Class 3 notes

Anita Kurm

9/18/2019

Part 1: R Markdown intro

(you can use ## so something you write looks like a header 2 like this one)

1. Make a new R Markdown (with default: html as output)

(you can use ### so it looks like header 3)

Let's look at the completely new .Rmd file together :)

2. Let's see how to work with .Rmd files.

First, change author name to yours - it'll be your notes and comments that will make this file way more valuable!

Then we go to the setup chunk.

A setup chunk {r setup}

(you can use #### so it looks like header 4... you get it)

Below you can see a setup chunk. Any chunk can be a setup chunk if you write {r setup} at its beginning.

In this case we also ask R to not include the chunk in the output of this file by writing the setting {..., include=FALSE}.

In the setup chunk, you can change default settings for all chunks (e.g. see Command 1 and Command 2).

'knitr::' - means that we want to use functions specifically from the package knitr.

```
#Command 1: find default chunk options (opts_chunk), and set the setting echo (whether to show your cod
knitr::opts_chunk$set(echo = TRUE)


#Command 2: set a new working directory to ALL chunks - not just the current chunk
#You need to remove the # for it to work
#knitr::opts_knit$set(root.dir = 'relative_path_to_root_from_Rmd' )

library(pacman)

pacman::p_load(tidyverse, data.table, lme4)
```

Making a new chunk

You can make a new R code chunk using keyboard shortcut Ctrl + Alt + I (Cmd + Option + I on macOS). Try it! Once the code chunk appears, try importing your data inside of it, using read.csv() function.

It will only work if the file is inside of your working directory

```
df <- read.table("PTD.txt", header = TRUE, sep = ",")
```

Troubleshooting if you can't import data:

1. Check your working directory getwd()
2. Go to this working directory folder on your compurer and check if the file is out there in the open, not in a subfolder
3. If the file is not there - put it there!
4. Make sure that you wrote the name of the file correctly in R, using quotation marks: e.g. "filename.csv"
5. If it still does not work, check if you have any other folders on your computer with the same name as your working directory (you might be putting your file to an irrelevant folder instead of working directory)
6. If it still doesn't work, ask me or Fabio.

A new chunk with an output

Now make a new chunk! In it, find what class of data is stored in the column 'balloon_balance'. Use class() function we've learned in the first week. How does one refer to a column from a dataframe?

```
class("balloon_balance")
```

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

```
class(df$balloon_balance)
```

```
## [1] "numeric"
```

Knit

Now press the Knit button! See the result!

When you click the **Knit** button, R will generate an output of your Markdown script. In our case we chose the output to be an HTML document. As you can see, the document shows your text, your chunks of code and outputs of these chunks of code.

This makes it really nice and easy for Fabio to read through your portfolio assignments :)

Knitting repeatedly

Now go to the setup chunk and remove the 'include = FALSE' part, so only {r setup} stays at the beginning of the chunk.

Press Knit button again!

Every time you press Knit button - your script is saved and your html output is rewritten with new changes. (Means you will not have 1000 html documents from every time you knit - super nice!)

Part 2: New functions (if we have time)

We will look at some new functions from the package tidyverse. Make a new chunk and load the tidyverse library - use either pacman::p_load() or library().

summarise() function

This function collapses the whole dataframe into a single summary. For this function to work, it has to follow certain pattern. First, you specify the data frame you want to summarise and then you say what values you want to have in your summary.

See examples in the chunk below, try to run it and make sense of results:

```
#Make a summary with just one value - the average shoesize
summarise(df,mean(shoesize))
```

```
##   mean(shoesize)
## 1       40.64516
```

```
#summary with several values: the average shoesize and its standard deviation
summarise(df,mean(shoesize),sd(shoesize))
```

```
##   mean(shoesize) sd(shoesize)
## 1       40.64516     2.975841
```

summarise() is quite useless by itself, but everything changes when we **group** our data!

group_by() function

group_by() takes an existing data frame and converts it into a data frame grouped by some principle. See examples in the chunk below:

```
#group data by gender
grouped_bygender <- group_by(df, gender)

#group data by native language
grouped_bylanguage <- group_by(df, native_Danish)
```

When we apply different functions/operations on grouped data, we get outcomes for every group.

For instance:

```
summarise(grouped_bygender, mean(breath_hold))
```

```
## # A tibble: 2 x 2
##   gender `mean(breath_hold)`
##   <fct>                <dbl>
## 1 female                49.7
## 2 male                  65.5
```

As you can see, by using summarise() on grouped data, we can already get some insight into our data and possible findings ("Guys are on average better at holding breath?!")

Another example:

```
summarise(grouped_bylanguage, mean(tongue_twist), sd(tongue_twist))
```

```
## # A tibble: 2 x 3
##   native_Danish `mean(tongue_twist)` `sd(tongue_twist)`
##   <fct>                        <dbl>              <dbl>
## 1 No                            42.6               5.35
## 2 Yes                           46.6              12.4
```

Knowing about mean and standard deviation, what do you think this summary shows?

Pipes

Let's rehearse how we grouped data and summarised it. In the chunk below I group data by gender and then get a summary, that shows the mean shoe size in females and males. Run it and see for yourself:

```
grouped_data <- group_by(df, gender)
summary_shoe_bygender <- summarise(grouped_data, mean(shoesize))

#call the variable to see the summary output
summary_shoe_bygender
```

```
## # A tibble: 2 x 2
##   gender `mean(shoesize)`
##   <fct>             <dbl>
## 1 female             38.7
## 2 male               43.7
```

This seems a bit clumsy to write all of this... There must be a better way!

There is a better way and it's called pipes!!!

