SCR Exam 1: the basics

Exam Instructions

- You are allowed to consult the internet and all files on your computer. However, keep in mind that any form of direct communication with others e.g. chatting, apping, facebook is not allowed and is considered **FRAUD**.
- Your exam answers need to be a neat `.R` script, including comments.
- Upload your `Lastname_ULCN.Rmd` file only (e.g. Tzortzakis_0730406.Rmd or Kampert_002143.R). Turn it in **BEFORE** 13:05 hours to **Blackboard**:
  - Go to Statistical Computing with R –> Course Documents –> Exam 26 October 2017
  - Every minute too late will cost you 10 out of 100 points: when you submit your exam at 13:05 spot on, it means you already lose 50 points. Exams submitted after 13:05 will not be graded.

- The exam consists of 6 tasks, divided into subtasks. Each subtask is worth an equal amount of points. There is also a bonus question that will allow you to earn 5 points.

- Your style of coding can affect your final exam grade. Your code does not need to be beautiful, the correct answer is the most important, but e.g. very complicated code where a simple known base `R` function will also work may cost you points.

- Adhere to consistent and neat programming style, for some examples:
  - https://google.github.io/styleguide/Rguide.xml

  - http://style.tidyverse.org

- Assume that you work in a similar working directory as the one in which you can find this `.Rmd` file. E.g. scriptlines such as `load("0_data/scr_exam_1_the-basics.Rdata")` should evaluate correctly.

Success!

The R-team.

The start

In the previous assignment we've seen a data set about blood samples, taken from 628 subjects, and analysed in 12 different centers. In this exam we will perform analyses on this data, and visualize the results. The correct data for this exam can be read in using the file `exam_1.Rdata`.

```
load("0_data/scr_exam_1_the-basics.Rdata")
```

1 Introduction

a

Take a look at `bloodsamples_data`, you should be familiar with the dataset as it is the same one we used in the assignment. Remove the rows in the dataset for which `sex` is missing.

Answer:

```
bloodsamples_data <- bloodsamples_data[-which(is.na(bloodsamples_data$sex)), ]
```

b

Provide a `summary` for the variables `sex`, `age_years`, `hg`, `ir` and `creat`.

Answer:

```
summary(bloodsamples_data[, c("sex", "age_years", "hg", "ir", "creat")])
```

```
##      sex              age_years          hg              ir
##  Length:634         Min.   :18.00   Min.   :11.48   Min.   : 20.38
##  Class :character   1st Qu.:21.00   1st Qu.:13.65   1st Qu.: 93.96
##  Mode  :character   Median :25.00   Median :14.66   Median :115.55
##                     Mean   :25.21   Mean   :14.83   Mean   :115.30
##                     3rd Qu.:30.00   3rd Qu.:15.94   3rd Qu.:135.76
##                     Max.   :33.00   Max.   :19.52   Max.   :201.37
##                                                     NA's   :15
##      creat
##  Min.   :-0.9464
##  1st Qu.: 5.3586
##  Median : 7.2526
##  Mean   : 7.1307
##  3rd Qu.: 8.8138
##  Max.   :15.5484
##
```

c

Sex is a categorical variable. Change it's mode so that when you use the `summary` function, `R` knows to regard `sex` as a categorical variable.

Answer:

```
bloodsamples_data$sex <- as.factor(bloodsamples_data$sex)
```

2 Set-up

In this analysis we will look at the relation between two variables found in the data: `hg` and `ir`. `hg` is a variable that represents the amount of hemoglobin the blood of the subjects at the time of measurement and `ir` is a variable that represents of the amount of iron in the blood of the subjects at the time of measurement. It is known that males tend to have more iron, and more hemoglobin in their blood than females.

More specifically, in this exam we will look at the correlation between these two variables.

Some of the data for the variable `ir` is missing however. In this exam we will look at two simple ways to impute some of the missing data in our data set. Note that there are much better ways available, but these are often more complicated!

a

Calculate the correlation between `hg` and `ir`. Tell `R` to only use complete observations.

