

R class notes 2022 11 01

Adam Tallman

2022-10-31

Introduction

Today's lecture is going to focus on writing data onto a datasheet & reading data into R. We will also go into more detail on R language, for Objects, modes and attributes. We will talk about ordered and unordered factors,, matrices, lists and dataframes. Then we will do more plotting with R with real data.

Reading data

Make a dataset with excel or libraoffice calc. Libraoffice is better for encoding issues, so I would recommend downloading it - its very similar to excel, just a little more flexible.

fake.proficiencydata2.csv - LibreOffice Calc


Id	Age	Months.learn	Proficiency.score	First.language
1	22	12	1	Russian
2	23	15	2	German
3	19	16	4	French
4	20	12	5	French
5	21	15	6	Arabic
6	22	16	8	Russian
7	22	22	9	Russian
8	23	5	2	Russian
9	24	12	1	Russian
10	21	14	5	French
11	19	6	1	Arabic

Make a dataset in excel or libraoffice calc

You can read the data in with the following code.

```
fakedata <- read.csv("/Users/Adam/Desktop/fake.proficiencydata.csv")
```

Let's say the data is in a text file with tab separated values.

 fakedata - Notepad

File	Edit	View
<pre> Id Age Months.learning Proficiency.score First.language 1 22 12 1 Russian 2 23 15 2 German 3 19 16 4 French 4 20 12 5 French 4 21 15 6 Arabic 5 22 16 8 Russian 6 22 22 9 Russian 7 23 5 2 Russian 8 24 12 1 Russian 9 21 14 5 French 10 19 6 1 Arabic </pre>		

Make a dataset in excel or libraoffice calc

You can read the data in as a text file with the following code. Note that the `sep = ...` part means that the data are seperated by tabs. They could be seperated by anything though, you have to look at the data first before uploading it.

```
fakedata <- read.table("/Users/Adam/Desktop/fakedata.txt", sep="\t", header=TRUE)
```

You always want to check whether the data loaded okay, by looking at `head()` and then `summary()`. The latter is useful because it gives you a statistical summary of the data.

```
head(fakedata)
```

```
##   Id Age Months.learning Proficiency.score First.language
## 1  1  22             12                1      Russian
## 2  2  23             15                2       German
## 3  3  19             16                4       French
## 4  4  20             12                5       French
## 5  4  21             15                6       Arabic
## 6  5  22             16                8      Russian
```

```
summary(fakedata)
```

```
##           Id           Age      Months.learning Proficiency.score
## Min.      : 1.000    Min.      :19.00    Min.      : 5.00    Min.      :1.0
## 1st Qu.: 3.500    1st Qu.:20.50    1st Qu.:12.00    1st Qu.:1.5
## Median : 5.000    Median :22.00    Median :14.00    Median :4.0
## Mean     : 5.364    Mean     :21.45    Mean     :13.18    Mean     :4.0
## 3rd Qu.: 7.500    3rd Qu.:22.50    3rd Qu.:15.50    3rd Qu.:5.5
## Max.     :10.000    Max.     :24.00    Max.     :22.00    Max.     :9.0
## First.language
## Length:11
## Class :character
## Mode  :character
##
##
##
```

Note the type of information it gives you. Some of the data refers to means, medians quantiles etc. (stuff we learnt about last class). What about this Language one? It says its a “character”. This is a very important distinction. One common error code is when you apply a function that expects one type of data (numeric data), but the data is actually coded as a character.

```
mean(fakedata$First.language)
```

```
## Warning in mean.default(fakedata$First.language): argument is not numeric or
## logical: returning NA
```

```
## [1] NA
```

R cannot read your mind. Sometimes data are stored as characters even though they look like numbers. Let’s look in more detail at the way R codes data.

Atomic vector

An atomic vector is a single vector of data. Let’s pull out some atomic vectors from our fake data. The following should be read like “Make a new object from firstdata’s”First.language” vector/variable. Think of that \$ like a Saxon genitive 's - the thing on the right belongs to the larger dataframe of the thing on the left.

```
firstlanguage <- fakedata$First.language
months.learning <- fakedata$Months.learning
proficiency <- fakedata$Proficiency.score
```

Each of these *Objects* are atomic vectors. You can ask R if they are atomic vectors with the following function.

```
is.vector(firstlanguage)
```

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

The function `length()` as we saw last time gives you the length of the atomic vector.

```
length(firstlanguage)
```

```
## [1] 11
```

Each atomic vector store its value as a one-dimensional vector and each atomic vector can store only one type of data.

R recognizes 6 basic types of atomic vectors; doubles, integers, characters, logicals, and raw.

Doubles/Numeric

A double vector stores regular numbers. In datascience doubles are typically just referred to as numeric. We can check which of our vectors are numeric by asking.

```
is.numeric(months.learning)
```

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

```
is.numeric(firstlanguage)
```

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

Integers

Integers are numbers without decimals. We don't use these very often, but sometimes they can be useful if you need to round up or down and you are simulating data. In general we do everything with numerics/doubles.

```
head(fakedata)
```

```
##   Id Age Months.learning Proficiency.score First.language
## 1  1  22             12              1      Russian
## 2  2  23             15              2       German
## 3  3  19             16              4       French
## 4  4  20             12              5       French
## 5  4  21             15              6       Arabic
## 6  5  22             16              8      Russian
```

In principle we could code all of our numeric values as integers.

```
age <- as.integer(fakedata$Age)
age
```

```
## [1] 22 23 19 20 21 22 22 23 24 21 19
```

```
typeof(age)
```

```
## [1] "integer"
```

Sometimes integers can be useful doing math, because computers can give *floating point errors*...

```
sqrt(2)^2-2
```

```
## [1] 4.440892e-16
```

Wow, that is wrong (but barely). Using integers can fix the problem...

```
as.integer(sqrt(2)^2)-2
```

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

Those floating point errors are so insignificant that they rarely come up. I've never heard of a linguist having to pay attention to that, so for now just code everything as a double/numeric.

Characters

A character is a small piece of text. The vector `firstlanguage` is a character vector.

```
typeof(firstlanguage)
```

```
## [1] "character"
```

It doesn't make any sense to ask what the mean of a character value is and R will complain if you try.

```
mean(firstlanguage)
```

```
## Warning in mean.default(firstlanguage): argument is not numeric or logical:  
## returning NA
```

```
## [1] NA
```

Sometimes you read something in and you think its numeric when in fact R has analyzed it as a character.