You might ask - what is pipes? It's a very efficient way to write stuff down in R and it looks like this: %>%

%>% reads 'send the resulting dataframe(s) to the following function'

You can type pipes faster by using the shortcut: cmd+shift+M (MacOS) or ctrl+shift+M (Others)

Let's see how to do pipes in practice. We will try to recreate the same summary as in the previous code chunk:

```
summary_shoe_bygender2 <- df %>%  #send the df to the next line:
  group_by(gender) %>% #group the data from the previous line by gender and send the result to the next
  summarise(mean(shoesize)) #summarise mean shoesize for data from the previous line

summary_shoe_bygender2
```

```
## # A tibble: 2 x 2
##   gender `mean(shoesize)`
##   <fct>             <dbl>
## 1 female             38.7
## 2 male               43.7
```

As you can see, summary_shoe_bygender and summary_shoe_bygender2 are the same! Both ways work, but pipes make your code more efficient and much easier to read.

Exercises

Try the following exercises to practice summarise(), group_by(), mean() and pipes. Make a separate code chunk for every question.

1. Is there a gender difference when it comes to balloon balancing?
2. Is there a relation between sound level preference and which cola was chosen?
3. Does handedness influence average tongue twisting speed?

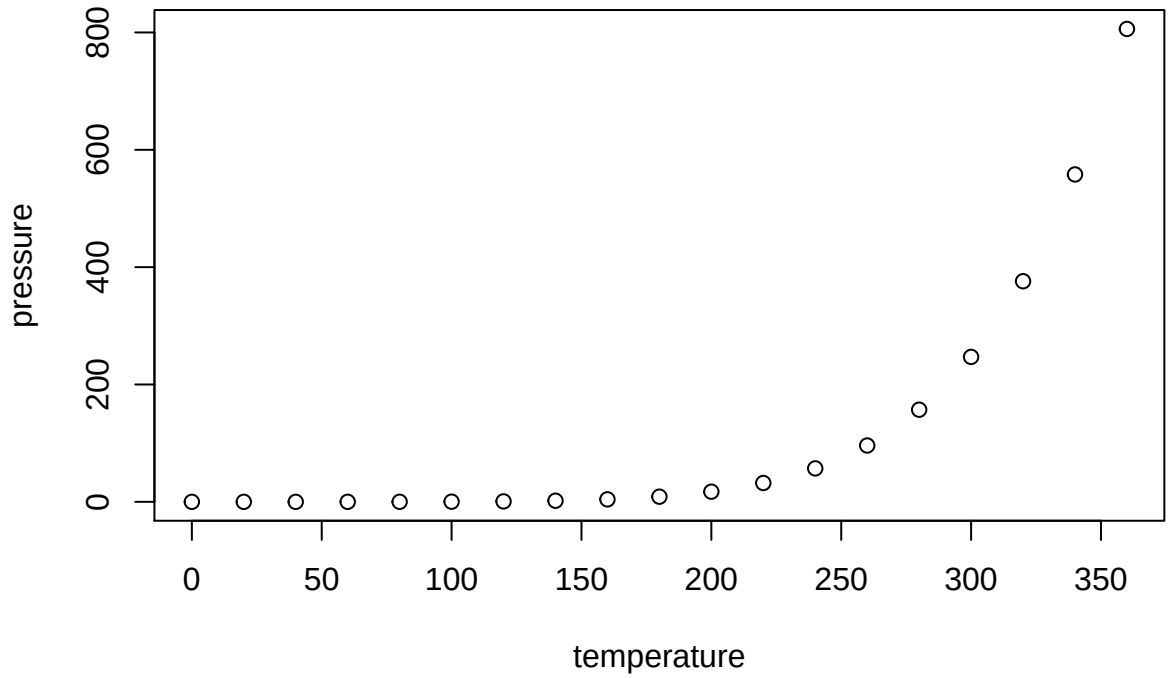
3a. Can you add a column to the summary, which contains number of people in each group (e.g. number of right handed people), hint: look at the n() function

3b. What does number of people tell us about the estimates of the mean? Are our results reliable?

Part 3: Including Plots

R can draw plots, but they are not that pretty.

An example that is always included when you start a new Rmd file - plotting some pressure data from package 'pressure':



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

We want to see all of your code, guys! So go ahead and fix it - make the code visible by removing the whole echo part, leaving just {r pressure}.

Knit again! Is the code there? Yes! And we want it there!

The plot is still ugly though...

Ggplot 2

Ggplot 2 is a package for data visualization - a part of tidyverse. If you have tidyverse installed, you have ggplot2. Make a new chunk and read in the tidyverse library, if you haven't done so yet

We will go through basics step by step (I basically copied this tutorial: http://r-statistics.co/ggplot2-Tutorial-With-R.html)

