Descriptive Statistics

Yebelay Berehan

Biostatistician

yebelay.ma@gmail.com

2022-06-09

Descriptive statistics in R

- Introduction
- Descriptive statistics
  - Min,max,Range
  - Mean,Median, Mode,Quantiles, IQR
  - Sd and variance
  - Summary
  - Contingency table
- Advanced descriptive statistics
- Graphical procedures
  - Barplot, Boxplot, Pie chart
  - Histogram, Scatterplot, Line plot, QQ-plot, Density plot

Introduction

- How to compute the main descriptive statistics in R and how to present them graphically.
- descriptive statistics is a branch of statistics aiming at summarizing, describing and presenting a series of values or a dataset.
- Descriptive statistics is often the first step and an important part in any statistical analysis.
- It allows to check the quality of the data and it helps to “understand” the data by having a clear overview of it.
- If well presented, descriptive statistics is already a good starting point for further analyses. There exists many measures to summarize a dataset.

They are divided into two types:

- **location measures** and **dispersion measures**
- Location measures give an understanding about the central tendency of the data, whereas dispersion measures give an understanding about the spread of the data.
- We use the dataset iris and we need to load it by running iris:

```r
dat <- iris # load the iris dataset and renamed it dat
```

- Below a preview of this dataset and its structure:

```r
head(dat) # first 6 observations
str(dat) # structure of dataset
```

- The dataset contains 150 observations and 5 variables. Length and width of the sepal and petal are numeric variables and the species is a factor with 3 levels.

Minimum and maximum

- Minimum and maximum can be found thanks to the min() and max() functions:

```
min(dat$Sepal.Length)
max(dat$Sepal.Length)
```

- Alternatively the range() function:

```
rng <- range(dat$Sepal.Length)
```

- gives you the minimum and maximum directly.

```
rng[1] # rng = name of the object specified above
# and the maximum with:
rng[2]
```

Range

- The range can then be easily computed, as you have guessed, by subtracting the minimum from the maximum:

```
max(dat$Sepal.Length) - min(dat$Sepal.Length)
```

- We can create our own function to compute the range:

```
range2 <- function(x) {
  range <- max(x) - min(x)
  return(range)
}
range2(dat$Sepal.Length)
```

- which is equivalent than max−minmax−min presented above.

Mean

- The mean can be computed with the mean() function:

```r
mean(dat$Sepal.Length)
```

Tips: if there is missing value in your dataset, use `mean(dat$Sepal.Length, na.rm = TRUE)`.

Median

- The median can be computed thanks to the median() function:

```r
median(dat$Sepal.Length) # or with the quantile() function:
quantile(dat$Sepal.Length, 0.5)
```

- since the quantile of order 0.5 (q0.5) corresponds to the median.
- by setting the second argument to **0.25 or 0.75**, to compute first and third quartile.

Mode

- To my knowledge there is no function to find the mode of a variable.
- However, we can easily find it thanks to the functions `table()` and `sort()`:

```r
tab <- table(dat$Sepal.Length) # number of occurrences for each unique value
sort(tab, decreasing = TRUE) # sort highest to lowest
```

- `table()` gives the number of occurrences for each unique value, then `sort()` with the argument `decreasing = TRUE` displays the number of occurrences from highest to lowest.

- The mode of the variable Sepal.Length is thus 5.

- This code to find the mode can also be applied to qualitative variables such as Species:

```r
sort(table(dat$Species), decreasing = TRUE) # or:
summary(dat$Species)
```

Interquartile range

The interquartile range can be computed with the IQR() function:

```
IQR(dat$Sepal.Length)# or using quantile()
quantile(dat$Sepal.Length, 0.75) - quantile(dat$Sepal.Length, 0.25)
```

Standard deviation and variance

- The standard deviation and the variance is computed with the sd() and var() functions:

```
sd(dat$Sepal.Length) # standard deviation
var(dat$Sepal.Length) # variance
```

Tip: to compute the standard deviation of multiple variables at the same time, use `lapply()`

```
lapply(dat[, 1:4], sd)
```

Summary

- You can compute the minimum, 1st quartile, median, mean, 3rd quartile and the maximum for all numeric variables of a dataset at once using summary():

```
summary(dat, digits = 3)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
##  Min.   :4.30   Min.   :2.00   Min.   :1.00   Min.   :0.1   setosa    :50  
##  1st Qu.:5.10   1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3   versicolor:50  
##  Median :5.80   Median :3.00   Median :4.35   Median :1.3   virginica :50  
##  Mean   :5.84   Mean   :3.06   Mean   :3.76   Mean   :1.2                  
##  3rd Qu.:6.40   3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8                  
##  Max.   :7.90   Max.   :4.40   Max.   :6.90   Max.   :2.5                  
```

- Tip: if you need these descriptive statistics by group use the by() function:

```
by(dat, dat$Species, summary)
```

- where the arguments are the name of the dataset, the grouping variable and the summary function.
- If you need more descriptive statistics, use **stat.desc()** from the package `pastecs`:

```
library(pastecs)
stat.desc(dat)
```

- You can have even more statistics (i.e., skewness, kurtosis and normality test) by adding the argument `norm = TRUE` in the previous function.

- Note that the variable Species is not numeric, so descriptive statistics cannot be computed for this variable and NA are displayed.

Contingency table

- `table()` introduced above can also be used on two qualitative variables to create a contingency table.
- The dataset iris has only one qualitative variable so we create a new qualitative variable just for this example.
- We create the variable size which corresponds to small if the length of the petal is smaller than the median of all flowers, big otherwise:

```r
dat$size <- ifelse(dat$Sepal.Length < median(dat$Sepal.Length),
  "small", "big")
```

- Here is a recap of the occurrences by size:

```
table(dat$size)
```

- We now create a contingency table of the two variables Species and size with the `table()` function:

```
table(dat$Species, dat$size)
# or with the xtabs() function:
xtabs(~ dat$Species + dat$size)
```

- The contingency table gives the number of cases in each subgroup.

- Note that Species are in rows and size in column because we specified Species and then size in table().
- Change the order if you want to switch the two variables.
- Instead of having the frequencies you can also have the relative frequencies (i.e., proportions) in each subgroup by adding the `table()` function inside the `prop.table()` function:

```
prop.table(table(dat$Species, dat$size))
```

- Note that you can also compute the percentages by row or by column by adding a second argument to the `prop.table()` function: 1 for row, or 2 for column:

```
# percentages by row:
round(prop.table(table(dat$Species, dat$size), 1), 2)
round(prop.table(table(dat$Species, dat$size), 2), 2)
```

Advanced descriptive statistics

- There are also, many more functions and packages to perform more advanced descriptive statistics in R.

summarytools package

- One package for descriptive statistics I often use for my projects in R is the **summarytools()** package. The package is centered around 4 functions:

1. `freq()` for frequencies tables
2. `ctable()` for cross-tabulations
3. `descr()` for descriptive statistics
4. `dfSummary()` for dataframe summaries

- A combination of these 4 functions is usually more than enough for most descriptive analyses.

