DATA 227

Uncertainty in Data Visualizations

2022-11-17

Deadlines

Here is a list of tentative due dates for the rest of the quarter:

- HW5: Due Friday, November 18, at 6:00pm
- Final Project Drafts (optional): Tuesday, November 29. This is relatively early so that we can make sure to get feedback to you soon enough to use it. If you would prefer, you are welcome to stop by office hours and get feedback informally.
- Final Project: Due Thursday, December 8, at 2:30pm (the end of our scheduled exam period).

R vs. Python

- Choosing to use R or Python depends greatly on the task you are trying to accomplish. Many describe it as a question of specification vs. generalization.
  - "Python is generally used when the data analysis tasks need to be integrated with web apps or if statistics code needs to be incorporated into a production database. Since it's a full-fledged programming language, Python is a good tool to implement algorithms for use in production."
  - "R is mainly used when the data analysis tasks require standalone computing or analysis on individual servers. For exploratory work, R is easier for beginners. Statistical models can be written with a few lines of code."

DataCamp: Choosing Python or R for Data Analysis? An Infographic

Advantages

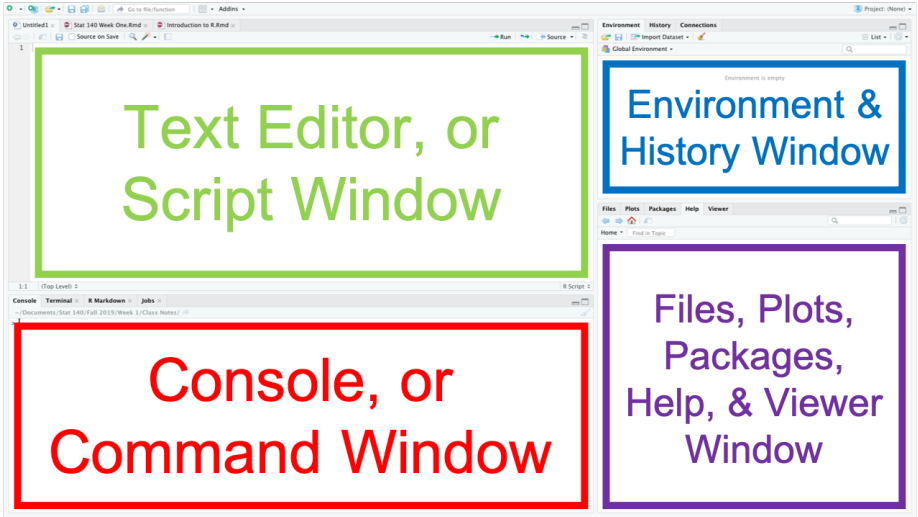
Python

- General-purpose programming languages are useful beyond just data analysis.
- Has gained popularity for its code readability, speed, and many functionalities.
- Great for mathematical computation and learning how algorithms work.
- Has high ease of deployment and reproducibility.

R

- Widely considered the best tool for making beautiful graphs and visualizations.
- Has many functionalities for data analysis.
- Great for statistical analysis.
- Built around a command line, but the majority of R users work inside of RStudio, an environment that includes a data editor, debugging support, and a window to hold graphics as well.

Introduction to RStudio



Console and Text Editor

- You can type directly in the console, and R will evaluate whatever it is that you type in the console.
- However, we often want to save a document and come back to it later.
  - To save an R document, first open a new file–it will appear in the text editor.
  - Whatever you type here won't run unless you highlight it and click the 'Run' button at the top left corner of the text editor window.
  - You can save whatever code you write and revisit it if needed.

Environment, History, Connections, and Tutorials

- I mostly use this window for the Environment tab–it's the thing I miss most about R when coding in Python.
- Any objects you save in R will show up in the environment and history window.
  - Data
  - Values
  - Imported functions sorted by packages
- There's also an option for showing current memory usage, which is useful when importing massive datasets.

Files, Plots, Packages, Help, and Viewer

- Files: Can be used to import data/
- Plots: When we create plots, they're confined to this window.
- Packages: We can use this window to see what packages are installed and loaded.
- Help: To access the help menu, type a question mark (?) before a command in the console.
- Viewer: Shows the output of Quarto or .Rmd files.