Base setup

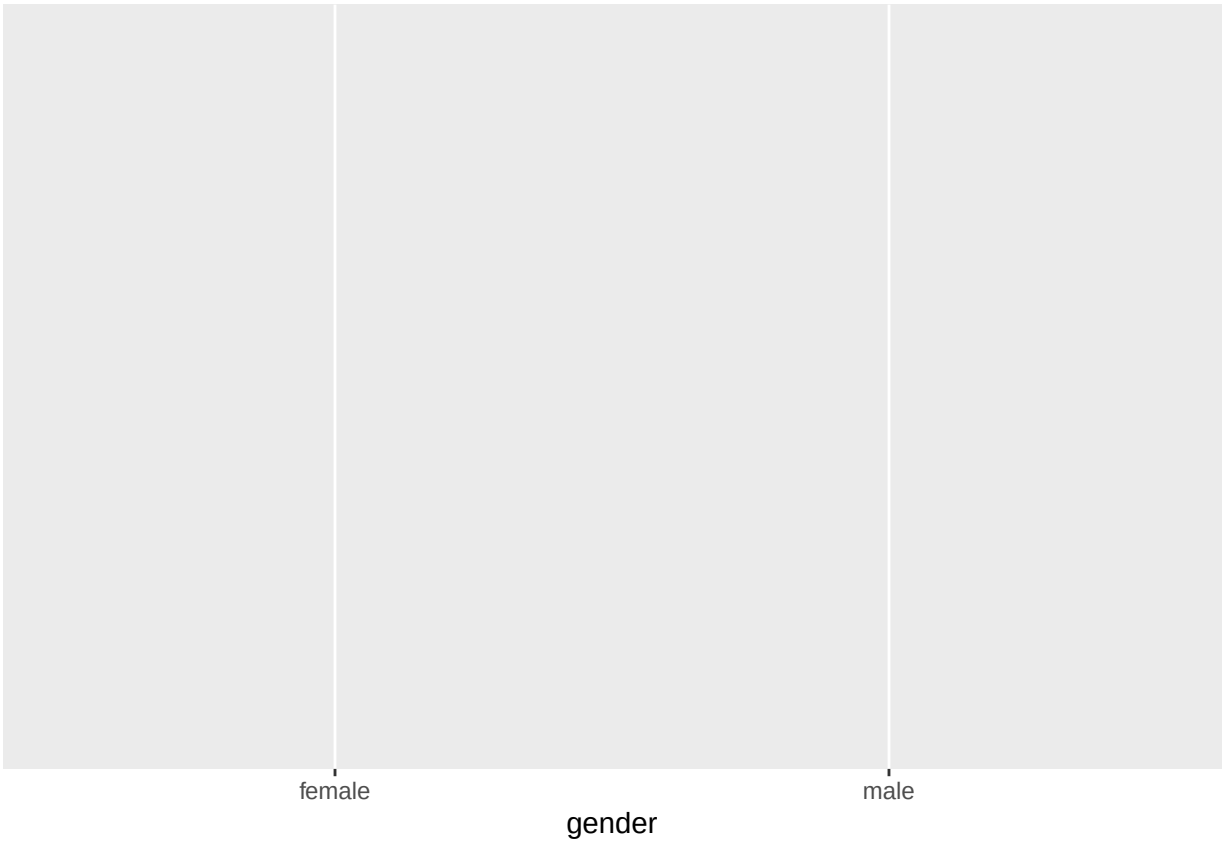
First, you need to tell ggplot what dataset to use. This is done using the ggplot(df) function, where df is a dataframe that contains all features needed to make the plot. This is the most basic setup.

```
ggplot(df) # the most basic setup
```


Optionally you can add whatever aesthetics you want to apply to your ggplot (inside aes() argument) - such as X and Y axis by specifying the respective variables from the dataset. The variable based on which the color, size, shape and stroke should change can also be specified here itself. The aesthetics specified here will be inherited by all the geom layers you will add later. See examples below:

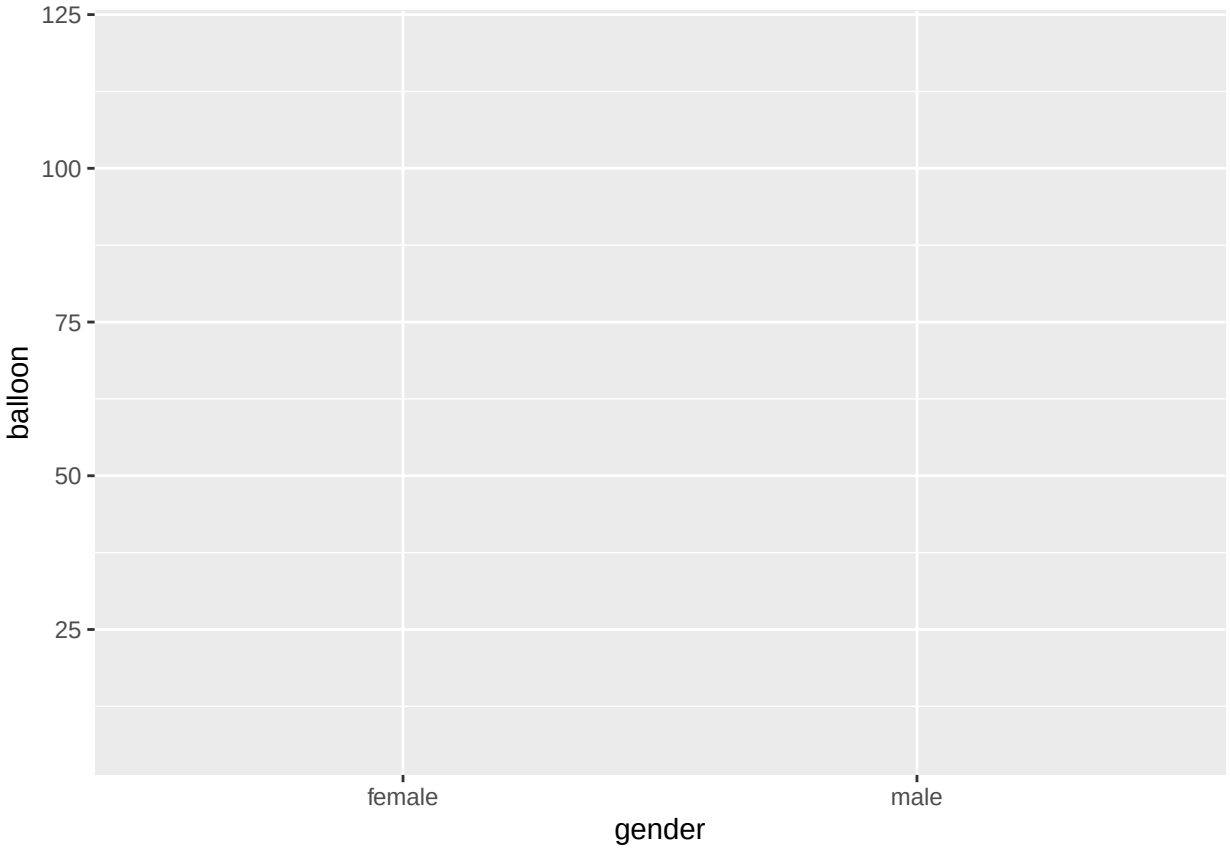
```
#Examples of setups with different aesthetics:

ggplot(df, aes(x=gender))
```