Answer:

```
cor(bloodsamples_data$hg, bloodsamples_data$ir, use="complete.obs") # pairwise.complete.obs is also ok.
```

```
## [1] 0.1258918
```

b

Deleting observations that have missing values is only proper if the missing observations are missing *completely at random*. That is: if the probability of a value missing does not depend on any variables observed, or unobserved.

Look at the distribution of the observations for which `ir` has missing values in relation to the variables `sex` and `locations`. Does anything stand out?

Answer:

```
table(is.na(bloodsamples_data$ir), bloodsamples_data$location)
```

```
##
##         's-Hertogenbosch Arnhem Assen Den Haag Groningen Haarlem
##   FALSE               61      5    52       78        48      45
##   TRUE                 1      0     1        1         2       3
##
##         Leeuwarden Lelystad Maastricht Middelburg Utrecht Zwolle
##   FALSE         11       92         88         10      93     36
##   TRUE           0        1          2          0       0      4
```

```
table(is.na(bloodsamples_data$ir), bloodsamples_data$sex)
```

```
##
##         female male
##   FALSE    313  306
##   TRUE       0   15
```

The missings are all males!

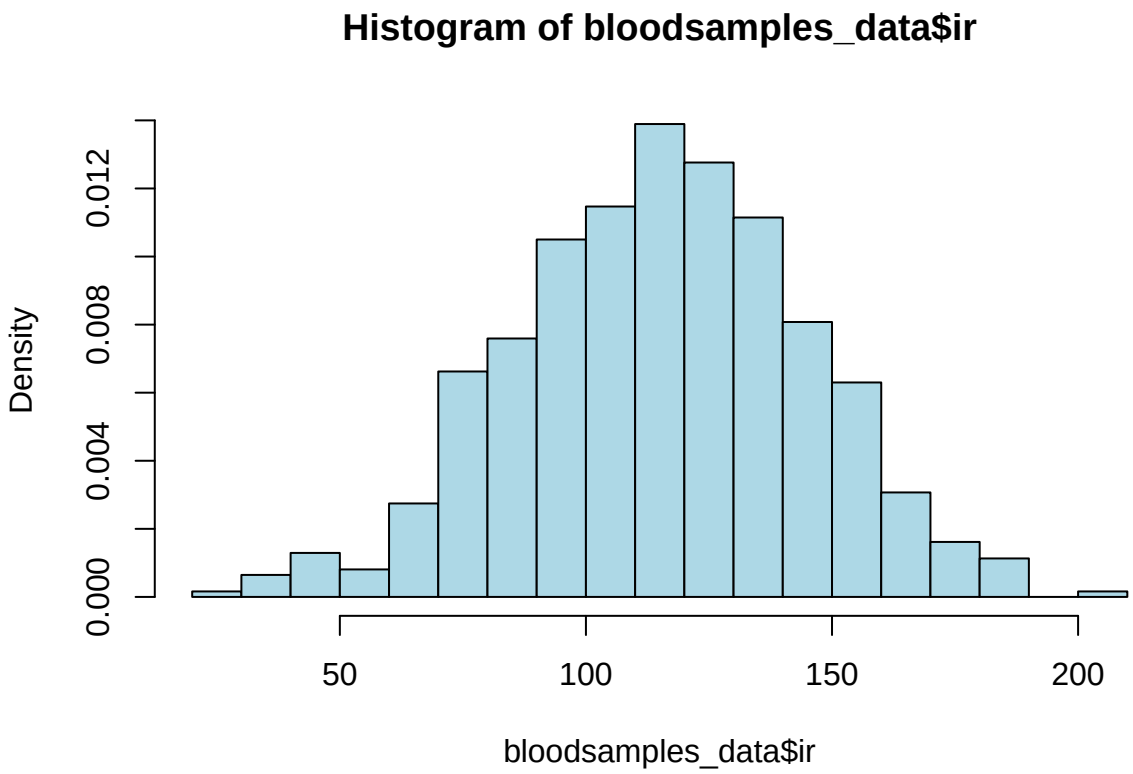
3 Random imputation

a

Make a histogram of the variable `ir`. Let the height of the bars not represent the frequency with which an observation within that bin has been observed, but rather the proportion of the data that falls within that bin.