Introduction to Storage

- It is often useful to assign a value to a variable to save and use for later. In R, this can be done in multiple ways.
  - One way (which I use most frequently) is an arrow made of the left caret and the dash (<-).
  - You can also use the = sign, as you would in Python.
  - The symbol or phrase to which you are assigning the value is written on the left-hand side of the equation, and the value you wish to assign is written on the right.
  - For instance, if I wanted to assign a value of three to a, I would write:

```
a <- 3
```

Data Types and Storage

- Everything is an object in R, and all objects have a type.
  - character (defined by `''` or `""`)
  - numeric (real or decimal)
  - integer
  - logical
- Elements of these data types may be combined to form data structures, including:
  - atomic vector (*atomic* means that the vector contains a single data type)
  - list
  - matrix
  - data frame
  - factors

Vectors

- A vector in R can be thought of as a list of numbers, and can be created using the `c()` command.
  - The entries in the vector are separated by commas–for example, `c(1, 1, 2, 3, 5, 8, 13, 21, 34)`.
  - The Python equivalent of a vector is a list, created by `[]`.
- The colon (`:`) in R defines a sequence, and can be read as "to". For example, the line of code `41:50` produces a sequence of integers, stored as a vector, from 41 to 50.
- Vector arithmetic in R works differently than arithmetic with integers or numeric values–arithmetic is vectorized, which is one of the advantages of R.

Reviewing Data Types and Structures

- Consider an example with your first name stored as a character string `name`, a number describing the amount of siblings you have as `siblings`, and a vector containing the digits of your phone number as `phone`.
- Once you save an object (or use an object stored in R), you may need to check the storage type of the object. You can do so using the functions `class()` and `mode()`.

```
name <- "Amy"
siblings <- 6
phone <- c(6, 3, 0, 8, 7, 5)

test_vec <- c(name, siblings, phone)
```

Lists

- The previous line of code creates a vector of all of the different values in the objects `name`, `siblings`, and `phone`, but you can't tell which object each value belongs to–everything is combined.
- Furthermore, the storage type of the entire vector is automatically character, even though `siblings` and `phone` are both numeric.
- There are times when it useful to create a vector of different objects, where the objects maintain their original structures. This is essentially the function of a list in R.

```
test_list <- list(name, siblings, phone)
test_list
```

```
[[1]]
[1] "Amy"
```

More Lists

- Each of the original objects are contained in `test_list`. Theoretically, you could extract each of these objects in their original forms.

```
test_list[[1]]
```

```
[1] "Amy"
```

- Here, I extract my name, since it's the 1st element. You can also extract using the dollar sign operator.

```
test_list$Phone
```

```
NULL
```

Even More Lists

- We expected to extract the digits in our phone numbers, but we were unable to extract anything. This is because we need to supply names for each element in the list.

```
names(test_list) <- c("Name", "Siblings", "Phone")
names(test_list)
```

```
[1] "Name"     "Siblings" "Phone"
```

```
test_list$Phone
```

```
[1] 6 3 0 8 7 5
```

- You can find out which elements are contained in a list by using `names()`". You can also find it using `attributes()`.

Packages

- R comes with a decent amount of built-in functionality, but to do anything useful you will need to install and load **packages** that contain additional functionality.
- Similar to modules in Python, you only need to install a package once, which can be done in the lower righthand window in RStudio under the "Packages" tab.
- Packages must be loaded any time you want to use them in an R document. You load packages with the `library` command. * We are going to work with `dplyr` and `ggplot2`.

Introduction to the tidyverse

- dplyr and ggplot2 are a part of the tidyverse.
- The tidyverse is a suite of packages that is designed to help make your code elegant–easy for you to write, and easy for other people (including future you) to read.

Introduction to Data Frames

- Most of the `tidyverse` requires that you are working with a data structure called a data frame.
- A data frame is a special type of list–each element of the list is a vector, and all of the vectors are the same length.
- In a data frame, each vector (column) represents a different variable.
- Each row represents a different observation/subject.
- We will be using the `starwars` data frame.

Why ggplot2?

- ggplot2 is a "system for declaratively creating graphics".
- It relies on the "grammar of graphics", a philosophy developed by Hadley Wickham for creating intuitive and consistent graphics.

ggplot2

Grammar of Graphics

The layered grammar defines the components of a plot as:

- A default dataset and set of mappings from variables to aesthetics,
- One or more layers, with each layer having
  - One geometric object,
  - One statistical transformation,
  - One position adjustment, and
  - Optionally, one dataset and set of aesthetic mappings.
- One scale for each aesthetic mapping used,
- A coordinate system,
- The facet specification.

Wickham, Hadley. "A layered grammar of graphics." Journal of Computational and Graphical Statistics 19, no. 1 (2010): 3-28.