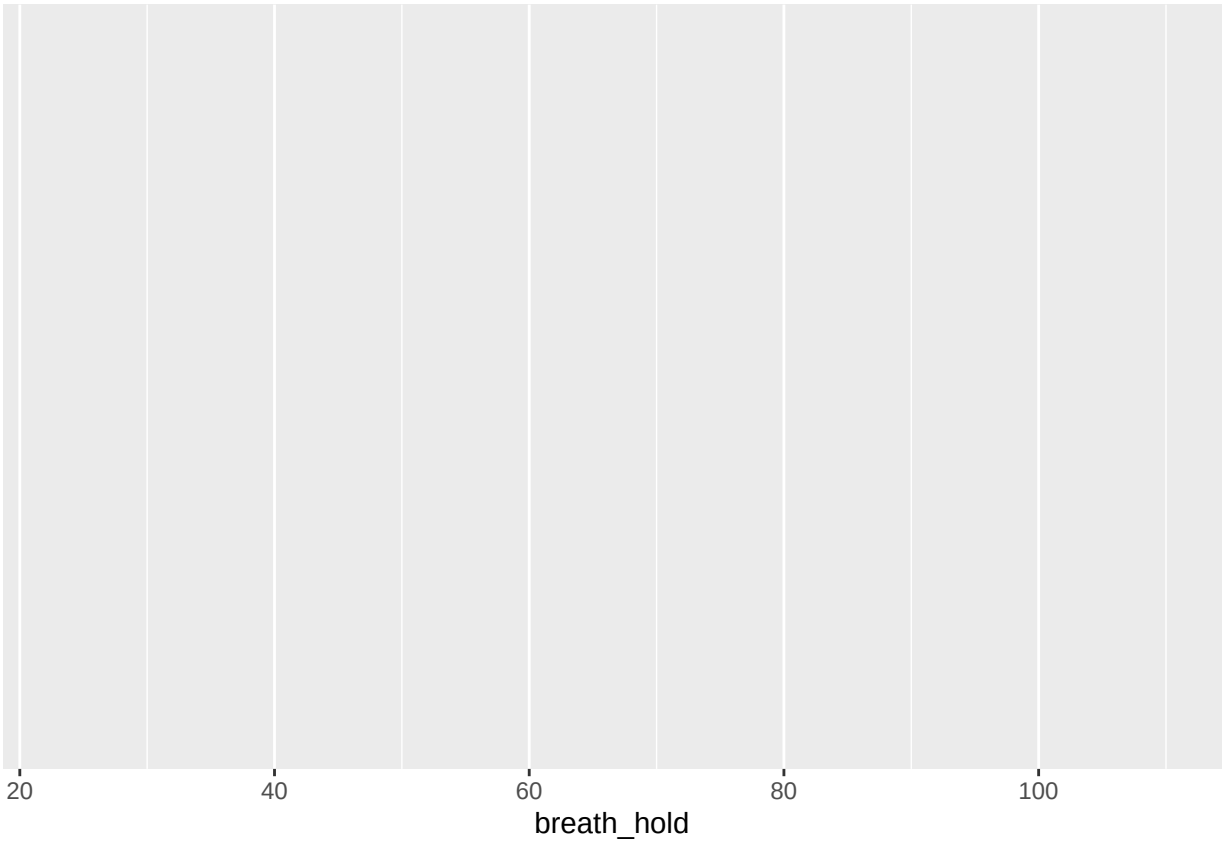

```
# if only X-axis is known. The Y-axis can be specified later

ggplot(df, aes(x=gender, y=balloon))
```




```
# if both X and Y axes are fixed for all layers.

ggplot(df, aes(x=breath_hold, color=gender))
```




```
# Each category of the 'gender' variable will now have a distinct  color, once a geom is added.
```

As you can see, this graph does not show any data yet, we need to give ggplot more information! We can do it by adding 'geoms' (layers) to our base setup.

Geoms

Once the base setup is done, you can append the geoms one on top of the other by using the plus sign! So technically all plots can be produced with the same structre:

basesetup + geom 1 + geom 2 + ...

There is a lot of different geoms in the ggplot2 package. See the cheatsheet to have an idea: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

