through each unique cutoff value, calculates the
  #misclassification rate associated with that cutoff, and stores the rate in
  #the misclassification_rates vector for further analysis.
  for (i in seq_along(unique_cutoffs)) {
    N <- unique_cutoffs[i]  # Current cutoff value
    predicted_labels <- ifelse(x > N, "spam", "ham")
    misclassification_rates[i] <- sum(predicted_labels != y) / length(y)
  }

  # Find the optimal cutoff with the smallest misclassification rate
  optimal_cutoff_index <- which.min(misclassification_rates)
  optimal_cutoff <- unique_cutoffs[optimal_cutoff_index]

  # Return the optimal cutoff and its misclassification rate
  return(list(optimal_cutoff = optimal_cutoff, misclassification_rate =
              misclassification_rates[optimal_cutoff_index]))
}

y <- c("ham", "ham", "ham", "ham", "ham", "spam",
       "ham", "spam", "ham", "spam", "ham", "ham",
       "spam", "spam", "spam", "ham", "spam", "spam")
x <- c(0, 0, 0, 1, 1, 1,
       2, 2, 2, 2, 2, 4,
       5, 5, 6, 6, 8, 8)


# Find the optimal cutoff and misclassification rate
result <- find_optimal_cutoff(x, y)
cat("Optimal Cutoff:", result$optimal_cutoff, "\n")
```

```
## Optimal Cutoff: 4
```

```r
cat("Misclassification Rate:", result$misclassification_rate, "\n")
```

```
## Misclassification Rate: 0.2222222
```

In this function, we iterate through the unique cutoff values in the x vector, calculate the misclassification rate for each cutoff and then find the optimal cutoff with the smallest misclassification rate. The result includes the optimal cutoff value and its corresponding misclassification rate.

Feedback:

(96,00 / 100,00)

Task 8b: How did you get to this result? -3

Task 8c: Which basic law of arithmetic is not satisfied by floating point arithmetic in the example below? -1

10i: Instead of commenting a block you can just set evaluation false when creating the code chunk. This works with

```
library(package)
```

Task 11d: Seems fine to me

Great work!