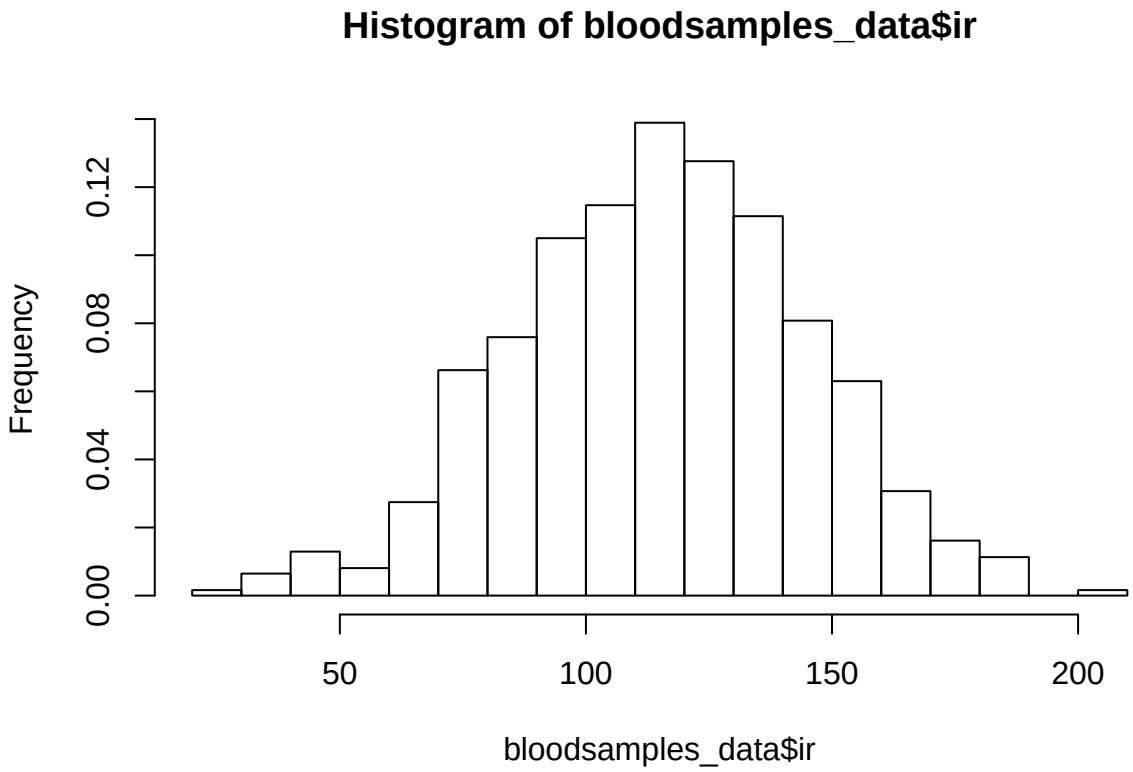
Answer:

```
hist(bloodsamples_data$ir, breaks="FD", col='lightblue', freq=F) # breaks="FD" is optional
```




```
# or

hist_data <- hist(bloodsamples_data$ir, plot=FALSE, breaks="FD")
hist_data$counts <- hist_data$counts/sum(hist_data$counts)
plot(hist_data)
```