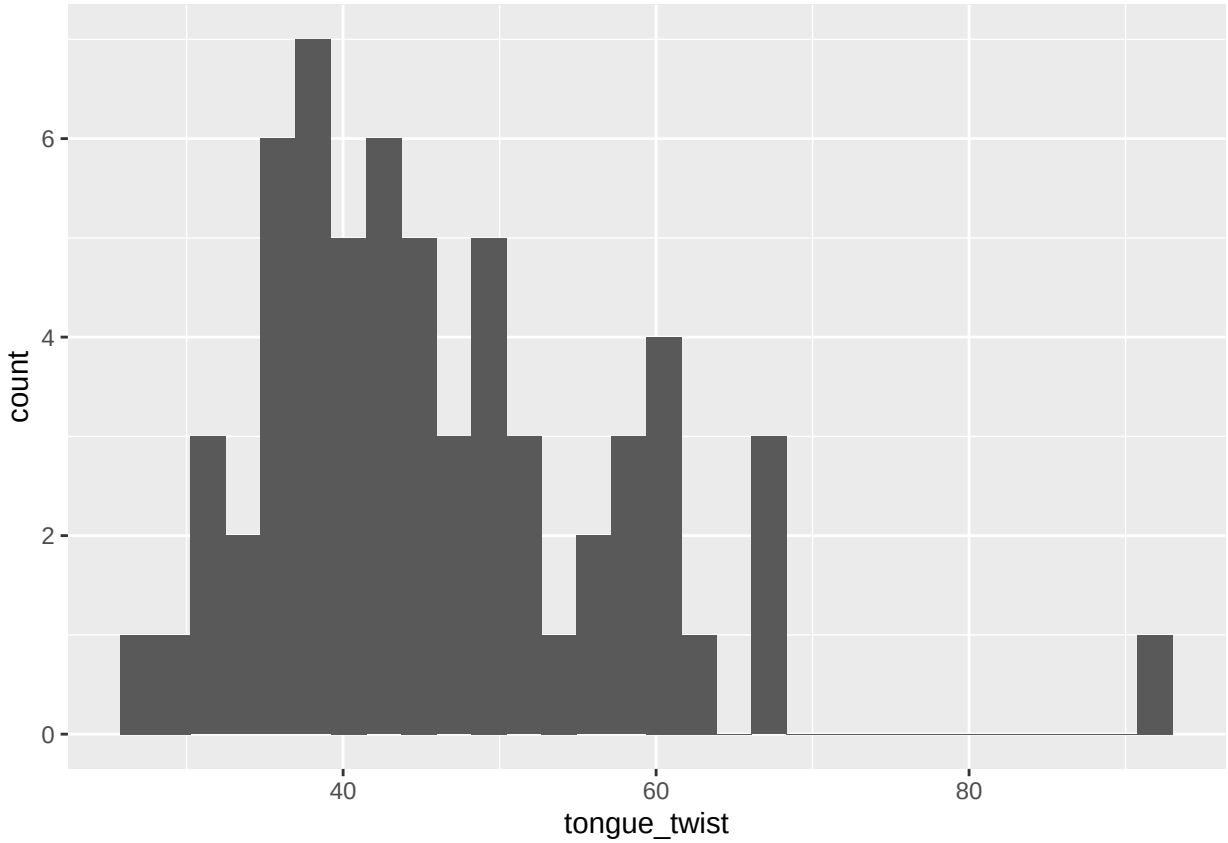
geom_histogram()

geom_histogram() shows distribution of values, i.e. how many times which value has appeared in data

Let's say we just want to see what is going on in the tongue_twist measure. We know that this measure is continuous. From the cheatsheet, we can see that one of our options for a continuous variable is using geom_histogram:

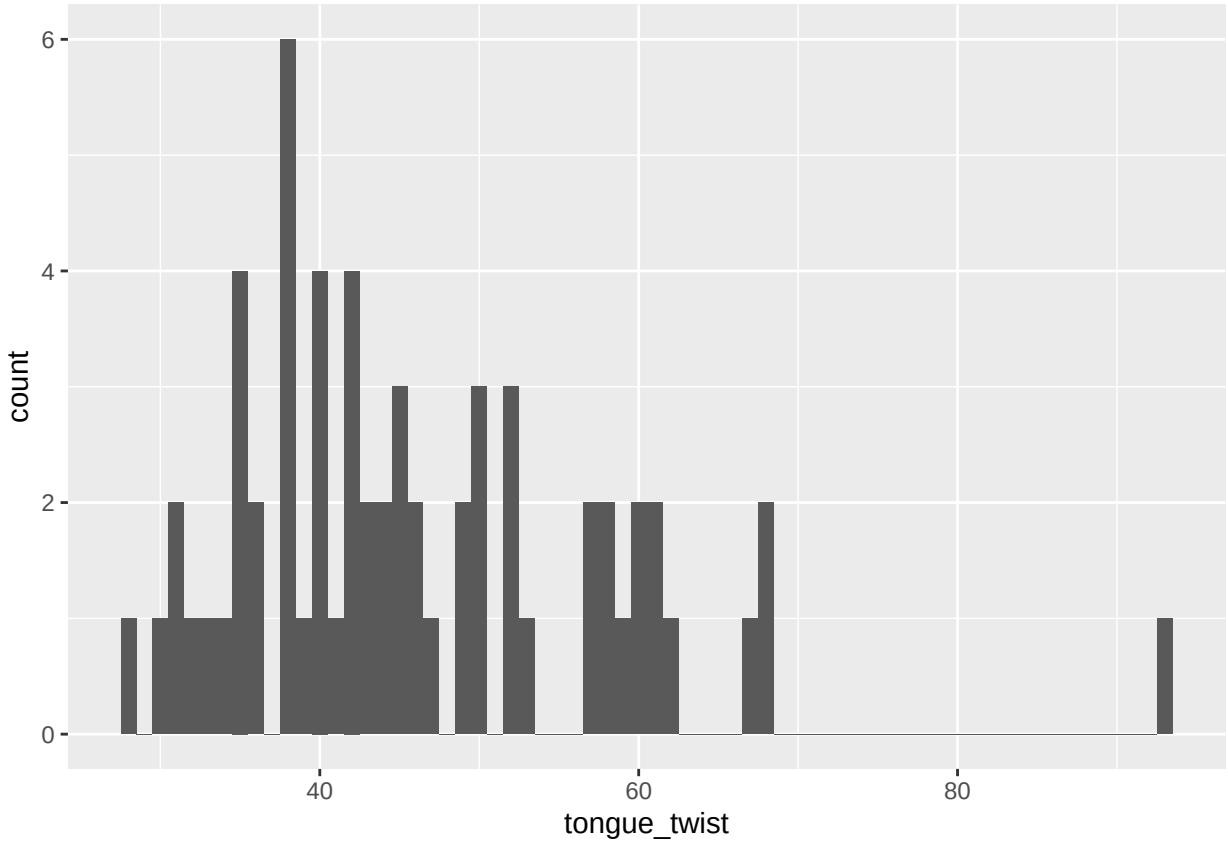
```
ggplot(df, aes(x=tongue_twist))+
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



We can kind of see already, what is going on, but let's make it prettier by changing one of parameters of the geom:

```
ggplot(df, aes(x=tongue_twist))+
  geom_histogram(binwidth = 1) #you can change binwidth if you want to, it's cosmetic
```



Geom_histrogram just showed counts for different values of tongue twister column - majority reported time around 40 seconds.

Bonus question: Is there something weird with tongue twister data?

Practice: Make a ggplot with geom_histogram to see distribution of choose_rand_num (Make a code chunk for it).

Question: Do you think it's normally distributed?

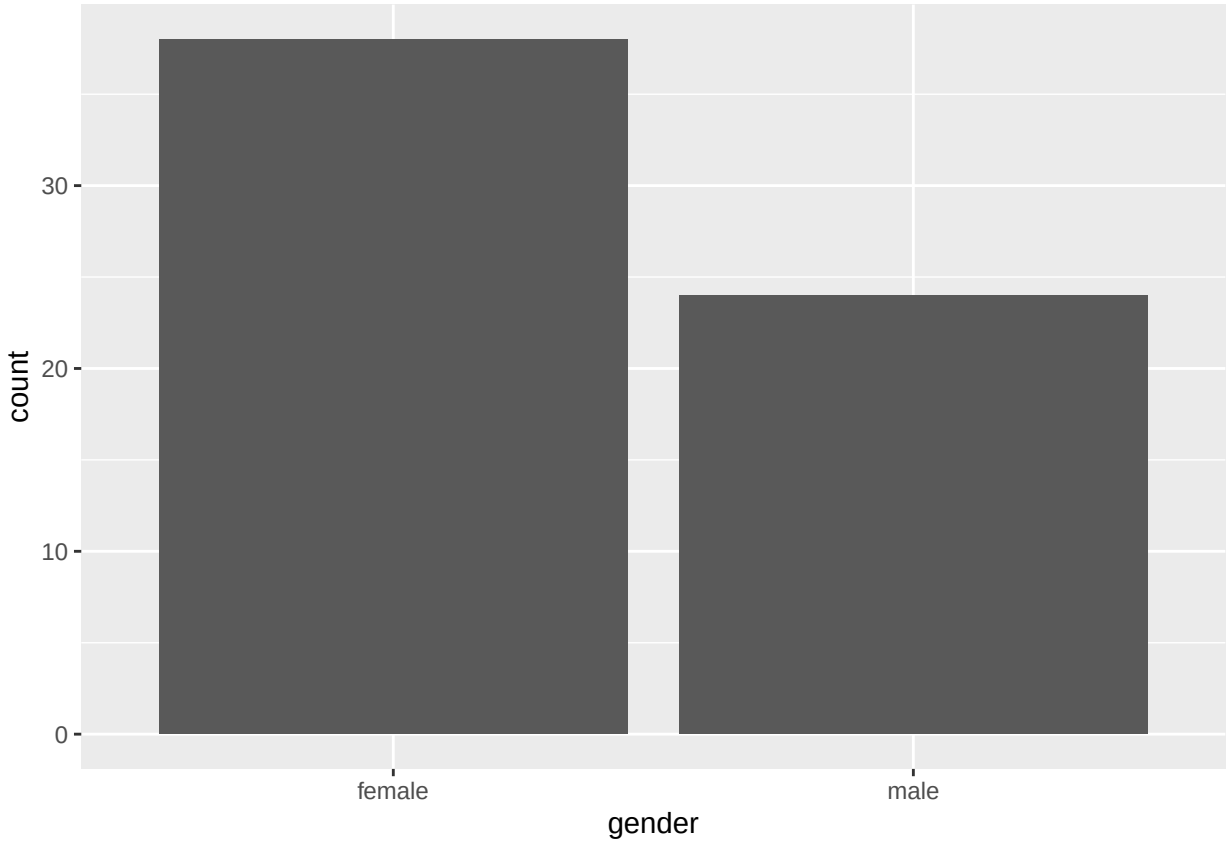
Next week we will learn how to check for sure!

geom_bar()

Bar plots are fun and pretty simple. Bar plots work especially well, when x coordinate is categorical If we specify only x coordinate - geom_bar will put counts on y coordinate.

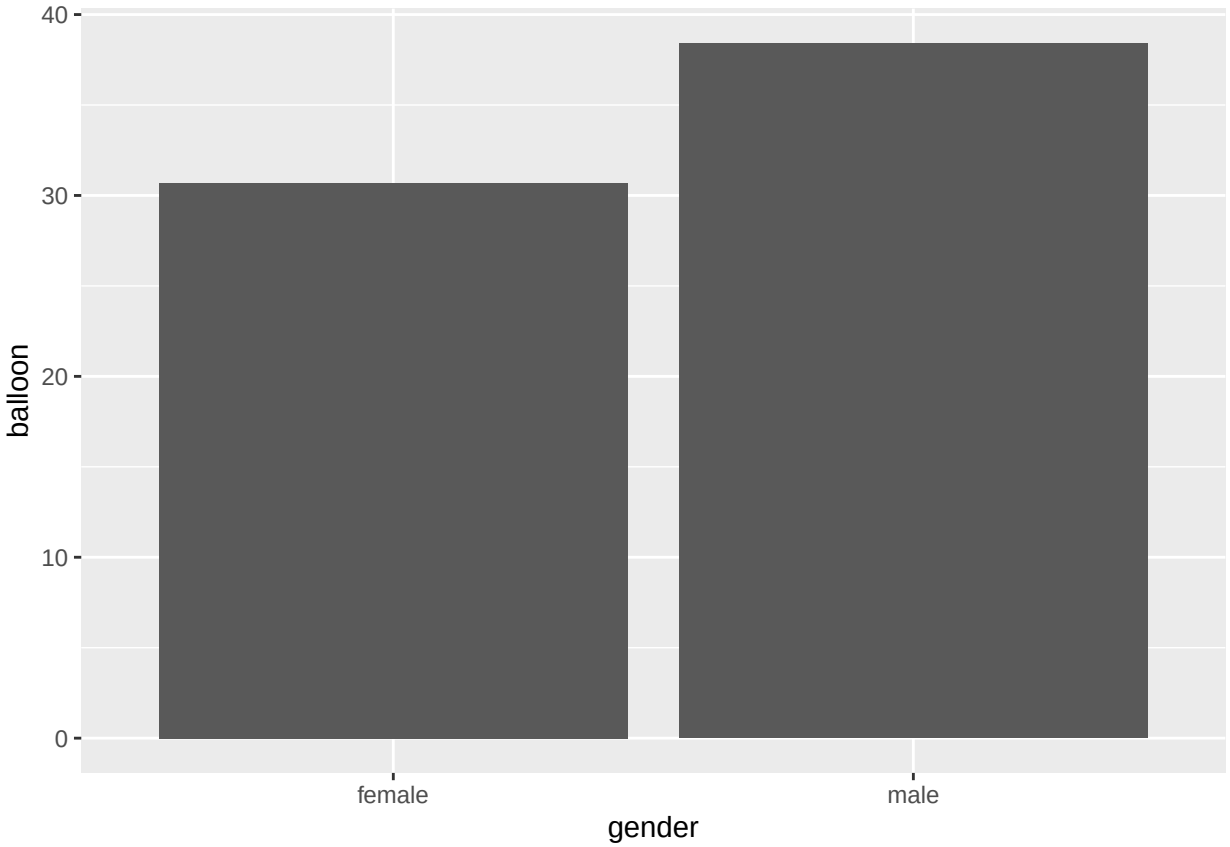
Let's try it out by setting x coordinate to show gender and plotting geom_bar:

```
ggplot(df, aes(x=gender))+
  geom_bar()
```



If we know y-coordinate too, then we need to help geom_bar() to decide what to do with data instead of counting:

```
ggplot(df, aes(x=gender, y=balloon))+
  geom_bar(stat='summary', fun.y = mean) #we ask to show us the mean of values on y coordinate
```

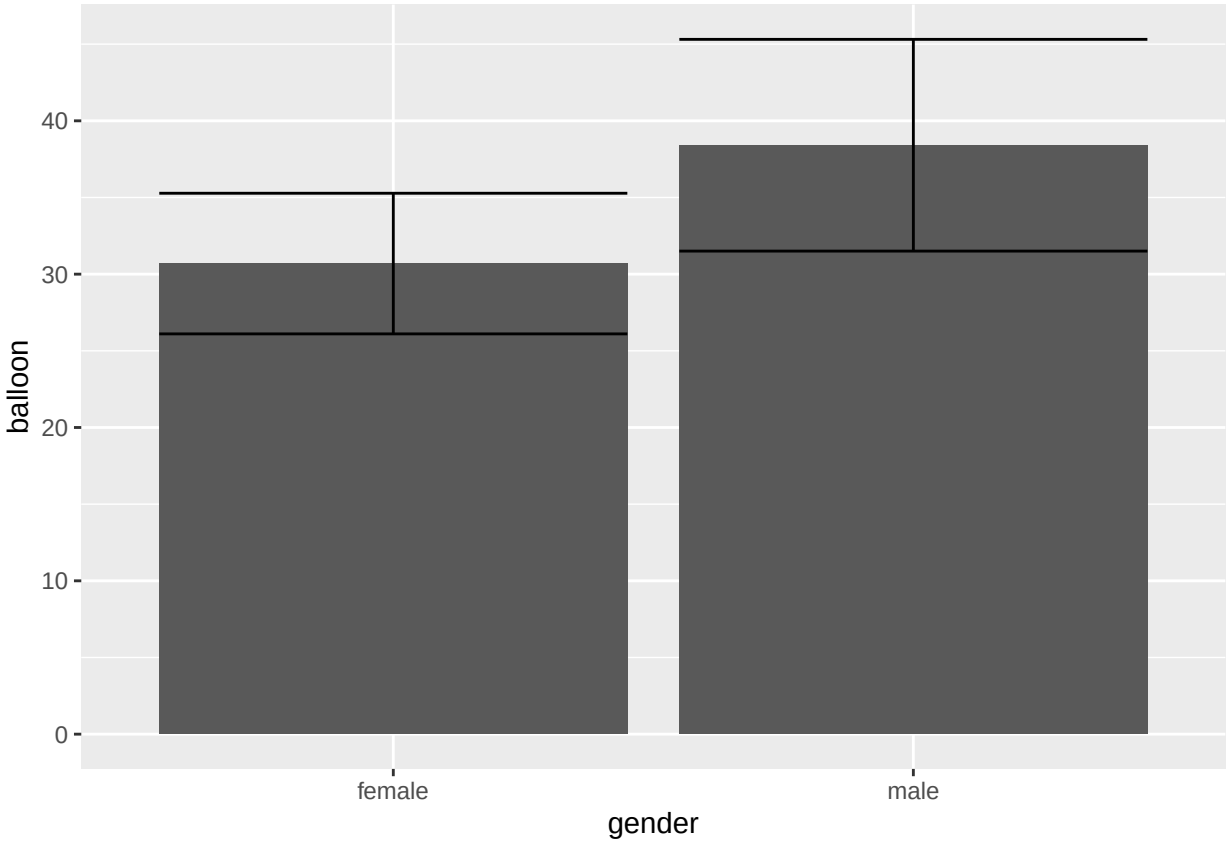


As you can see, geoms are pretty flexible! We can change functionality by changing stuff in parantheses - as you saw in geom_bar(stat='summary', fun.y = mean)

geom_errorbar()

Needs specification in the parantheses to claculate and plot standard error of the mean : geom_errorbar(stat = 'summary', fun.data = mean_se)