Introduction to ggplot2 1

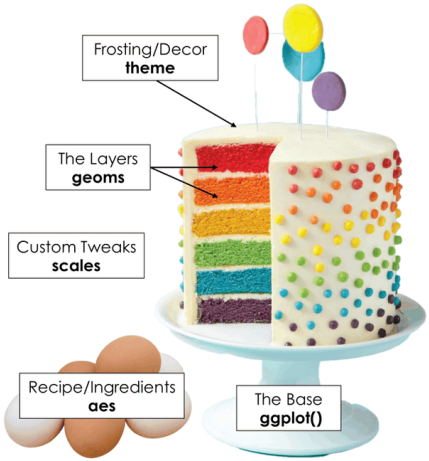
ggplot is a little bit like cake...

We *always* start by setting up the foundation with **ggplot()**

We specify our ingredients (data variables) with an **aes mapping**

We can create *layers* to our plot with **geoms**

We can style our ~~cake~~ ggplot with **themes.** We have out-of-the-box options, or we can go totally custom!



Introduction to ggplot2 2

- You tell ggplot2 what data to use, how to map the variables to the different aesthetics of the graph, and what type of graph you need–ggplot2 takes care of the rest!
- First, we start by providing the data and mapping the variables to the graph's *aesthetics*–what's on the x-axis, what color the graph is, etc.
- A sample line of code for investigating the heights of different characters appears below.

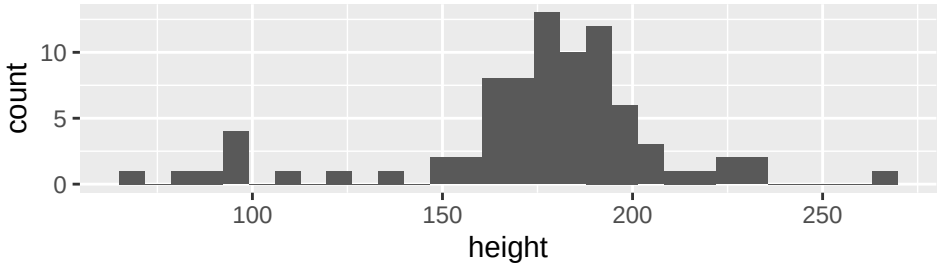
```
ggplot(data = starwars, aes(x = height))
```

- If you run this chunk, a nearly blank box should appear–but no actual graph.

Histograms 1

- To add a plot, we add geometric objects, or *geoms*. Specifically, for a histogram, we use `+ geom_histogram()`.

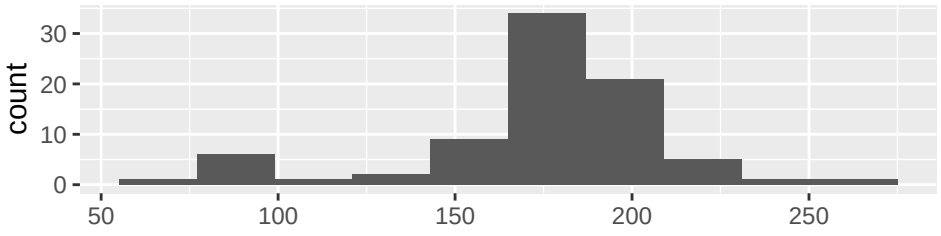
```
ggplot(data = starwars, aes(x = height)) +
  geom_histogram()
```



Histograms 2

- Do you see a warning message saying "`Pick better value with binwidth`"? This error is unique to `geom_histogram()`.
- To fix the warning, we can change the number of bins by adding a new argument, `bins = 10`, into the "`geom_histogram()`" function.

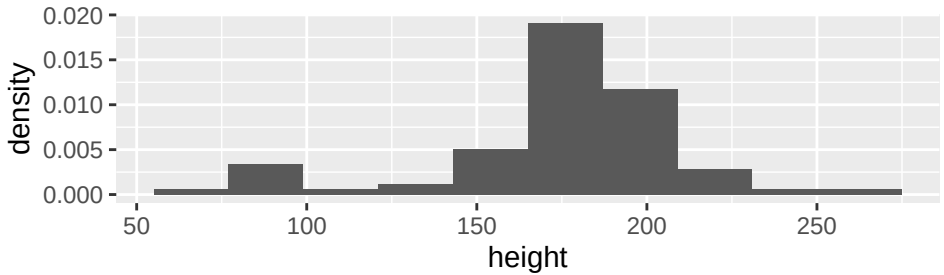
```
ggplot(data = starwars, aes(x = height)) +
  geom_histogram(bins = 10)
```