Note these

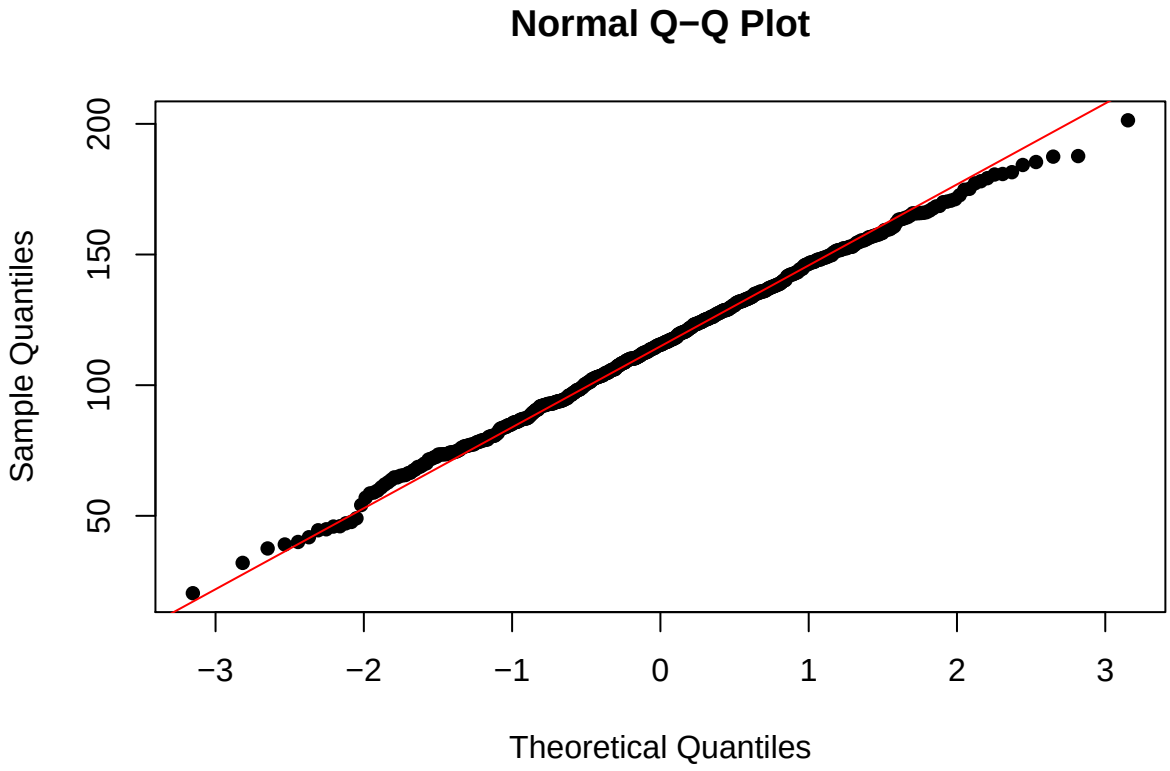
are completely different plots, but both solutions are awarded full points.

b

The histogram looks 'normal': the distribution of the data looks normal. However, a histogram is not the ideal method to investigate normality. Use another plot we discussed in the course, to investigate whether the data is normally distributed.

Answer:

```
qqnorm(bloodsamples_data$ir, pch=16)
qqline(bloodsamples_data$ir, col='red')
```



c

Let's assume for now the data is normally distributed. We might randomly impute some values for the missing data, using the information we have about the distribution of the values of `ir`. Calculate the mean and standard devation of the `ir` variable.

Answer:

```
ir.mean <- mean(bloodsamples_data$ir, na.rm = T)
ir.sd <- sd(bloodsamples_data$ir, na.rm = T)
```

d

Use `rnorm` to randomly sample as many data points as there are missing values in the variable `ir`. Make sure you set a seed.

Answer:

```
n.missing <- sum(is.na(bloodsamples_data$ir))
```

```r
set.seed(20170807)
ir.sample <- rnorm(n.missing, mean=ir.mean, sd=ir.sd)
```

e

Create a new variable `ir.imputed` that is the original `ir` variable, but with the `NA` entries replaced by values you just sampled.