```
ggplot(df, aes(x=gender, y=balloon))+
  geom_bar(stat='summary', fun.y = mean)+
  geom_errorbar(stat = 'summary', fun.data = mean_se)
```



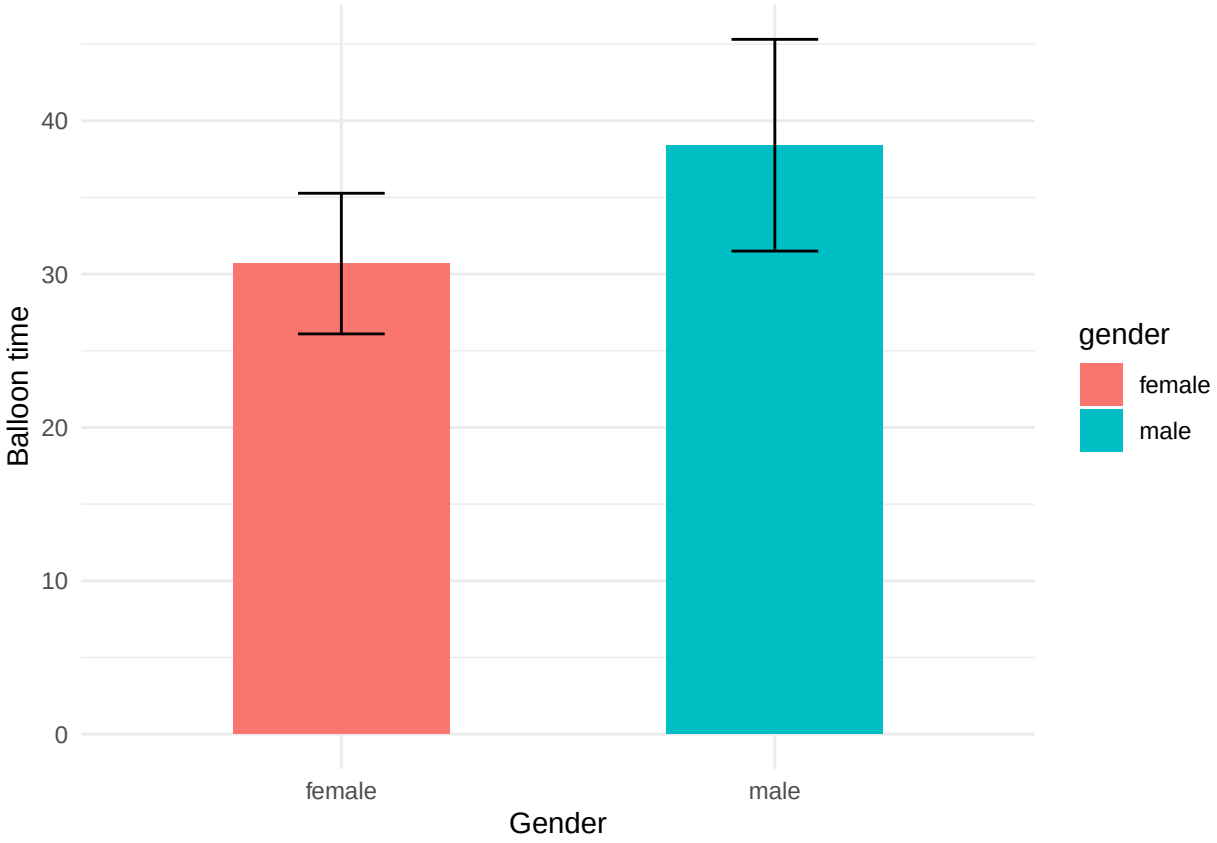
More layers and aesthetics

With ggplot, we can get more and more layers and settings - to make the plot look exactly like we want it.

I tried to make the previous plot pretty.

Try to understand the script below, what does each geom do? What happens when you change settings?

```
ggplot(df, aes(x=gender, y=balloon, fill = gender))+
  geom_bar(stat='summary', fun.y = mean, width = 0.5)+
  geom_errorbar(stat = 'summary', fun.data = mean_se, width = 0.2)+
  labs(x = "Gender", y = "Balloon time")+
  theme_minimal()
```



Visualization exercise

1. Identically to bar plots we've made above, make a bar plot showing average shoesize according to handedness.
2. Try to make it pretty!

Homework exercises

1. Make sure to finish Part 2 - before Tuesday Cognition and Communication class (just read it and solve the 3 questions in exercises, you'll feel better in the class)
2. Do the visualization exercise
3. Play around with Ggplot! Use the ggplot cheatsheet, change different parts of your code and see what changed

3.(Optional) Go through Chapter 3 in 'R for Data Science' book (http://r4ds.had.co.nz/index.html)