Frequency tables with freq()

- The `freq()` function produces frequency tables with frequencies, proportions, as well as missing data information.

```
library(summarytools)
freq(dat$Species)
```

- If you do not need information about missing values, add the `report.nas = FALSE` argument:

```
freq(dat$Species,
  report.nas = FALSE) # remove NA information
```

And for a minimalist output with only counts and proportions:

```
freq(dat$Species,
  report.nas = FALSE, # remove NA information
  totals = FALSE, # remove totals
  cumul = FALSE, # remove cumuls
  headings = FALSE) # remove headings
```

Cross-tabulations with ctable()

- The `ctable()` function produces cross-tabulations for pairs of categorical variables. Using the two categorical variables in our dataset:

```
ctable(x = dat$Species, y = dat$size)
```

- Row proportions are shown by default. To display column or total proportions, add the prop = "c" or prop = "t" arguments, respectively:

```r
ctable( x = dat$Species,  y = dat$size,
  prop = "t" ) # total proportions
```

- To remove proportions altogether, add the argument prop = "n".

- Furthermore, to display only the bare minimum, add the totals = FALSE and headings = FALSE arguments:

```r
ctable(x = dat$Species, y = dat$size,
  prop = "n", # remove proportions
  totals = FALSE, # remove totals
  headings = FALSE) # remove headings
```

- This is equivalent than `table(dat$Species, dat$size)` and `xtabs(~ dat$Species + dat$size)` performed in the section on contingency tables.

- To display results of the Chi-square test of independence, add the chisq = TRUE argument:3

```
ctable(x = dat$Species, y = dat$size,
  chisq = TRUE, # display results of Chi-square test of independence
  headings = FALSE) # remove headings
```

- The p-value is close to 0 so we reject the null hypothesis of independence between the two variables.
- This indicates that species and size are dependent and that there is a significant relationship between the two variables.
- It is also possible to create a contingency table for each level of a third categorical variable thanks to the combination of the `stby()` and `ctable()` functions.

- There are only 2 categorical variables in our dataset, so let’s use the tabacco dataset which has 4 categorical variables (i.e., gender, age group, smoker, diseased).

- For this example, we would like to create a contingency table of the variables smoker and diseased, and this for each gender:

```
stby(list(  x = tobacco$smoker, # smoker and diseased
  y = tobacco$diseased),
INDICES = tobacco$gender, # for each gender
FUN = ctable # ctable for cross-tabulation
)
```

Descriptive statistics with `descr()`

- The `descr()` function produces descriptive statistics with common central tendency and measures of dispersion.
- A major advantage of this function is that it accepts single vectors as well as data frames.
- If a data frame is provided, all non-numerical columns are ignored so you do not have to remove them yourself before running the function.
- The `descr()` function allows to display:
    - only a selection of some descriptive statistics, use `stats = c("mean", "sd")` argument.
    - the min, 1st quartile, median, 3rd quartile and max, use `stats = "fivenum"`
    - the most common descriptive statistics (mean, sd, min, median, max, valid observations), with `stats = "common"`:

```r
descr(dat,headings = FALSE, # remove headings
  stats = "common" # most common descriptive statistics
)
```

```
## Non-numerical variable(s) ignored: Species, size
```

Tip: if you have a large number of variables, add the`transpose = TRUE` argument for a better display.

- In order to compute these descriptive statistics by group (e.g., Species in our dataset), use the `descr()` function in combination with the `stby()` function:

```r
stby( data = dat, INDICES = dat$Species, # by Species
  FUN = descr, # descriptive statistics
  stats = "common") # most common descr. stats
```

```
## Non-numerical variable(s) ignored: Species, size
```

Data frame summaries with `dfSummary()`

- The `dfSummary()` function generates a summary table with statistics, frequencies and graphs for all variables in a dataset.
- The information shown depends on the type of the variables and also varies according to the number of distinct values.

```
dfSummary(dat)
```

`describeBy()` from the psych() package

- The `describeBy()` function from the `psych()` package allows to report several summary statistics
  - (i.e., total valid cases, mean, sd, min, max, range, skewness and kurtosis) by a grouping variable.

```r
library(psych)
describeBy( dat,dat$Species) # grouping variable
```

`aggregate()` function

- The `aggregate()` function allows to split the data into subsets and then to compute summary statistics for each.
- For instance, if we want to compute the mean for the variables Sepal.Length and Sepal.Width by Species and Size:

```r
aggregate(cbind(Sepal.Length, Sepal.Width) ~ Species + size,
  data = dat, mean)
```

summaryBy() from doBy() package

- An alternative is the `summaryBy()` function from the `doBy()` package:

```
# summary statistics by group
library(doBy)
summaryBy(Sepal.Length + Sepal.Width ~ Species,
  data = dat,  FUN = summary)
```

- If you are interested in some specific descriptive statistics, you can easily specify them via the FUN argument:

```
summaryBy(Sepal.Length + Sepal.Width ~ Species,
  data = dat,  FUN = c(mean, var))
```

group_by() and summarise() from {dplyr}

- Another alternative is with the `summarise()` and `group_by()` functions from the

dplyr() package:

```
library(dplyr)
group_by(dat, Species) %>%
  summarise(mean = mean(Sepal.Length, na.rm = TRUE),
    sd = sd(Sepal.Length, na.rm = TRUE))
```

Graphical Procedure

OVERVIEW

- Data visualization is part art and part science.

Graphics in base R

- Class sensitive commands
- Everything can be adjusted through parameters.
- But sometimes hard to find the right ones.
- Require lots of trial-error, no easy patterns.
- Memorizing lots of plotting commands and options.
- Hard to manage, save, update, reproduce.

Why ggplot2?

- A grammar of graphics is a grammar used to describe and create a wide range of statistical graphics.
- The promise of a **grammar for graphics**.
- Easy to manage, save, etc.
- Graphs are composed of layers.
- Easy to add stuff to existing graphs.
- ggplot2 graphics take less work to make beautiful and eye-catching graphics.
- Enables creation of reproducible visualization patterns.
- Publication quality & beyond

Components of the layered grammar

- Layer
    - Data
    - Mapping
    - Statistical transformation (stat)
    - Geometric object (geom)
    - Position adjustment (position)
- Scale
- Coordinate system (coord)
- Faceting (facet)

Layers

- Layers are used to create the objects on a plot.
- A layer is composed of four parts:

1. data and aesthetic mapping
2. geometric object (geom)
3. statistical transformation (stat)
4. position adjustment

- Layers are typically related to one another and share many common features.
- example: a scatterplot overlayed with a smoothed regression line.

data and aesthetic mapping

Data

- Data defines the source of the information to be visualized.
- Must be a data.frame
- Gets pulled into the ggplot() object

Aesthetics (a.k.a. mapping)

- Defines how the variables are applied to the plot.

- They are properties of graphical elements, such as:

    - x, y coordinates of points
    - line type, size, shape,
    - colour, fill colour of points, lines, arrows, bars, boxes, and so forth.
    - alpha level of transparency.