Answer:

```r
ir.imputed <- bloodsamples_data$ir
ir.imputed[is.na(ir.imputed)] <- ir.sample
```

f

Calculate the correlation between `hg` and `ir.imputed`. Did the correlation change?

Answer:

```r
cor(bloodsamples_data$hg, ir.imputed)
```

```
## [1] 0.1096192
```

4 Informative imputation

We will now look at imputation using some information from our dataset. Instead of randomly sampling values for `ir`, using the distribution characteristics, such as the mean and standard devation. We can also use the other variables in the dataset, to given a reasonable estimate of the missing values should have been.

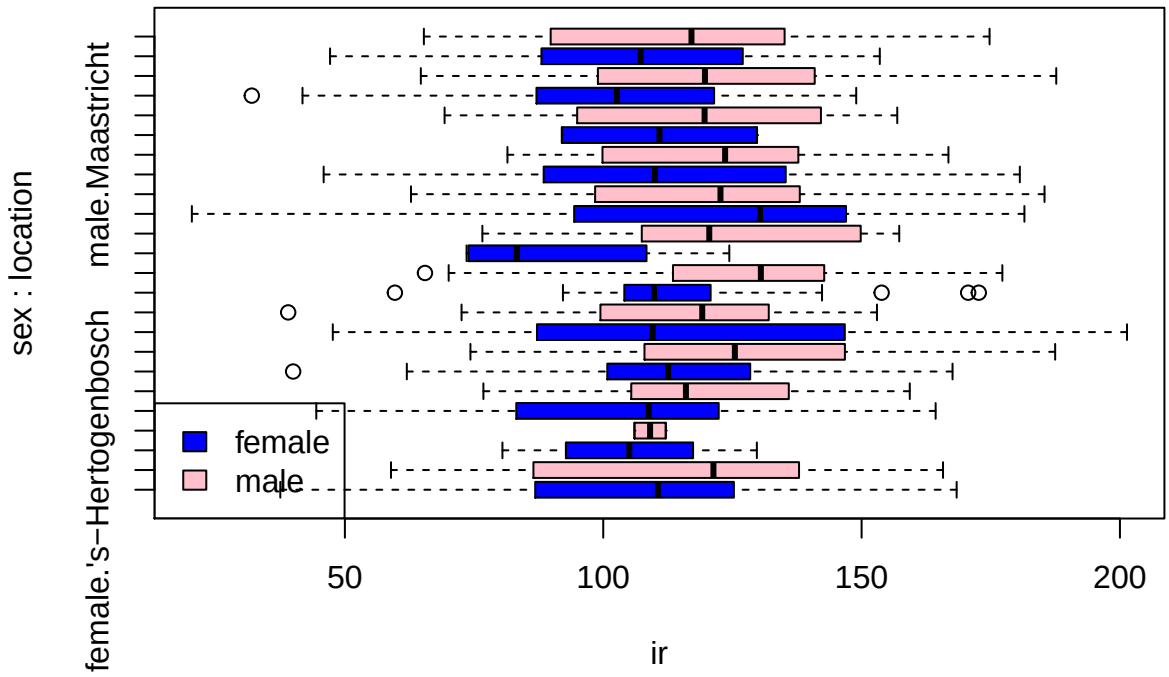
a

Use `boxplot` and a `formula` to visualize the various distributions of `ir` conditional on the observed value of `sex` and `location`.

- Assign a pink colour the boxes for the males, and blue to the boxes for the females.
- Rotate the plot so that the boxes are rows, instead of columns.
- Change the length of the 'whiskers' so that they are as wide as the boxes.
- Add a legend to indicate that the pink boxes belong to the male groups, and the blue ones the female groups.