Densities 1

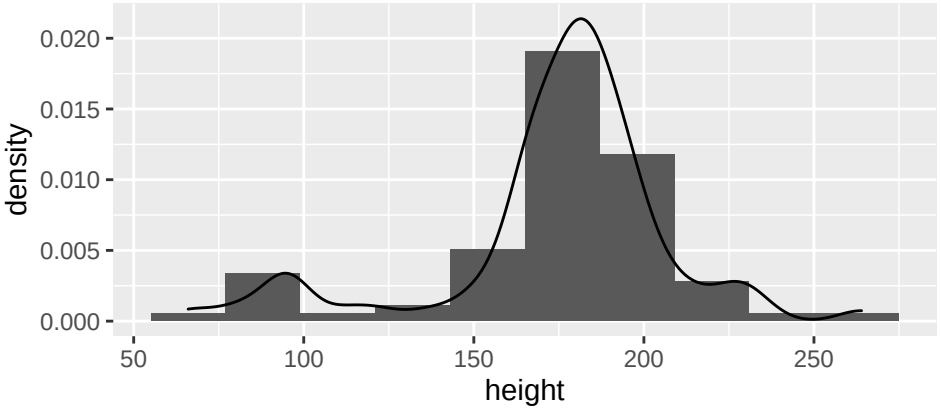
- Notice that on the y-axis, we are displaying the counts of the observations.
- We can change this to instead display the frequency by adding another argument to `geom_histogram()`.

```
ggplot(data = starwars, aes(x = height)) +
  geom_histogram(bins = 10, aes(y=..density..))
```



Densities 2

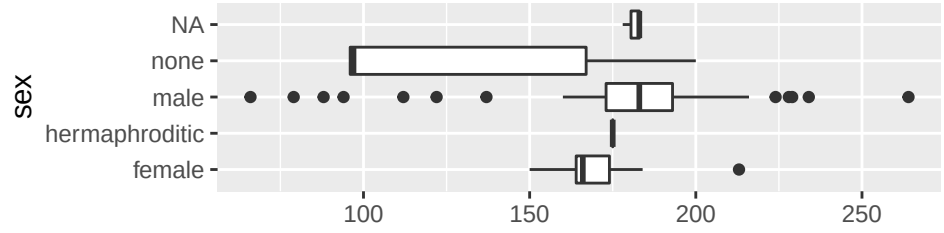
- This way, we can directly compare a histogram and a density plot!



Boxplots

- Now, we can try creating a box plot. Remember that we use boxplots to look at the distributions over different levels of a categorical variable.
- To create a plot with a boxplot for every level, you can add a `y =` argument to the aesthetics function.

```
ggplot(data = starwars, aes(x = height, y = sex)) +
  geom_boxplot()
```

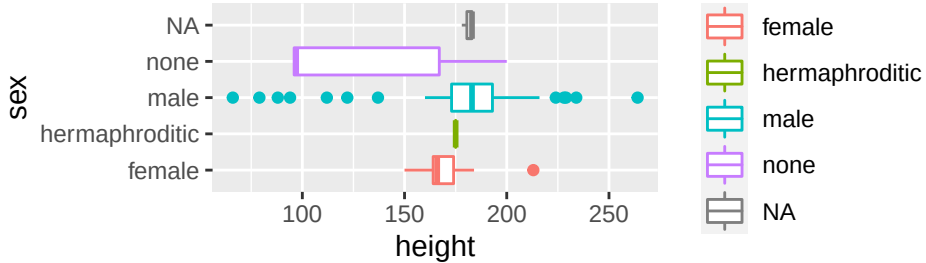


Colors 1

- You may also be interested in changing colors on your graph.
  - Adding a `color = "red"` argument to the `geom_boxplot()` statement.
  - You can use this R color guide to look up the available colors.
- In addition, you can change the color according to the categorical variable, so that each level is its own color. To do this, you can change the `color` argument to be the name of a variable (in this case, `sex`) inside the `aes()` statement (rather than `geom_boxplot()`).

Colors 2


```
ggplot(data = starwars, aes(x = height, y = sex, color = sex)) +
  geom_boxplot()
```



Changing Axes 1

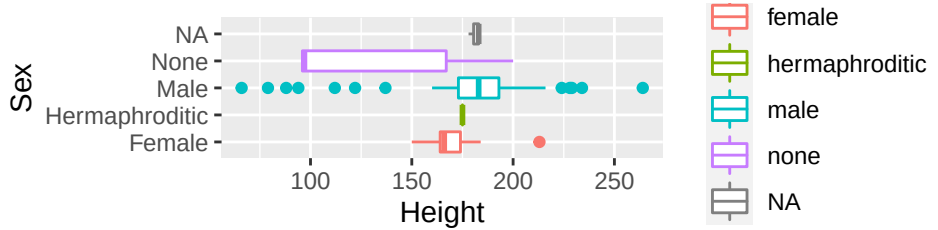
- Obviously, we want to have "nice" titles for our graphs.
  - `ggtitle()`
  - `xlab()`
  - `ylab()`
- There may also be times where you want to change other components of the axes. In those cases, you want to use `scale_x_*()` or `scale_y_*()`.
  - The star should be replaced by `continuous` or `discrete`, depending on what your variable is.