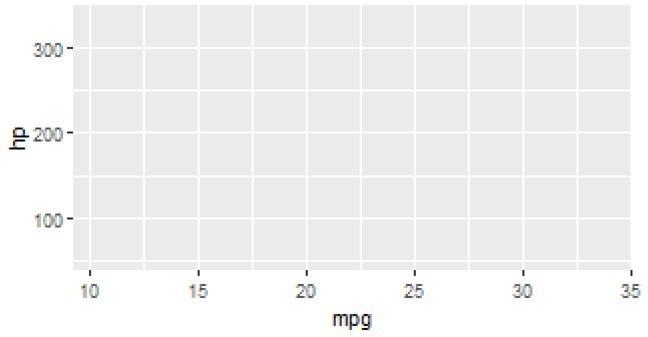
1. data and aesthetic mapping

- `ggplot(data = DATA, mapping = aes(MAPPINGS))+ GEOM_FUNCTION()`

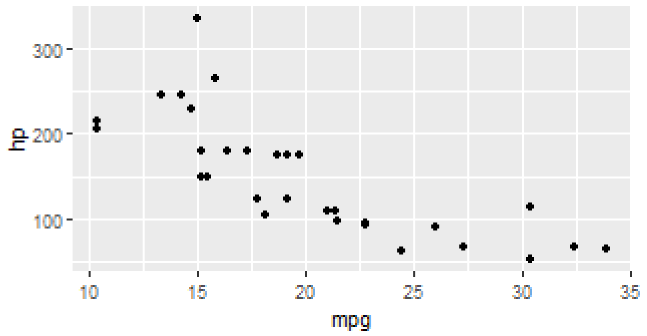
```
ggplot(dataset, aes(x=xvar, y=yvar)) + geom_function()
```

- **geom_function()** to produce shapes for the graph like: `geom_point()`, `geom_boxplot()`, ...
- Inside **aes()**, we can map more variables like: color, size and shape of plotted objects.

```
ggplot(mtcars,aes(x=mpg, y=hp))
```



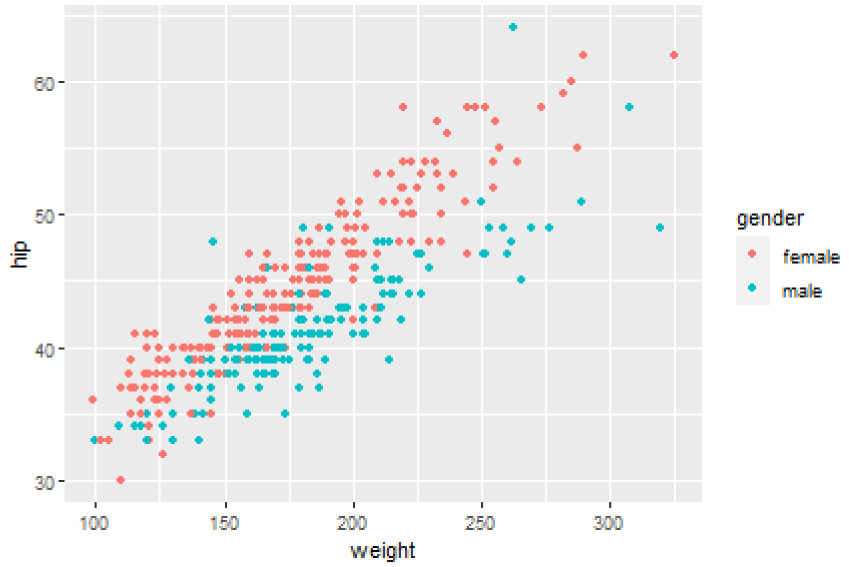
```
ggplot(mtcars, aes(x=mpg,y =hp)) +
geom_point()
```



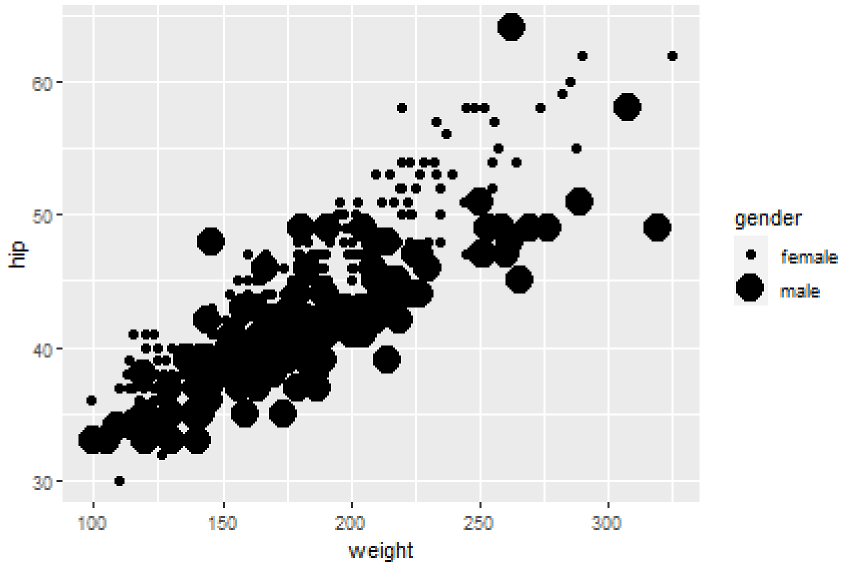
Aesthetics: `aes()`

- `ggplot(data = DATA, mapping = aes(MAPPINGS))+ GEOM_FUNCTION()`
- Let us look what do `color`, `size`, `alpha` and `shape` in ggplot?

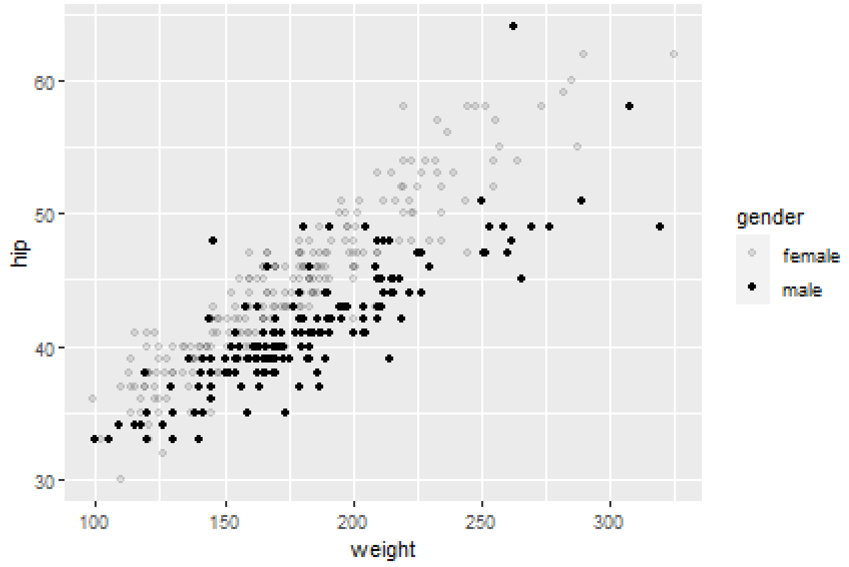
```
ggplot(data = diabetes,mapping
   = aes(x = weight, y = hip,
         color=gender))+geom_point()
```