Answer:

```r
boxplot(
  formula = ir ~ sex + location,
  data = bloodsamples_data,
  col=c("blue", "pink"),
  horizontal = TRUE,
  staplewex = 1
)
legend("bottomleft", fill=c("blue", "pink"), legend=c("female", "male"))
```



b

Using `tapply` to calculate the average value of `ir` seperately for all 24 groups that result from all combinations of the variables `sex` and `location`. Make sure that you tell `R` to remove the values that are `NA`.

Answer:

```
my_means <- tapply(
  X = bloodsamples_data$ir,
  INDEX = list(bloodsamples_data$sex, bloodsamples_data$location),
  FUN = mean, na.rm=TRUE
)
```

c

In case you did not manage to create the collection of means in **a**, you may continue with the values in `means_exercise_a`.

Create a function that takes a row from `bloodsamples_data` and checks if the observation has a missing value on `ir`. If so: use the means you've calculated to 'predict' the outcomes for the observations for which `ir` is missing. Use the average, corresponding to the group the observation is in, based on their value for `sex` and `location`. Have the function return the row of data, and in case the entry for `ir` was missing, with the imputed value for `ir`.

Answer:

```
locations <- sort(unique(bloodsamples_data$location))

ImputeIr <- function(x, my_means, locations){
  if (is.na(x$ir)){
    x$ir <- my_means[(x$sex=="male")+1, x$location==locations]
  }
  return(x)
}
```

d

Copy the existing `bloodsamples_data` object to a new object called `bloodsamples_data_imputed`. Use a `for` loop to apply the function you made in **b** to all the rows of `bloodsamples_data_imputed`. At the end of the `for` loop, you should thus have a dataset that is equal to `bloodsamples_data`, except that the `NA` values in `ir` have been imputed.

Answer:

```
bloodsamples_data_imputed <- bloodsamples_data
for (i in 1:nrow(bloodsamples_data_imputed)){
  bloodsamples_data_imputed[i, ] <- ImputeIr(bloodsamples_data_imputed[i, ], my_means, locations)
}
```

e

Calculate the correlation between `hg` and `ir` in the `bloodsamples_data_imputed` datset. Is the correlation different from the initial one?

Answer:

```
cor.imputed <- cor(bloodsamples_data_imputed$hg, bloodsamples_data_imputed$ir); cor.imputed
```

```
## [1] 0.1274753
```

5 Impute again, analyze again

In this last task we will repeatedly impute (random) values for the missing values in the variable `ir`. We will visualize this result and compare it with the result we got using the means approach.

a

First, write a function called `ImputedCor` that performs the procedure that you've written in steps **a** through **e** of task **Random imputation**.

- The function should take as argument the variables `hg` and `ir`, and a mean and a standard deviation.
- You create new variable called `ir.imputed`, that is equal to `ir`, but instead of `NA`'s has randomly sampled normally distributed values with mean and standard deviation equal to the inputs of the function.
- The output should be the correlation between `hg` and `ir.imputed`.

Answer:

```
ImputedCor <- function(hg, ir, mean, sd){
  ir.imputed <- ir
  na.index <- is.na(ir.imputed)
  n.missing <- sum(na.index)
  ir.imputed[is.na(ir.imputed)] <- rnorm(n.missing, mean=mean, sd=sd)

  return(cor(hg, ir.imputed))
}
```

b

Use `replicate` to run your `ImputedCor` function a 1000 times. What is the mean of these 1000 replications? How does it compare to the correlation calculated using only the complete cases?

Answer:

```
sampled.cor <- replicate(1000, ImputedCor(bloodsamples_data$hg, bloodsamples_data$ir, ir.mean, ir.sd))
```

c

Compare the three different estimates of correlation that you have created. Which of the correlations that you've calculated do you think is closer to the truth?

Answer:

```
mean(sampled.cor)
```

```
## [1] 0.1224023
```

```
cor(bloodsamples_data$hg, bloodsamples_data$ir, use="complete.obs") #pairwise.complete.obs is also ok.
```

```
## [1] 0.1258918
```

```
cor(bloodsamples_data_imputed$hg, bloodsamples_data_imputed$ir)
```

```
## [1] 0.1274753
```

```
# truth (unknown to students):
.1278132
```

```
## [1] 0.1278132
```

Deleting cases is okay if missing completely at random. Randomly sampling will underestimate, because it ignores any structural parts in the data. Using informative imputation might overestimate. So: we can't really answer this without know the missing generating process.