Imagine we have data from a second proficiency data, but half of the students are exempt for some reason.

```
proficiencytest2 <- c(3,4,7,9,10,"exempt","exempt","exempt","exempt","exempt")
```

```
mean(proficiencytest2)
```

```
## Warning in mean.default(proficiencytest2): argument is not numeric or logical:
## returning NA
```

```
## [1] NA
```

You think, okay I'll just look at the first five.

```
mean(proficiencytest2[1:5])
```

```
## Warning in mean.default(proficiencytest2[1:5]): argument is not numeric or
## logical: returning NA
```

```
## [1] NA
```

It still doesn't work, because even though this refers to the first five numbers, the numbers have been coded as characters as you can see by the fact that they have quotes around them.

```
proficiencytest2
```

```
## [1] "3"      "4"      "7"      "9"      "10"     "exempt" "exempt" "exempt"
## [9] "exempt" "exempt"
```

So you have to change the data type to `as.numeric()`. Don't forget this! I guarantee you will be doing it a lot!

```
mean(as.numeric(proficiencytest2[1:5]))
```

```
## [1] 6.6
```

Logicals

A logical vector is one where the values are either true or false. Let's say we have a TRUE or FALSE variable for whether the students studied at FSU or something like that.

```
studiedatFSU <- c(TRUE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE)
```

```
typeof(studiedatFSU)
```

```
## [1] "logical"
```

Complex and Raw

Okay, so those are the types of data you will most likely use. There are two more types. Complex and Raw. Raw is for coding data directly in terms of bytes and complex is for imaginary numbers, which are useful for

umm.. quantum physics I think. We won't learn these in this class, just if you ever hear about these types, then you can look them up yourself.

Factors

Factor's are R's way of storing categorical data as numbers. Some statistical functions in R need your characters to already be factors and some just make them factors automatically.

```
firstlanguage.factor <- factor(firstlanguage)
```

```
firstlanguage.factor
```

```
## [1] Russian German French French Arabic Russian Russian Russian Russian
## [10] French Arabic
## Levels: Arabic French German Russian
```

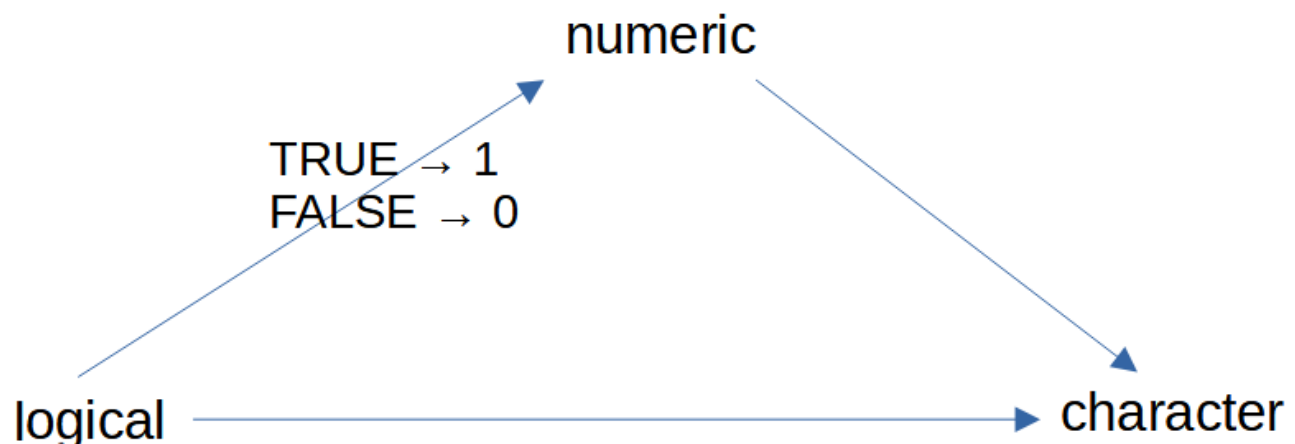
It'll become more obvious why this is important when we actually do statistical tests, but just note that factors are a thing with levels and some statistical tests make reference to them. You can see how R codes the factor with the following:

```
unclass(firstlanguage.factor)
```

```
## [1] 4 3 2 2 1 4 4 4 4 2 1
## attr(,"levels")
## [1] "Arabic" "French" "German" "Russian"
```

Coercion

Coercion refers to the process whereby R coerces data into a single type. Why do you need to know this term? Because R coerces your data (always) and because you can get coercion errors (e.g. some statistical operation can't coerce something that it needs to coerce).



Make a dataset in excel or libraoffice calc

You can always ask R to convert from one data type to another:

```
as.numeric("2.3")
```

```
## [1] 2.3
```

```
as.logical(1)
```

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

Lists

Lists are created the same way as vectors.

```
list1 <- list("a", "b", 1:20)
```

But there's an important difference ... Lists can code whole R objects (so you can have vectors in each position of a list). Or in other words, lists can be nested.

```
list2 <- list(1:20, 21:40)
```

```
vector1 <- c(1:20, 21:40)
```

Look the difference between these two objects.

Dataframes

Dataframes are two dimensional versions of lists, basically. Dataframes are like R's equivalent to an Excel spreadsheet and they are the most used type of datastructure.

Data frames store data in sequences of columns. Our fakedata is a dataframe.

Looking at real data and installing and loading packages

You can get new functions etc. by loading packages. Often when you look for a solution to a problem online you'll find a function that helps you figure it out. After you've found the function you have to figure out whether that function is in base R or whether it is in an R package. After that we download the R package and load it in R.

Okay, so here is some messy data, once you have unzipped the experiment file you should upload each of these datasets.


```
bda <- read.csv("/Users/Adam/Desktop/experiment/bda.csv", header=TRUE)
mco <- read.csv("/Users/Adam/Desktop/experiment/mco.csv", header=TRUE)
mct <- read.csv("/Users/Adam/Desktop/experiment/mct.csv", header=TRUE)
pyr <- read.csv("/Users/Adam/Desktop/experiment/pyr.csv", header=TRUE)
```

The first problem with these datasets is that they do not have the speaker information. So install tidyverse and then do the following.

```
bda$subject <- replace(bda$subject, bda$subject=="exp1", "BDA")
mco$subject <- replace(mco$subject, mco$subject=="exp1", "MCO")
mct$subject <- replace(mct$subject, mct$subject=="exp1", "MCT")
pyr$subject <- replace(pyr$subject, pyr$subject=="exp1", "PYR")
```