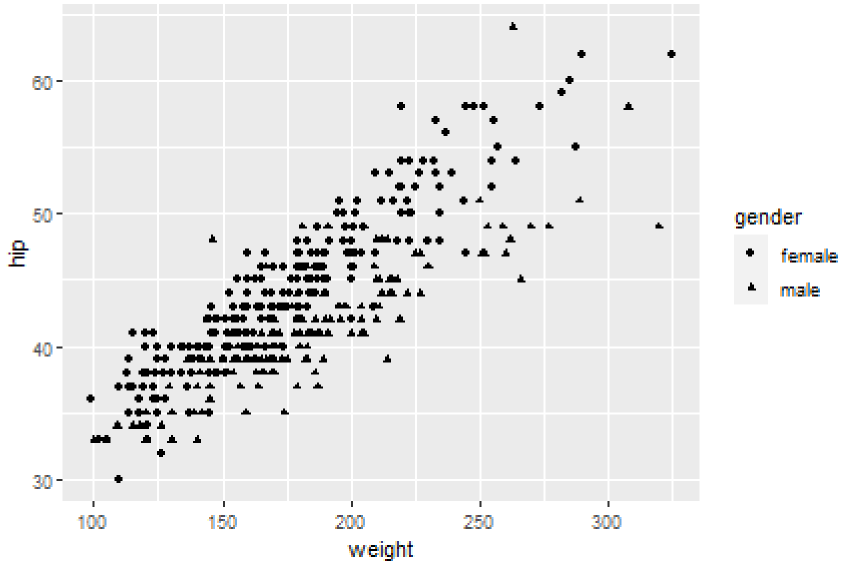
```
ggplot(data = diabetes,mapping =
aes(x=weight,y=hip,size=gender))+
  geom_point()
```



```r
ggplot(data = diabetes,mapping =
aes(x=weight,y=hip,alpha=gender))+
  geom_point()
```



```r
ggplot(data = diabetes, mapping =
aes(x=weight,y=hip,shape=gender))+
  geom_point()
```



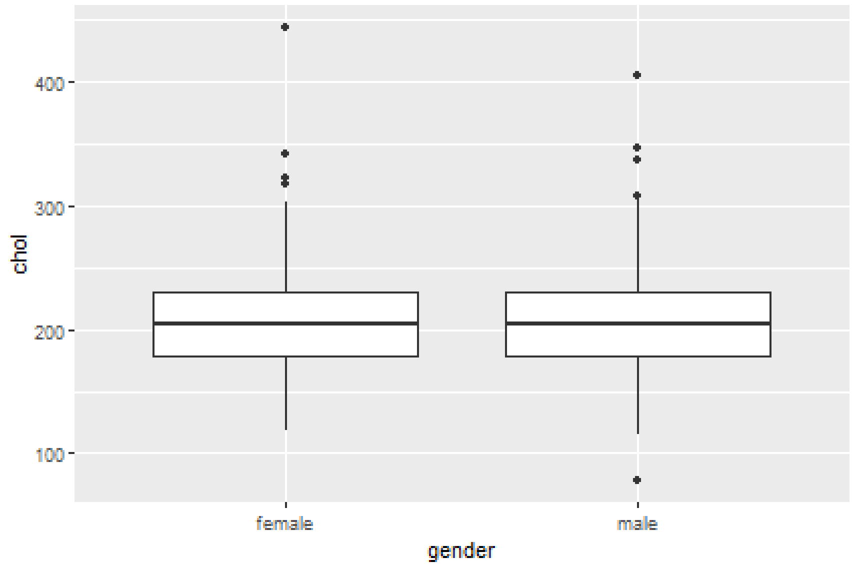
geoms

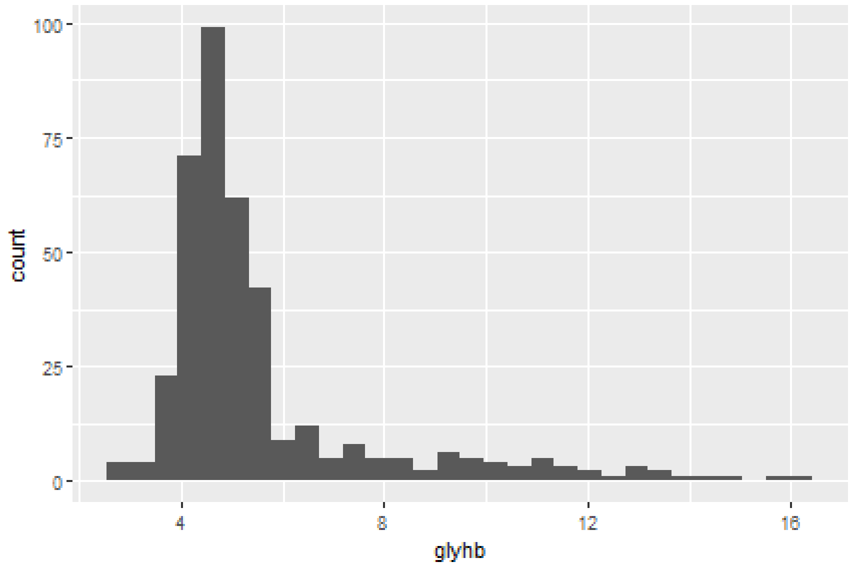
What shape does the data take?

- `geom_point()`, `geom_line()`, `geom_violin()`

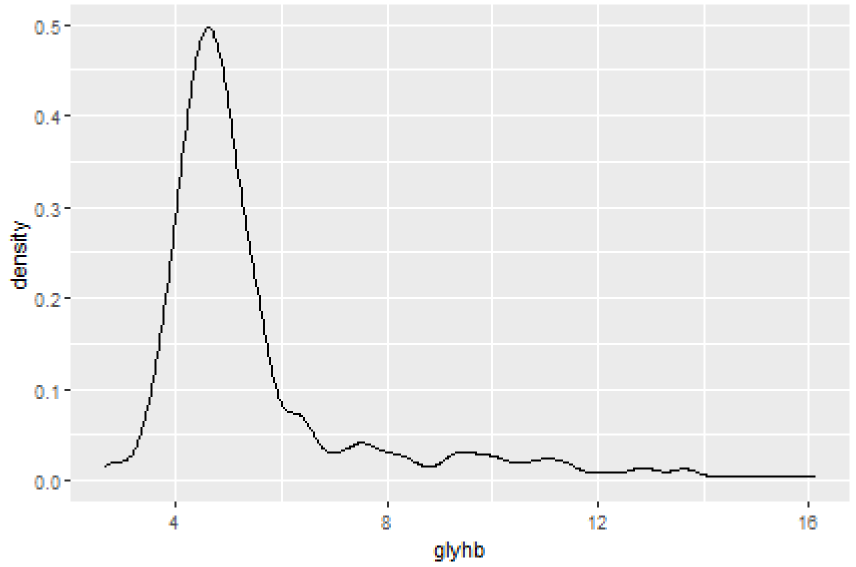
```
ggplot(diabetes,aes(gender,chol))+
  geom_boxplot()
```

```
ggplot(diabetes, aes(x = glyhb)) +
  geom_histogram()
```