6 Self Service Analytics

Now that we've done some work for the researcher, it is perhaps nice to provide him with some code that allows him to do some parts of the project again in `R`, interactively. In this last part we'll look at some interactivity in `R`.

a

Write a loop that `repeat`edly asks a user for text input using `readline`. Make sure that before you ask for input, you print to the console some information so that the user knows he/she should give some input. Have the loop stop if the input "EXIT" is given.

Answer:

```
# repeat{
#
#   cat("Please give some input: ")
#   input <- readline()
#   if (input == "EXIT"){
#     break
#   }
#
# }
```

Bonus

Doing this question is just for bonus points (maximally 5 out of 100 points). It will be more difficult than the rest of the exam and very inefficient in terms of time spent per gradepoint, so only do this if you have any time left.

Warning: this may take you a lot of time!

This will combine what you've already done above, but expand it to something that's actually useful. We'll create a function that allows us to interactively inspect variables part of a dataset.

- Write a function that takes as argument a data.frame and make sure the function gives an error if anything other than a dataframe is provided to the function.
- The function should repeatedly 'prompt' the user for strings *until* the word "EXIT" is written (*hint: use* `readline` *for this*).
    1. If the string is a variable part of the data provided to the function, it should print to the console the first `A` entries, the last `B` entries, and a summary of the variables. Set `A` and `B` to 10 by default.
    2. If the string provided by the user is one of the following two keywords: "change_A", "change_B", ask the user to what value the number of `A` or `B` should be changed respectively. Make sure you check if it's a valid value.
    3. If the string provided by the user is not a variable name of the dataset or one of the 'keywords', print a message that the user should provide one of the variable names, and then print the variable names of the dataset.
- Make sure the output looks nice.
- The function should return a list containing all of the output printed to the console so far.
- Make sure the list you return has proper names for its elements indicating what variable the output is from, and what each element of the list is.
- Make sure R does not print the return value to the console automatically, if the return value is not stored to some object.

Answer:

```
InterSumm <- function(x){

  if(!is.data.frame(x)){
    stop("argument x not a data.frame.")
  } else {
    x <- na.omit(x)
  }

  var_names <- names(x)
  return_list <- list()
  A <- 10
  B <- 10

  while(TRUE){
    cat("Please provide a string:")
    string <- readline()

    if (string == "EXIT"){
      break
    } else if (string == "change_A" || string == "change_B"){
      cat("Please provided a value: ")
      while(TRUE){
        sub_string <- readline()
        if (suppressWarnings(is.na(as.numeric(sub_string)))){
          cat("Please provide a proper numeric value: ")
        } else {
          if (string == "change_A"){
            A <- as.numeric(sub_string)
          } else {
            B <- as.numeric(sub_string)
```

```
          }
          break;
        }
      }
    }

    if (string%in%var_names){
      # create output object
      new_output <- list(head(x[, string], A), tail(x[, string], B), summary(x[, string]))

      cat("First 10 values: ", new_output[[1]],
          "\nLast 10 values: ", new_output[[2]],
          "\nSummary: ")
      print(new_output[[3]])

      # expand return list
      return_list <- c(return_list, list(new_output))
      names(return_list)[length(return_list)] <- string
      names(return_list[[length(return_list)]]) <- c("head", "tail", "summary")
    } else {
      cat(
        "Please provide a variable name, part of argument x. Options are:\n",
        paste(var_names, collapse =", ")
      )
    }

  }

  return(invisible(return_list))
}

# InterSumm(bloodsamples_data)

# interactive_summary <- InterSumm(bloodsamples_data)
```