Then we can stick all the data together into one dataset. Remember that rbind() glues together data sets vertically.

```
chacobo.mcf.df <- rbind(bda,mco,mct,pyr)
```

```
head(chacobo.mcf.df)
```

```
##  subject      stimulus      response reactionTime
## 1    BDA janaquë_103_80  jánaquë dejó      2.981856
## 2    BDA janaquë_143_40 janáquë vomitó    2.034328
## 3    BDA janaquë_118_110 janáquë vomitó    6.919692
## 4    BDA janaquë_114_100 jánaquë dejó      1.674339
## 5    BDA janaquë_118_90  janáquë vomitó    2.857278
## 6    BDA janaquë_110_100 jánaquë dejó      2.850445
```

```
summary(chacobo.mcf.df)
```

```
##      subject      stimulus      response      reactionTime
## Length:1620      Length:1620      Length:1620      Min.   : 0.0441
## Class :character  Class :character  Class :character  1st Qu.: 1.3296
## Mode  :character  Mode  :character  Mode  :character  Median : 1.6709
##                                     Mean   : 2.0245
##                                     3rd Qu.: 2.1385
##                                     Max.   :111.7687
```

I googled how to split strings and then found out about the function strsplit(). Here's what happens when I use it on the stimulus.

```
head(strsplit(chacobo.mcf.df$stimulus, "_"))
```

```
## [[1]]
## [1] "janaquë" "103"      "80"
##
## [[2]]
## [1] "janaquë" "143"      "40"
##
## [[3]]
## [1] "janaquë" "118"      "110"
##
## [[4]]
## [1] "janaquë" "114"      "100"
##
## [[5]]
## [1] "janaquë" "118"      "90"
##
## [[6]]
## [1] "janaquë" "110"      "100"
```

So it looks like what this does is split each element in the vector by whatever you put in the argument as. In this case I specified the argument as an underscore. This is a list because each element in it is an R object (its own vector) with three elements (i.e. its a nested list).

We want to turn this into a data frame. I googled turn nested list into dataframe and found the following code would work.

```
head(as.data.frame(do.call(rbind, strsplit(chacobo.mcf.df$stimulus, "_"))))
```

```
##      V1  V2  V3
## 1 janaquë 103  80
## 2 janaquë 143  40
## 3 janaquë 118 110
## 4 janaquë 114 100
## 5 janaquë 118  90
## 6 janaquë 110 100
```

Let's make another dataframe like this.

```
chacobo.stimuli <- data.frame(do.call(rbind, strsplit(chacobo.mcf.df$stimulus, "_")))
```

Now we have to cbind these data to our original dataset.

```
chacobo.mcf.df <- cbind(chacobo.mcf.df, chacobo.stimuli)
```

```
head(chacobo.mcf.df)
```

```
##   subject      stimulus      response reactionTime      X1 X2 X3
## 1    BDA janaquë_103_80  jánaquë dejó    2.981856 janaquë 103 80
## 2    BDA janaquë_143_40 janáquë vomitó    2.034328 janaquë 143 40
## 3    BDA janaquë_118_110 janáquë vomitó    6.919692 janaquë 118 110
## 4    BDA janaquë_114_100  jánaquë dejó    1.674339 janaquë 114 100
## 5    BDA janaquë_118_90  janáquë vomitó    2.857278 janaquë 118 90
## 6    BDA janaquë_110_100  jánaquë dejó    2.850445 janaquë 110 100
```

That's nice, but there's a few things we want to do now still. Three of our vectors are named X1, X2 and X3. We can rename things with the function `rename()` which is part of `tidyverse()`

```
library(tidyverse)
```

```
## — Attaching packages ————— tidyverse 1.3.2 —
## ✓ ggplot2 3.3.6      ✓ purrr 0.3.4
## ✓ tibble 3.1.8       ✓ dplyr 1.0.10
## ✓ tidyr 1.2.0        ✓ stringr 1.4.1
## ✓ readr 2.1.3        ✓ forcats 0.5.2
## — Conflicts ————— tidyverse_conflicts() —
## ✗ dplyr::filter() masks stats::filter()
## ✗ dplyr::lag()     masks stats::lag()
```

```
chacobo.mcf.df <- chacobo.mcf.df %>% rename("pitch.f0" = "X2",
                                           "duration.ms" = "X3")
```

We don't need the "stimulus" variable and the X1 variable is just junk, so we can make a subdatabase that just has what we are interested in. This can be done with the function `select()`, which selects the vectors we are interested in.

```
mcf.chacobo.results <- subset(chacobo.mcf.df, select = c(subject, stimulus, pitch.f0, duration.ms, reactionTime, response))
```

Great! Now let's look at the distribution of pitch and duration. This time we are going to use base R and `ggplot`. `ggplot()` is a graphics program for R that allows you to make prettier graphs. `Tidyverse()` has `ggplot2` in it, but just in case load it again. I'll continue to give you the code for base R so you can fall back on it in case you can't get what you want with `ggplot`.