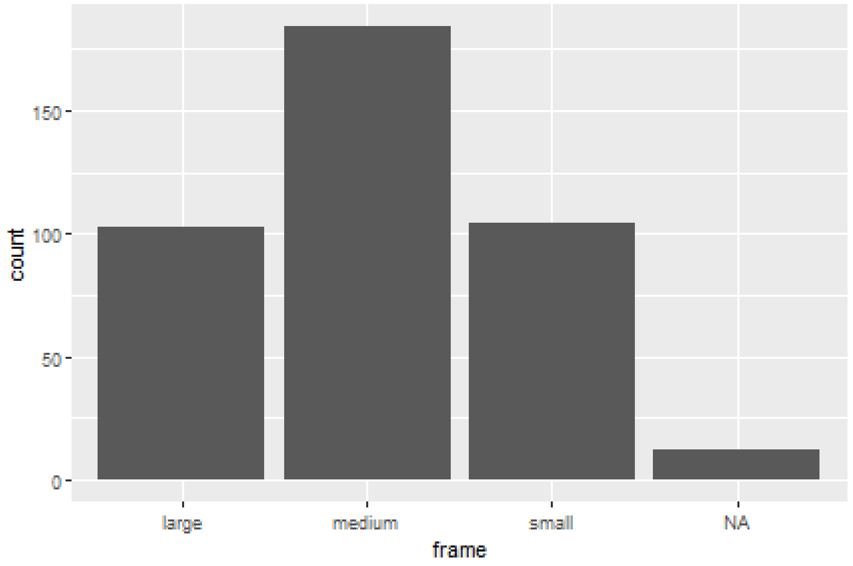




```r
ggplot(diabetes, aes(x = glyhb)) +
  geom_density()
```

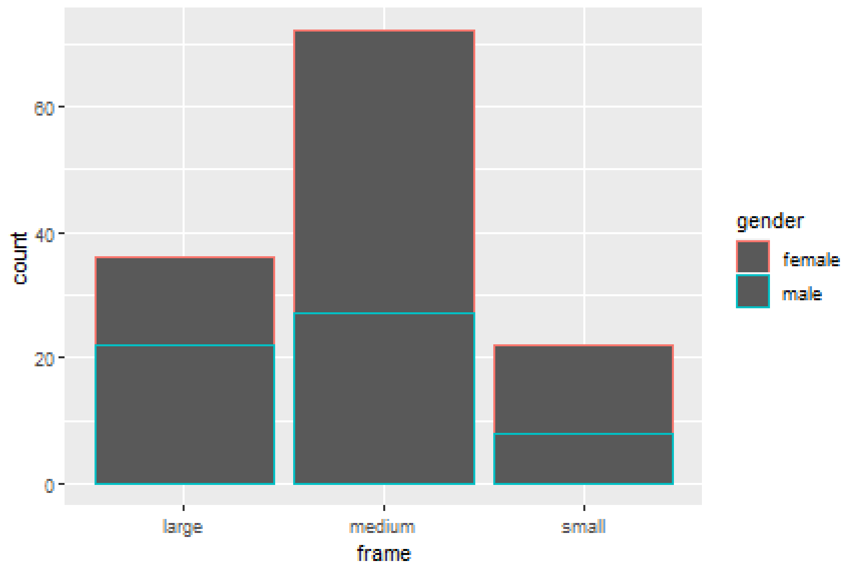


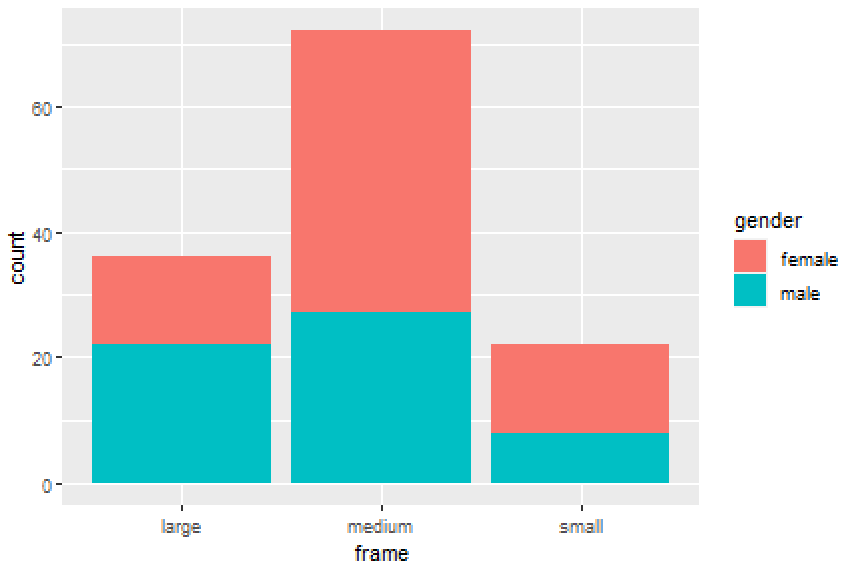
```r
diabetes %>%
  ggplot(aes(x = frame)) +
    geom_bar()
```



```
diabetes %>%
  drop_na() %>%
  ggplot(aes(x=frame,
color=gender))+geom_bar()
```

```
diabetes %>%
  drop_na() %>%
  ggplot(aes(x = frame,
  fill=gender))+geom_bar()
```



