Exercise #3

t-test and R programming

Brian Schweigler; 16-102-071

23/03/2022

Preliminaries

Loading the mean dataset:

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

mean20_df = read.csv("Mean20.txt", header = TRUE, comment.char = "#")
```

Handle the outliers:

- Remove NAs
- Remove illogical values (<= 0)

```
temp_mean20_df <- mean20_df
temp_mean20_df[temp_mean20_df <= 0] <- NA
cleaned_mean20_df <- na.omit(temp_mean20_df)
summary(cleaned_mean20_df)
```

```
##       time
##  Min.   :6.850
##  1st Qu.:6.968
##  Median :7.010
##  Mean   :7.008
##  3rd Qu.:7.072
##  Max.   :7.120
```

```
sd(cleaned_mean20_df$time)
```

```
## [1] 0.07515598
```

2. Suppose that mean delay between two calls is 7.05 minutes. Can you test this hypothesis using the available data? What is your conclusion? Do you see a difference when considering the original values and the preprocessed values?

```
assumed_mean <- 7.05
estimated_mean <- mean(cleaned_mean20_df$time)
t.test(x = cleaned_mean20_df, mu = assumed_mean, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  cleaned_mean20_df
## t = -2.4992, df = 19, p-value = 0.02178
## alternative hypothesis: true mean is not equal to 7.05
## 95 percent confidence interval:
##  6.972826 7.043174
## sample estimates:
## mean of x
##     7.008
```

Assuming we want the 95% confidance interval, we reject the hypothesis of the mean delay being 7.05 minutes.

Comparing to the unprocessed values:

```
t.test(x = mean20_df, mu = assumed_mean, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  mean20_df
## t = -1.0626, df = 20, p-value = 0.3006
## alternative hypothesis: true mean is not equal to 7.05
## 95 percent confidence interval:
##  4.947647 7.733306
## sample estimates:
## mean of x
##  6.340476
```

While for the unprocessed data the true mean is not 7.05, it is clear that there is a >95% probability that our hypothesis holds. This is because the unprocessed data includes a negative value (I'd assume NA's are ignored), which increases the total range of values immensly.

3. For Mary, the delay cannot be smaller than 7.05 minutes. Thus the only credible alternative hypothesis must take account of this (well-known) fact. How can you test Mary's hypothesis?

We can use a one-sided t-test for this:

```
t.test(x = cleaned_mean20_df, alternative = "g", mu = assumed_mean, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  cleaned_mean20_df
## t = -2.4992, df = 19, p-value = 0.9891
## alternative hypothesis: true mean is greater than 7.05
## 95 percent confidence interval:
##  6.978941      Inf
## sample estimates:
## mean of x
##     7.008
```

So for the processed data, this hypothesis holds in at least 95% of the cases.

```
t.test(x = mean20_df, alternative = "g", mu = assumed_mean, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  mean20_df
## t = -1.0626, df = 20, p-value = 0.8497
## alternative hypothesis: true mean is greater than 7.05
## 95 percent confidence interval:
##  5.188856      Inf
## sample estimates:
## mean of x
##  6.340476
```

For the unprocessed variant this also holds true as the range of the 95% confidence interval is larger.

4. Define a function secondMax(x), where x is a vector, returning the second largest value contained in x. If x is not a vector, return an error message. Test your implementation in different cases using the Mean20 dataset.

First we define the function and check if x is a vector

```
secondMax <- function(x) {
    if (!is.vector(x))  {
        return("Input x is not a numeric vector!")
    }
    # need at least 2 values
    if (length(x) <= 1)     {
        return("x must contain at least 2 values!")
    }
    if (!is.numeric(x)) {
        return("Vector components must be numeric!")
    }

    # Remove largest value, such that the second largest value is now the largest
    x_without_max = x[-which(x == max(x))]

    return(max(x_without_max))
}
```

Testing the implementation:

```
secondMax(cleaned_mean20_df$time)
```

```
## [1] 7.1
```

```
secondMax(runif(n = 100, min = 0, max = 372)) # random values from 0 to 372
```

```
## [1] 366.0107
```

```
secondMax(c("1","2","4")) # should fail
```

```
## [1] "Vector components must be numeric!"
```

```
secondMax(c("yes","this","is","patrick")) # should fail
```

```
## [1] "Vector components must be numeric!"
```

```
secondMax(cleaned_mean20_df) # should fail
```

```
## [1] "Input x is not a numeric vector!"
```

```
secondMax(NA) # should fail
```

```
## [1] "x must contain at least 2 values!"
```

```
secondMax(c(NA, NA, NA)) # should fail
```

```
## [1] "Vector components must be numeric!"
```

5. Define a function mySummary(x), where x is a vector composed by the mean, the median, the standard deviation, the minimum and the maximum values (in this order). Test your implementation in different cases using the Mean20 dataset

```
mySummary <- function(x, remove_na = TRUE) {
    if (!is.vector(x))  {
        return("Input x is not a numeric vector!")
    }
    # need at least 2 values
    if (length(x) <= 1)     {
        return("x must contain at least 2 values!")
    }
    if (!is.numeric(x)) {
        return("Vector components must be numeric!")
    }
    if (remove_na) {
        na_values = which(is.na(x))
        if (length(na_values) > 0) {
            x <- x[-isna]
        }
    }

    return(c(mean = mean(x), median = median(x), stdev = sd(x), min = min(x), max = max(x)))
}
```

Testing with similar cases as before:

```
mySummary(cleaned_mean20_df$time)
```

```
##       mean     median      stdev        min        max
## 7.00800000 7.01000000 0.07515598 6.85000000 7.12000000
```

```
mySummary(runif(n = 100, min = 0, max = 372)) # random values from 0 to 372
```

```
##      mean    median     stdev       min       max
## 180.14271 158.91774 100.28471  16.37301 366.59050
```

```
mySummary(c("1","2","4")) # should fail
```

```
## [1] "Vector components must be numeric!"
```

```
mySummary(c("yes","this","is","patrick")) # should fail
```

```
## [1] "Vector components must be numeric!"
```

```
mySummary(cleaned_mean20_df) # should fail
```

```
## [1] "Input x is not a numeric vector!"
```

```
mySummary(NA) # should fail
```

```
## [1] "x must contain at least 2 values!"
```

```
mySummary(c(NA, NA, NA)) # should fail
```

```
## [1] "Vector components must be numeric!"
```