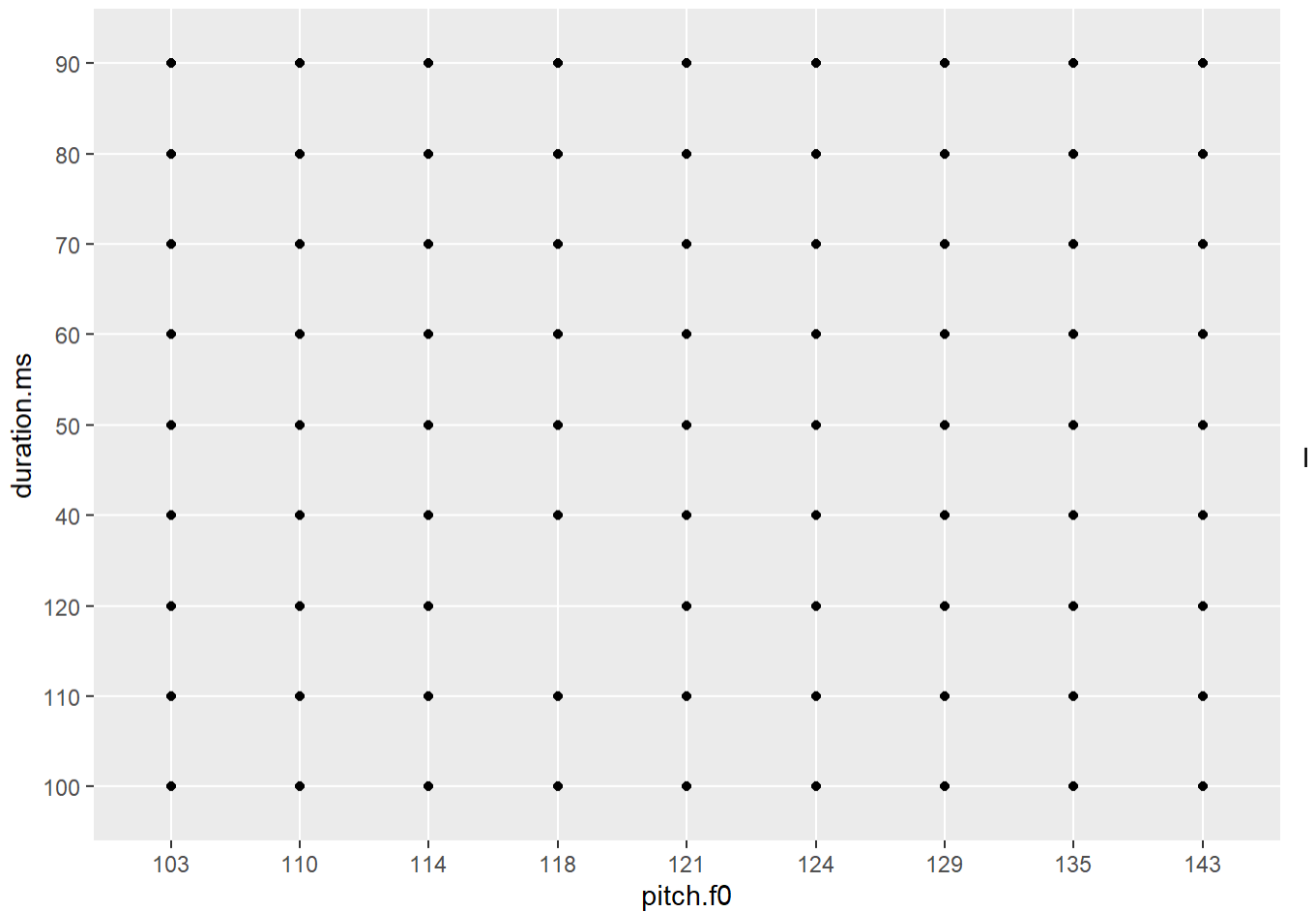
```
library(ggplot2)
```

Looking at the data with ggplot2

Let's look at the distribution of `f0` and duration in the data. Note that from now on we are going to save our plots as objects that we can call on rather than just calling them in the code.

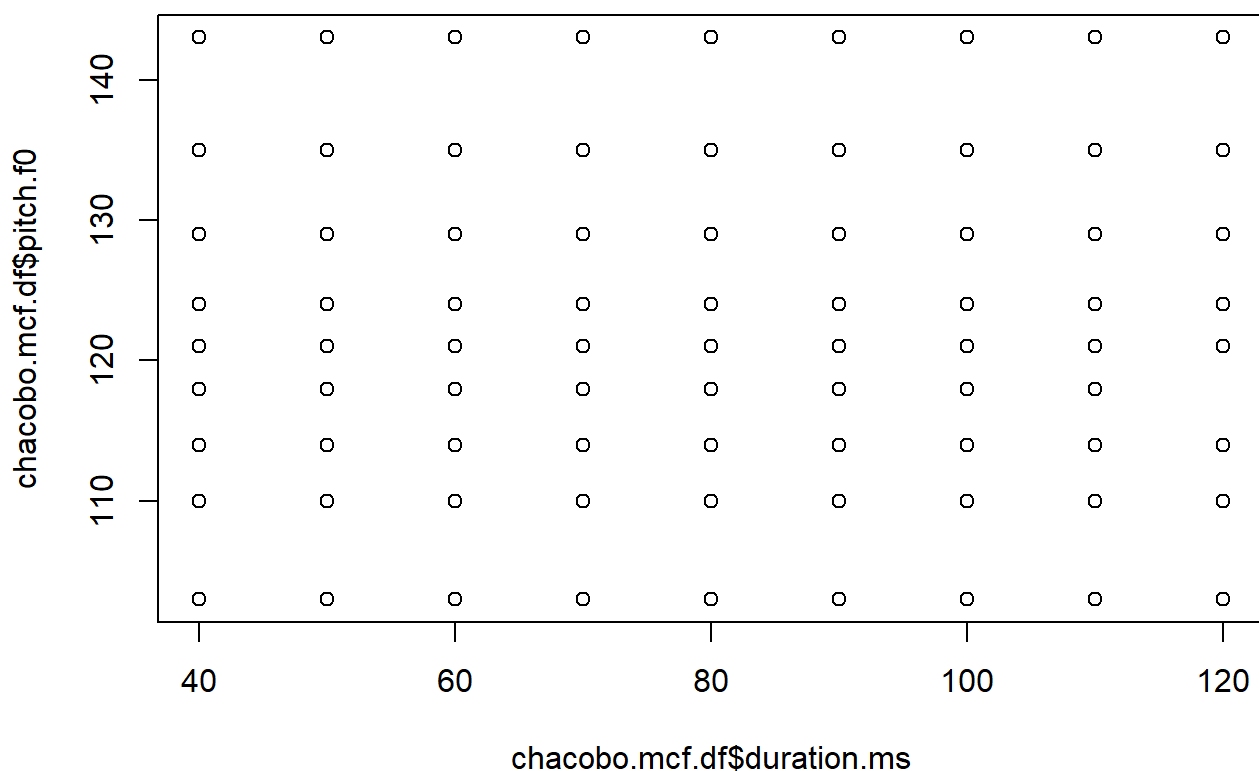
```
scatterplot1.ggplot <- ggplot(mcf.chacobo.results, aes(x=pitch.f0, y=duration.ms)) + geom_point()
```

```
scatterplot1.ggplot
```



made these data, so this is what they should look like. You can get this same plot in base R with the following code.

```
scatterplot1.baseR <- plot(chacobo.mcf.df$duration.ms, chacobo.mcf.df$pitch.f0)
```



```
scatterplot1.baseR
```

```
## NULL
```

There's one data point that's missing for some reason. A flaw in the experimental design. Let's see if there is a relationship between reaction time and the pitch values.