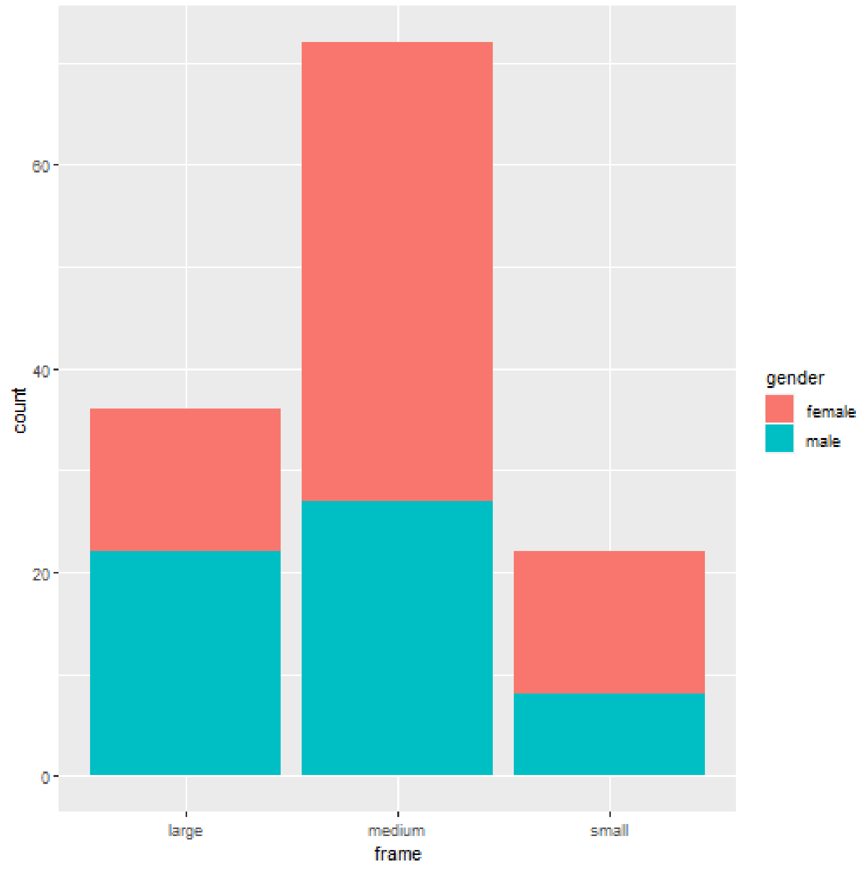


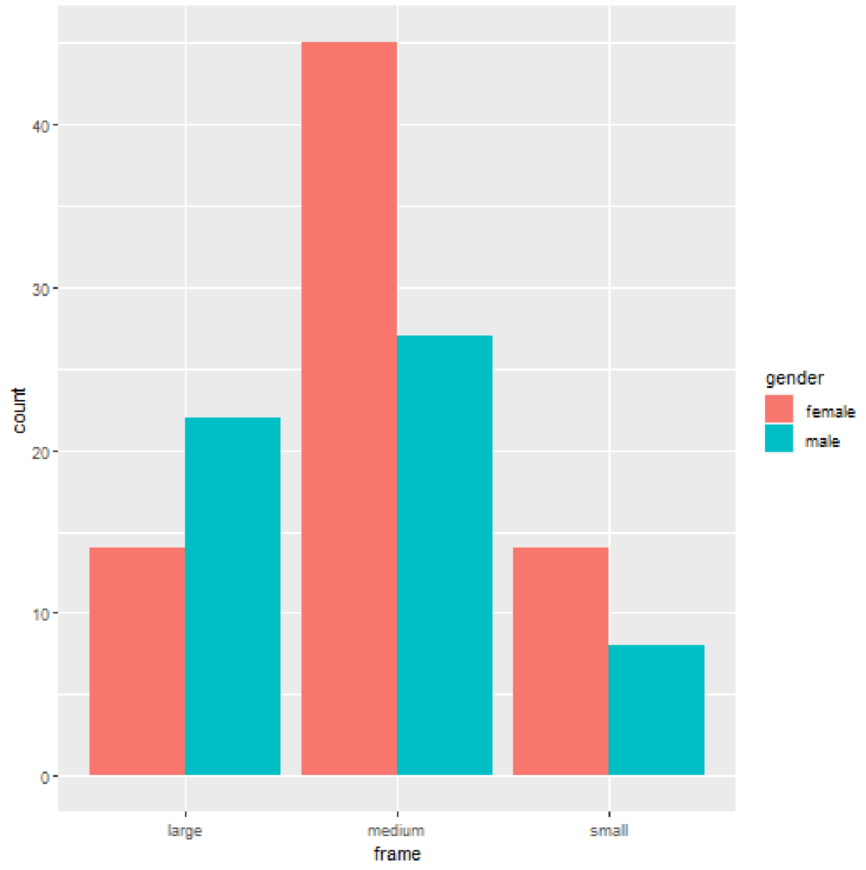
Positions

- `geom_bar(position = "<POSITION>")`
- When we have aesthetics mapped, how are they positioned?
- bar: dodge, fill, stacked (default)
- point: jitter

```
diabetes %>%
  drop_na() %>%
  ggplot(aes(x =frame,
  fill=gender))+
geom_bar(position ="stack")
```

```
diabetes %>%
  drop_na() %>%
  ggplot(aes(x = frame,
fill = gender)) +
  geom_bar(position = "dodge")
```





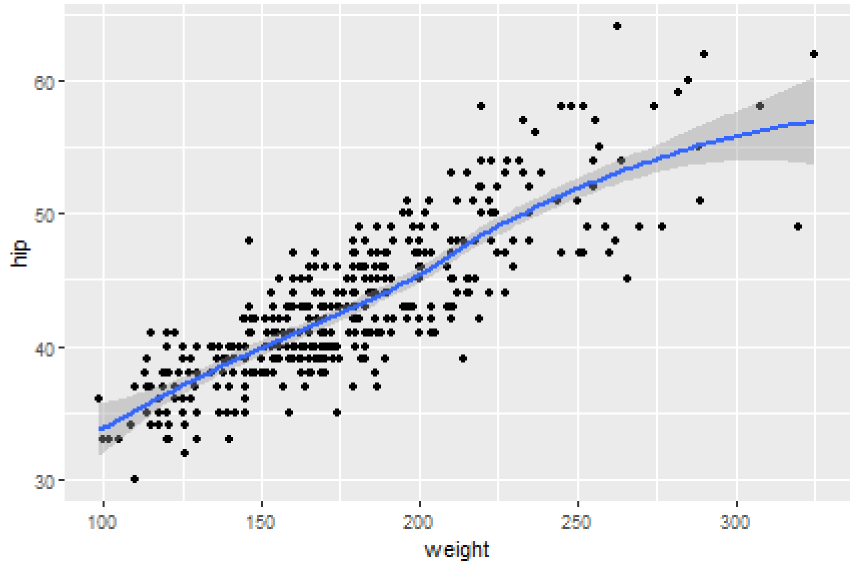
Live Code Part 1

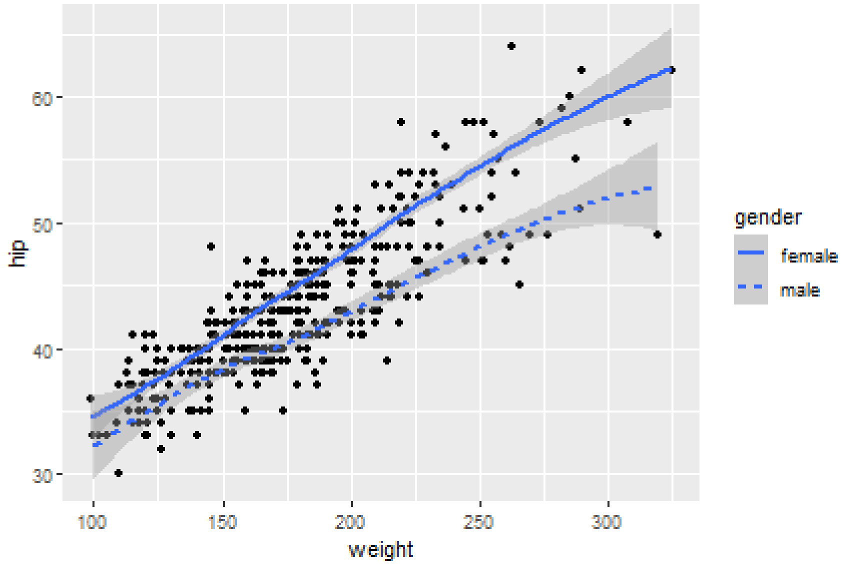
1. Predict what this code will do. Then run it.
2. Add a `linetype` aesthetic for `gender`. Run it again.
3. Set the color of `geom_smooth()` to "black"
4. Add `se = FALSE` to the `geom_smooth()`
5. It's hard to see the lines well now. How about setting `alpha = .2` in `geom_point()`?
6. Jitter the points. You can either change the geom or change the `position` argument.
7. Add another layer, `theme_bw()`. Remember to use `+`.

```
ggplot(diabetes, aes(weight, hip)) +
  geom_point() +
  geom_smooth()
```

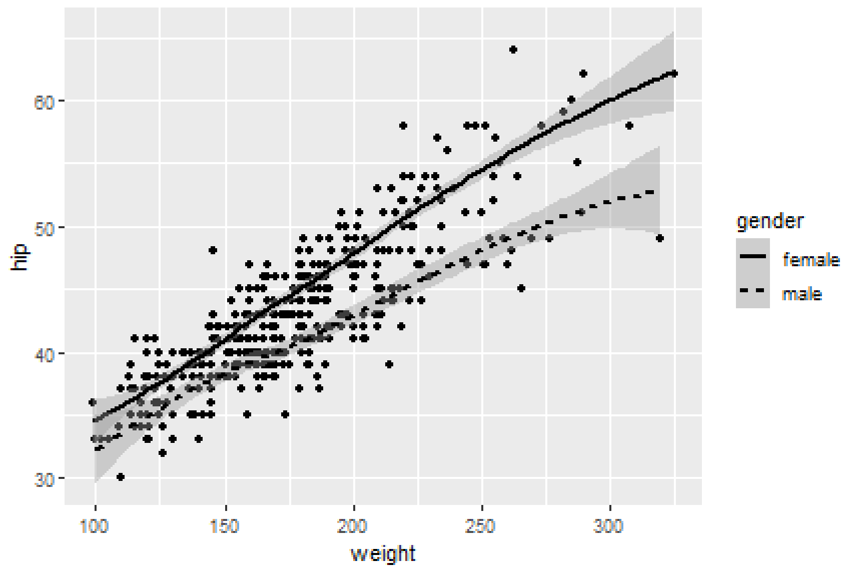
```r
ggplot(diabetes, aes(weight, hip)) +
  geom_point() +
  geom_smooth()
```

```r
ggplot(diabetes, aes(weight, hip)) +
  geom_point() +
  geom_smooth(aes(linetype = gender))
```

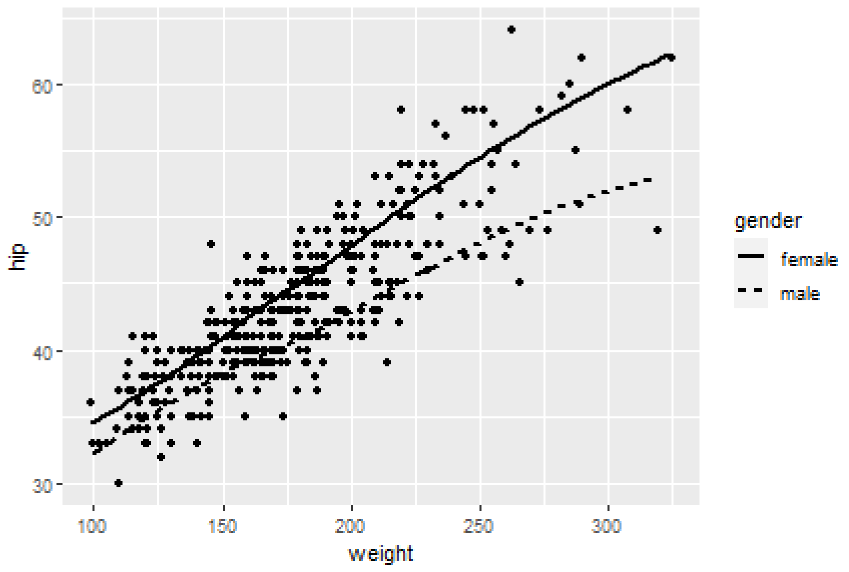




```r
ggplot(diabetes, aes(weight, hip)) +
  geom_point() +geom_smooth(
aes(linetype = gender),
col = "black")
```

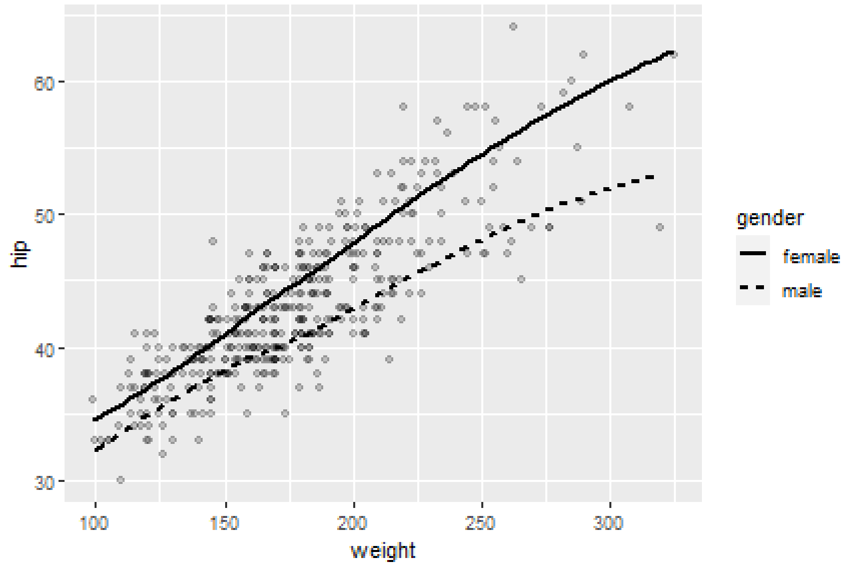


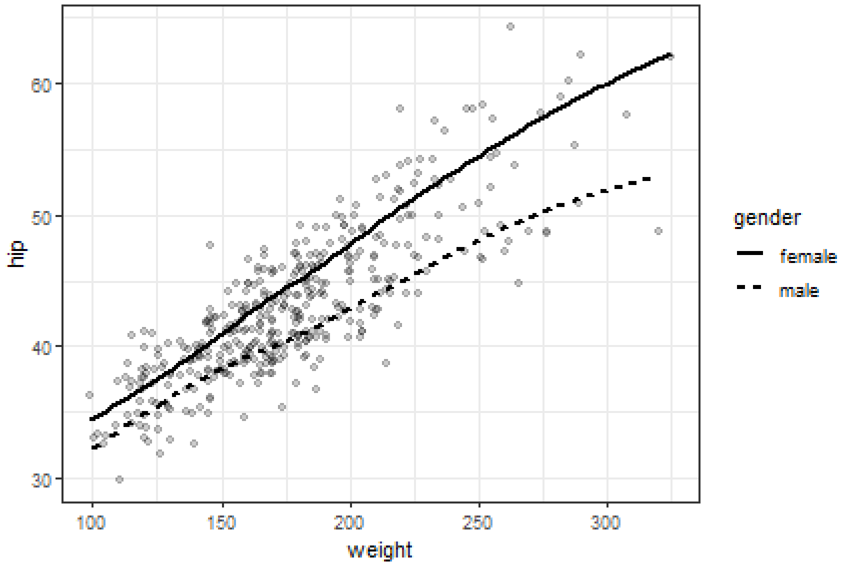
```r
ggplot(diabetes, aes(weight, hip)) +
  geom_point()+geom_smooth(
aes(linetype = gender),
col = "black", se = FALSE)
```



```
ggplot(diabetes, aes(weight, hip)) +
  geom_point(alpha = .2) +
  geom_smooth(
  aes(linetype = gender),
  col = "black", se = FALSE)
```

```
ggplot(diabetes, aes(weight, hip)) +
  geom_jitter(alpha = .2) +
  geom_smooth(aes(
    linetype = gender),
    col = "black", se = FALSE)+ theme
```