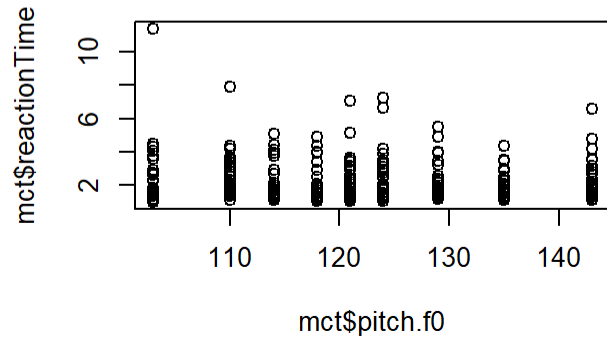
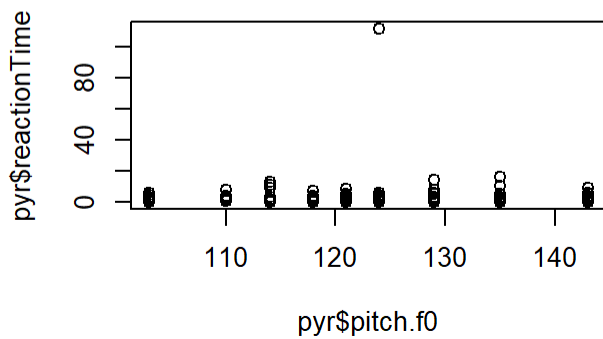
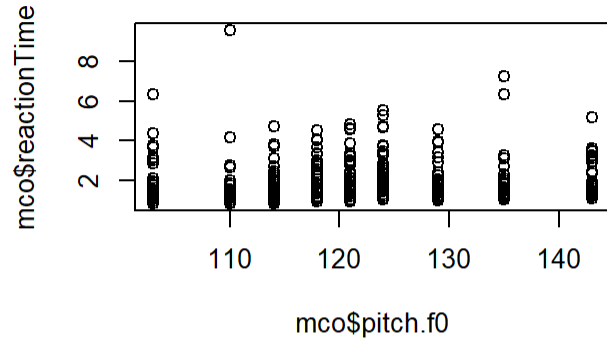
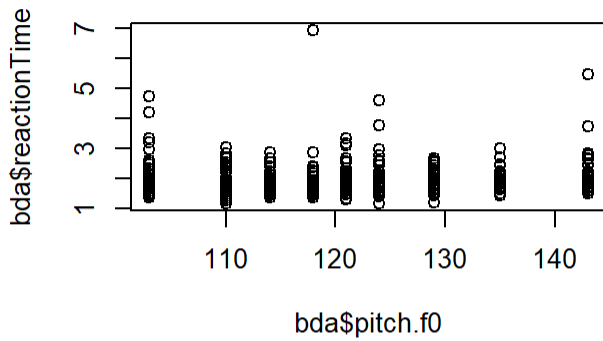
We want to look at it speaker by speaker, so we are really going to make four plots. We can set this up with the function `par(mfrow(2,2))`.

This will be substantially easier if we can just call on data for individual speakers at the moment (although there are more complicated ways of building the code that doesn't do this)

```
bda <- chacobo.mcf.df[chacobo.mcf.df$subject=="BDA",]
mco <- chacobo.mcf.df[chacobo.mcf.df$subject=="MCO",]
pyr <- chacobo.mcf.df[chacobo.mcf.df$subject=="PYR",]
mct <- chacobo.mcf.df[chacobo.mcf.df$subject=="MCT",]
```

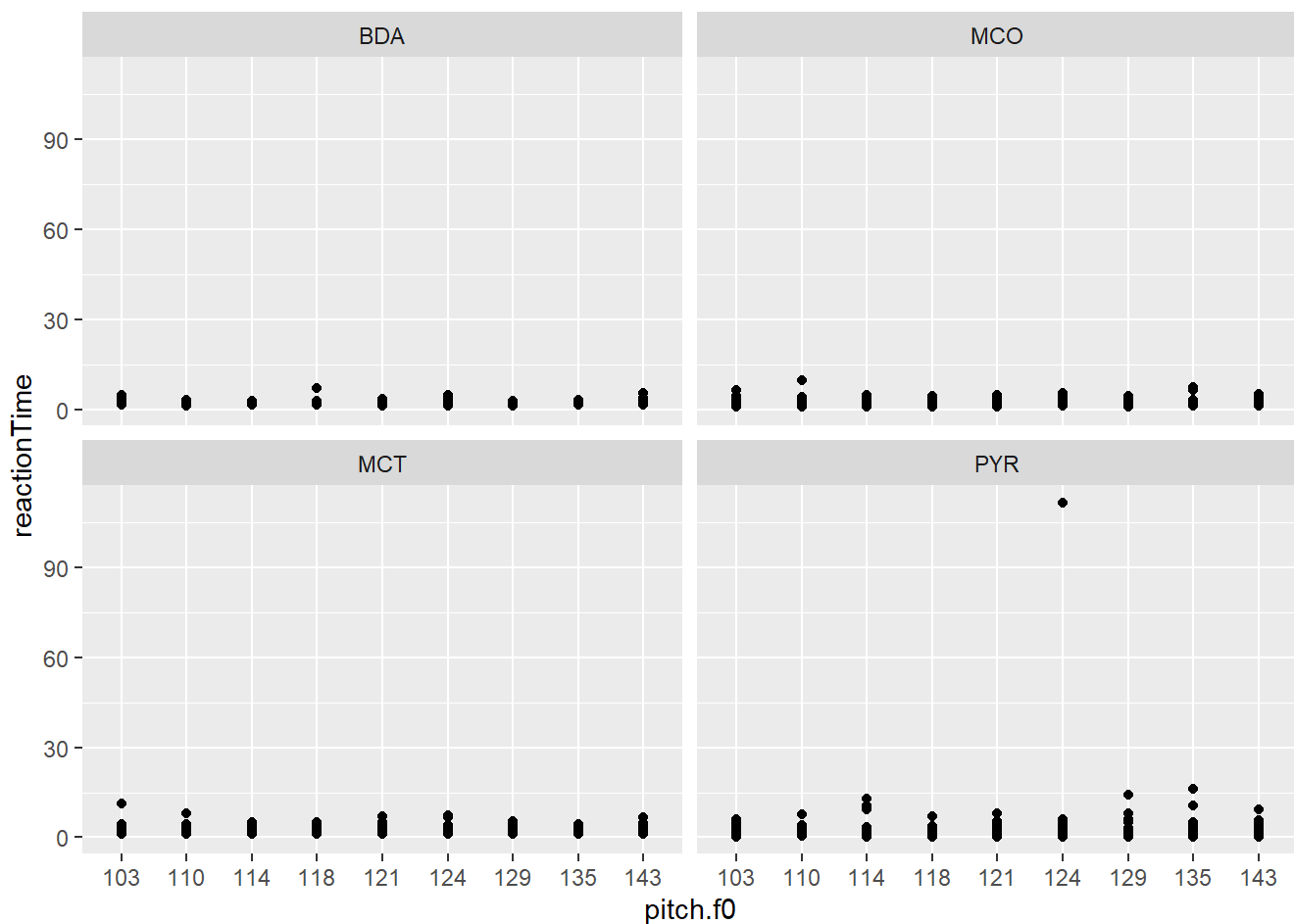
Now let's look at pitch against reaction time for the four speakers.

```
par(mfrow=c(2,2))
plot(bda$pitch.f0, bda$reactionTime)
plot(mco$pitch.f0, mco$reactionTime)
plot(pyr$pitch.f0, pyr$reactionTime)
plot(mct$pitch.f0, mct$reactionTime)
```