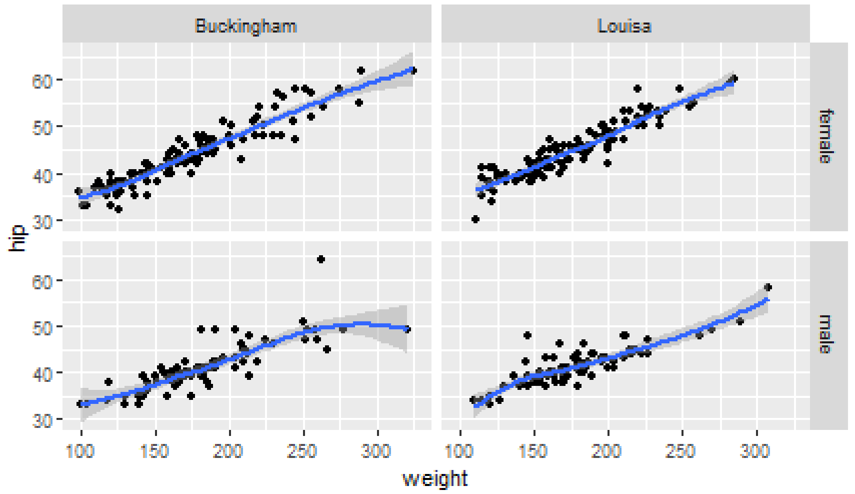


Facets

- Easy peazy panels
- `facet_grid()`
- `facet_wrap()`
- `x ~ y` or `~ y` or `x ~ .`

facet grid by `gender` and `location`

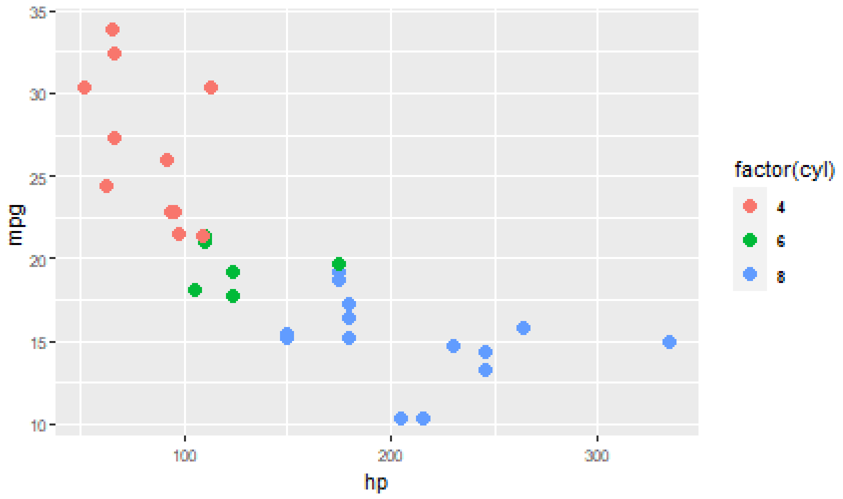
```
ggplot(diabetes, aes(weight, hip)) +
  geom_point() +
  geom_smooth() +
  facet_grid(
    gender ~ location)
```



Themes

- Non-data ink (text, background, etc)
- Prespecified themes: `theme_gray()` (default), `theme_minimal()`, `theme_light()`, etc.
- `theme()`

```
mtcars %>%
ggplot(aes(hp,mpg,
   col=factor(cyl)))+
geom_point(size=3) +
theme(axis.text=
element_text(size = 8),
legend.text=element_text(
  size=8,face="bold"),
  legend.direction = "vertical")
```



Labels, titles, and legends

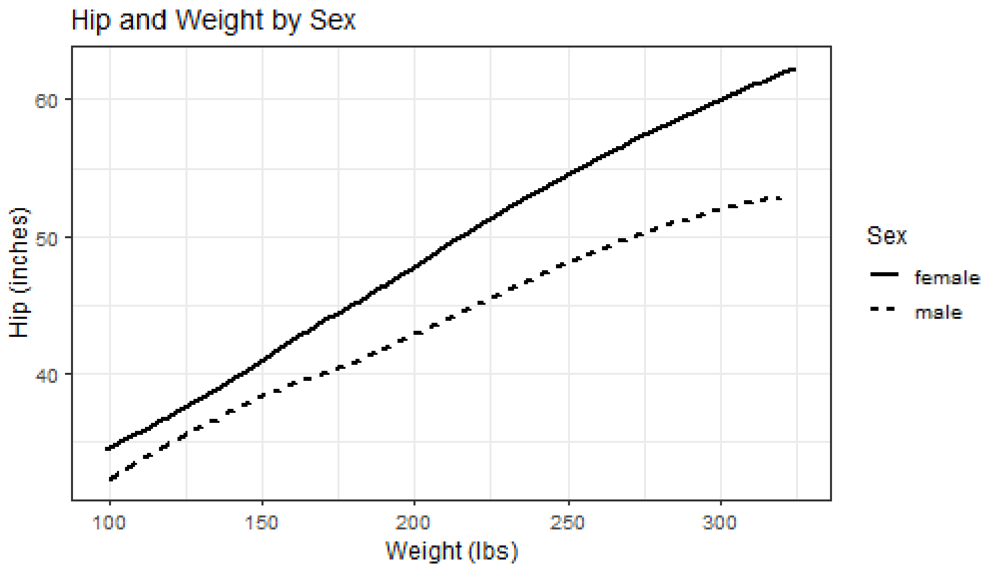
Add a title:

- `ggtitle()`, `labs(title = "My Awesome Plot")`

Change a label:

- `xlab()`, `ylab()`, `labs(x = "X!", y = "Y!!")`

```r
diabetes_plot <- ggplot(diabetes, aes(weight, hip, linetype = gender)) +
  geom_smooth(color = "black", se = FALSE) +
  theme_bw(base_size = 12) +
  labs(x = "Weight (lbs)", y = "Hip (inches)") +
  ggtitle("Hip and Weight by Sex") +
  scale_linetype(name = "Sex")
diabetes_plot
```


Hip and Weight by Sex
Hip (inches)
Weight (lbs)
Sex
female
male