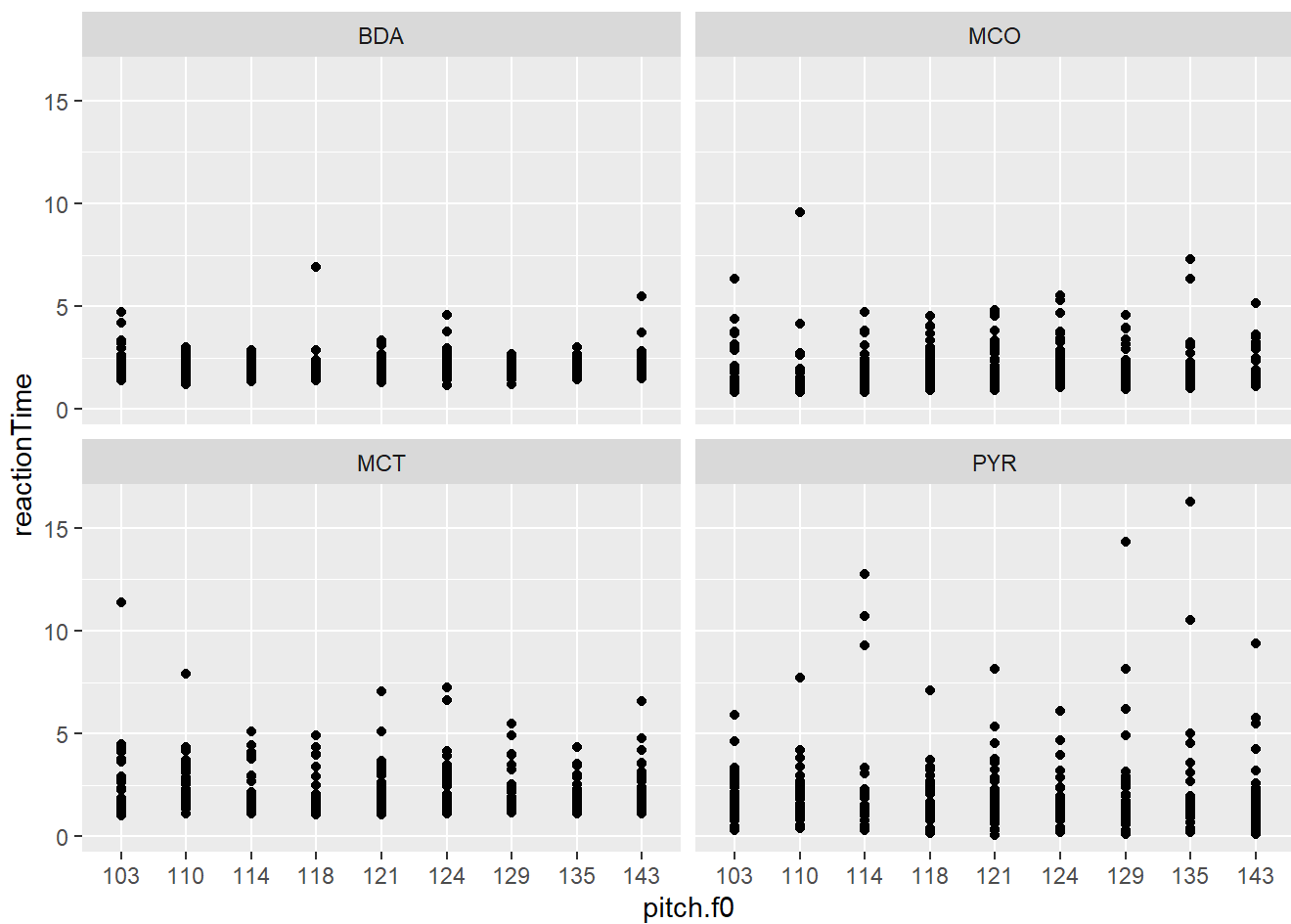
```
ggplot(econdatalong, aes(x=Country, y=value))+ geom_bar(stat='identity', fill="forest green")+
facet_wrap(~measure, ncol=1)
```

```
ggplot(chacobo.mcf.df, aes(x=pitch.f0, y=reactionTime))+
  geom_point()+
  facet_wrap(~subject, ncol=2)
```



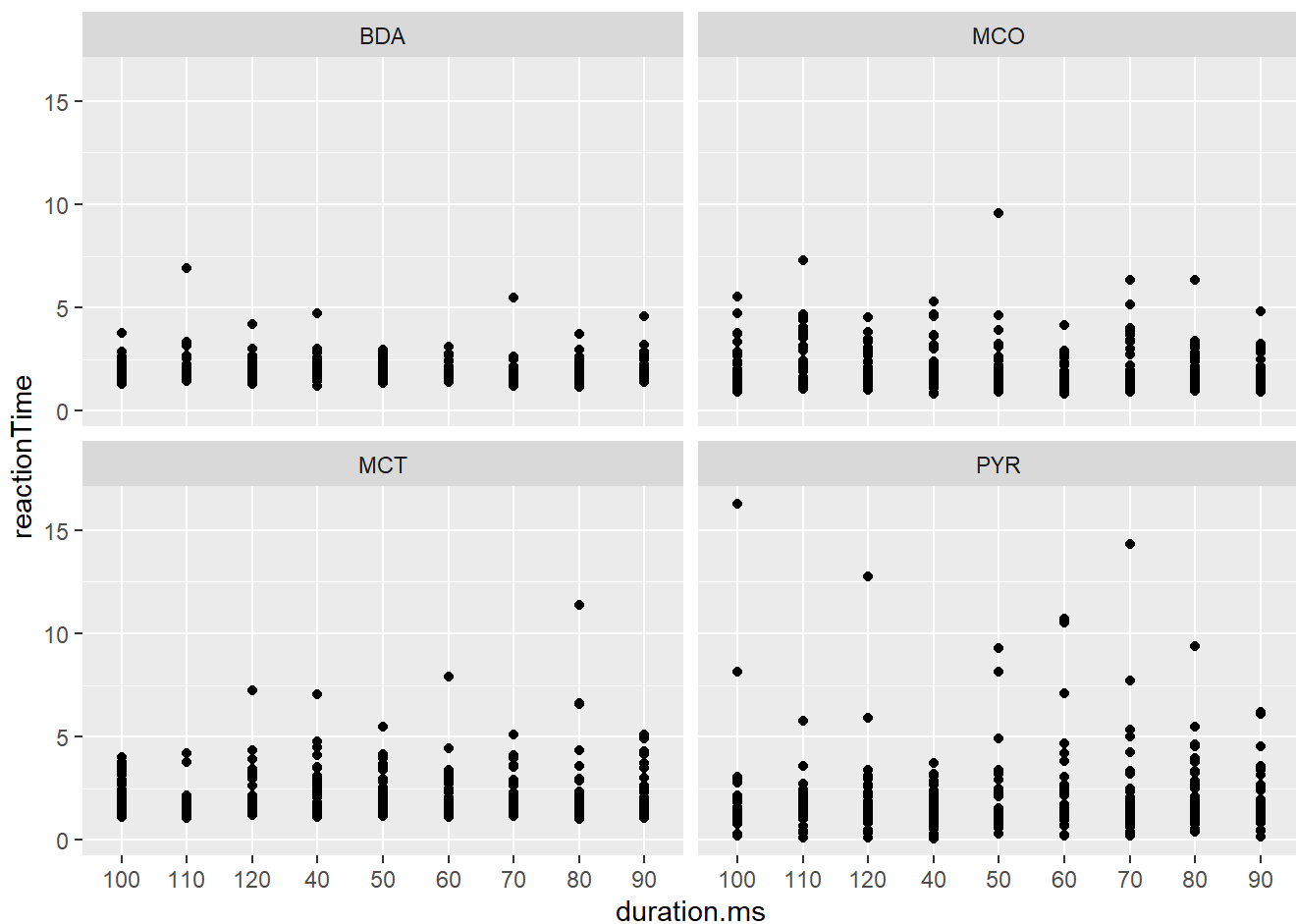
It looks like PYR just stopped doing the experiment for like 2 minutes, which is why there is this one super far out datapoint. Let's remove data that are above 20 seconds.

```
chacobo.mcf.df$reactionTime <- as.numeric(chacobo.mcf.df$reactionTime)
chacobo.mcf.df <- filter(chacobo.mcf.df, reactionTime<20)
ggplot(chacobo.mcf.df, aes(x=pitch.f0, y=reactionTime))+
  geom_point()+
  facet_wrap(~subject, ncol=2)
```



We can look at the same relationship for duration.

```
ggplot(chacobo.mcf.df, aes(x=duration.ms, y=reactionTime))+
  geom_point()+
  facet_wrap(~subject, ncol=2)
```

We can actually look at this in 3D with

```
library(plotly)
```

```
##  
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':  
##  
##   last_plot
```

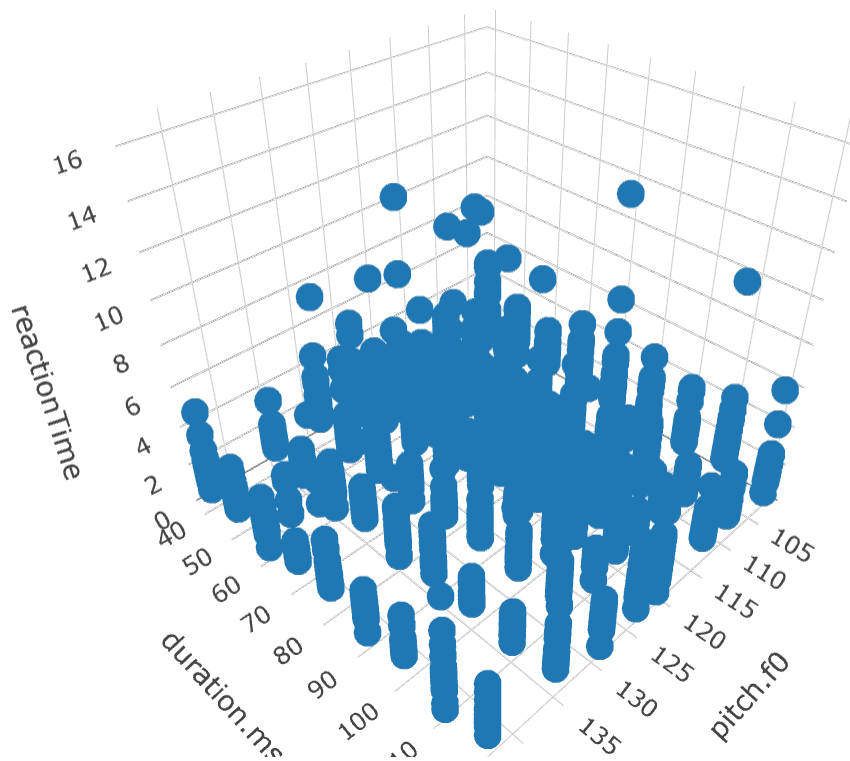
```
## The following object is masked from 'package:stats':  
##  
##   filter
```

```
## The following object is masked from 'package:graphics':  
##  
##   layout
```

```
chacobo.mcf.df$pitch.f0 <- as.numeric(chacobo.mcf.df$pitch.f0)
chacobo.mcf.df$duration.ms <- as.numeric(chacobo.mcf.df$duration.ms)
plot_ly(chacobo.mcf.df, x = ~pitch.f0, y = ~duration.ms, z=~reactionTime)
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter3d' trace seems appropriate.
##   Read more about this trace type -> https://plotly.com/r/reference/#scatter3d
```

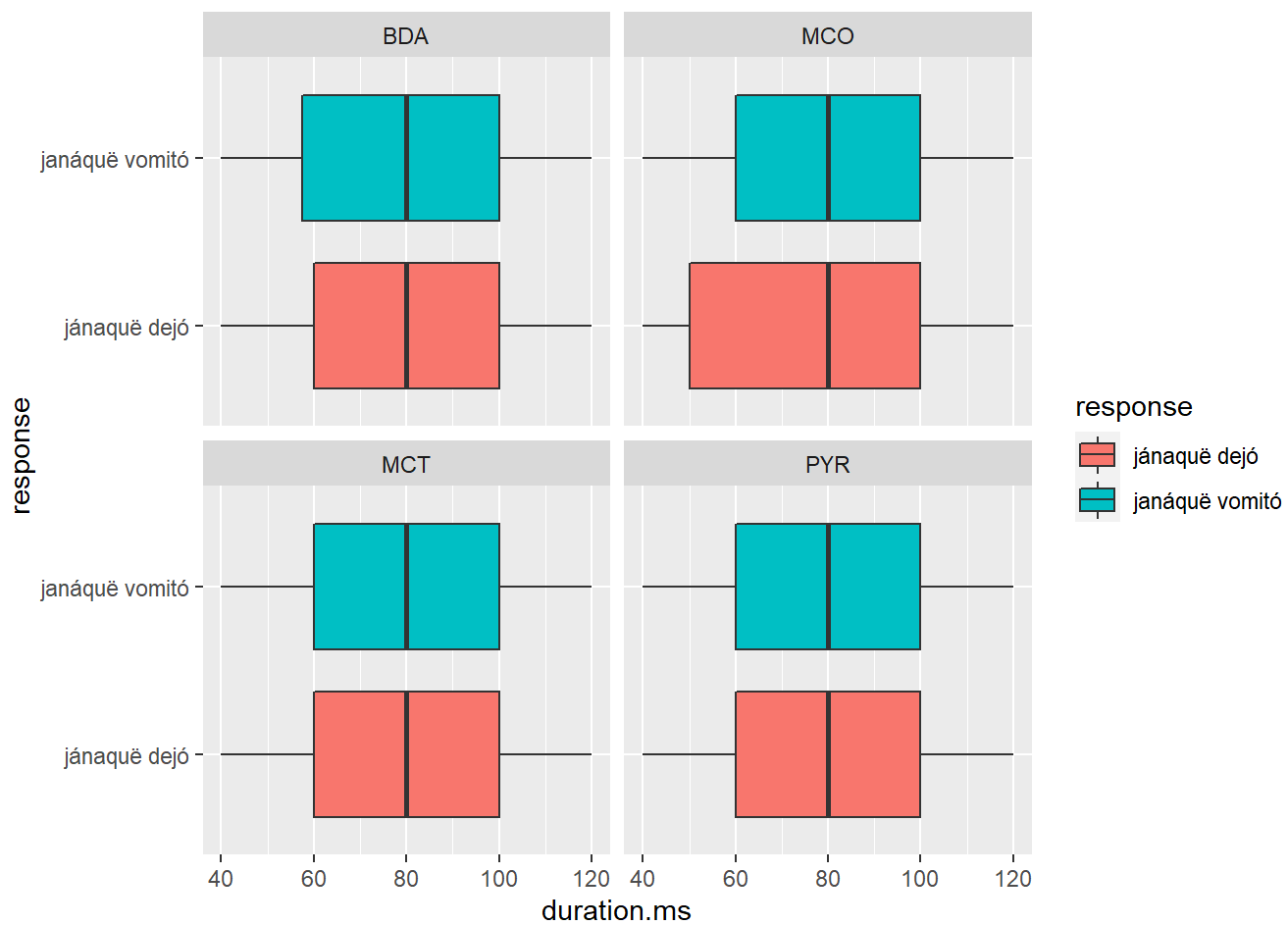
```
## No scatter3d mode specified:
##   Setting the mode to markers
##   Read more about this attribute -> https://plotly.com/r/reference/#scatter-mode
```



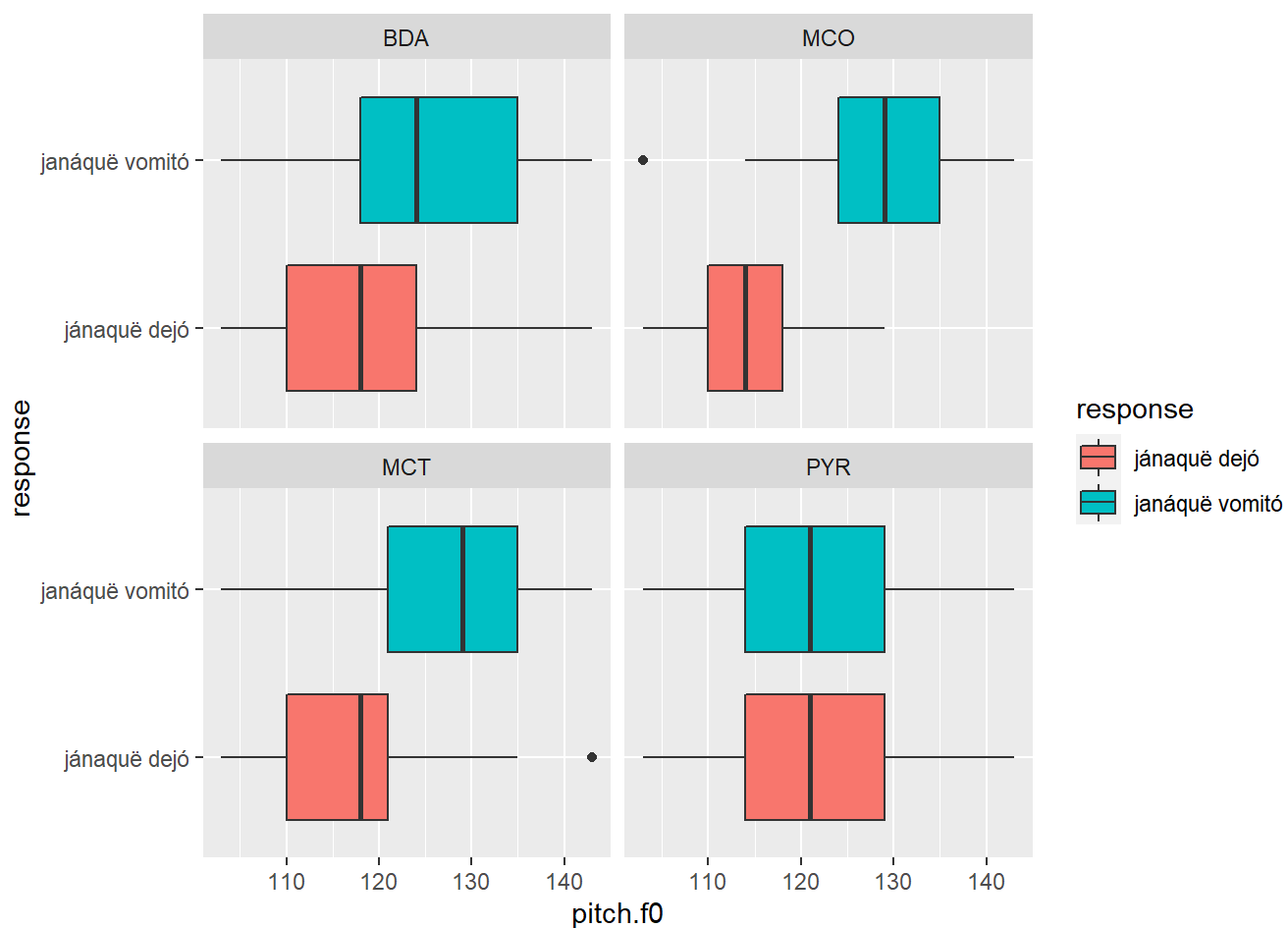
So it doesn't really look like much is going on with response time. Let's look at the binary response variable. Whether they thought the stimuli voice was saying "jánaque" 'leave' or "janáque" 'vomit'.

Boxplots in ggplot

```
ggplot(chacobo.mcf.df, aes(x=duration.ms, y=response, group=response)) +
  geom_boxplot(aes(fill=response))+
  facet_wrap(~subject, ncol=2)
```



```
ggplot(chacobo.mcf.df, aes(x=pitch.f0, y=response, group=response)) +
  geom_boxplot(aes(fill=response))+
  facet_wrap(~subject, ncol=2)
```



So we can conclude the following: (i) pitch seems to matter for which the speakers perceive; (ii) PYR was checked out during the experiment.