Changing Axes 2

- Arguments for these functions include:
  - `name`, which defines the name of the axis.
  - `breaks`, which defines the placement of the axis tick marks.
  - `labels`, which defines the labels of the axis tick marks.
  - `limits`, which defines the limits of the axis.
  - `palette`, where you can call existing or custom palettes.

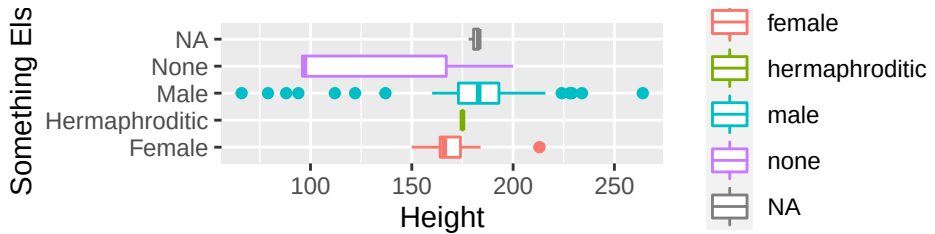
Changing Axes 3


```
ggplot(data = starwars, aes(x = height, y = sex, color = sex)) +
  geom_boxplot() +
  xlab("Height") +
  ylab("Sex") +
  scale_y_discrete(labels = c("Female", "Hermaphroditic", "Male",
                              "None", "NA"))
```



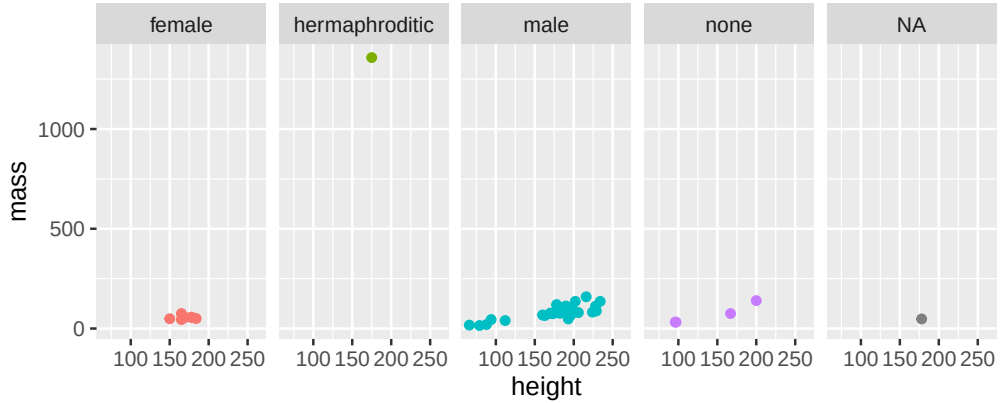
Order Matters!


```
ggplot(data = starwars, aes(x = height, y = sex, color = sex)) +
  geom_boxplot() +
  xlab("Height") +
  ylab("Sex") +
  scale_y_discrete(name = "Something Else", labels = c("Female",
                   "Hermaphroditic", "Male", "None", "NA"))
```



Facetting 1

There might be times where you prefer to have multiple small graphs rather than one large graph with a ton of different variables encoded.



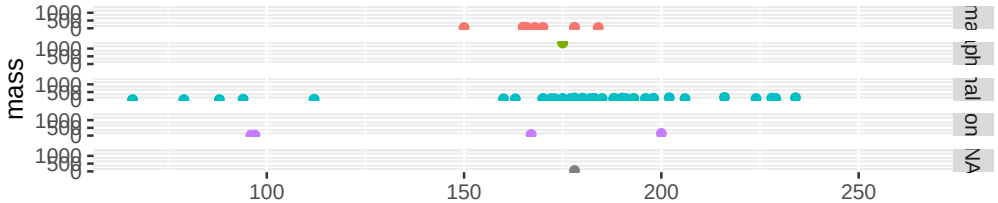
Facetting 2

- ggplot2 referes to these small multiples as "facets".
- Facets can be added to the graph with one relatively simple line of code.
  - `facet_wrap()`
  - `facet_grid()`.
- Each facet command requires a formula in R. Formulas in R are defined with the ~, which you can think of as the word "by".

Facetting by Row

- To separate by row, place the variable name on the left hand side of the ~ and a period on the right hand side.

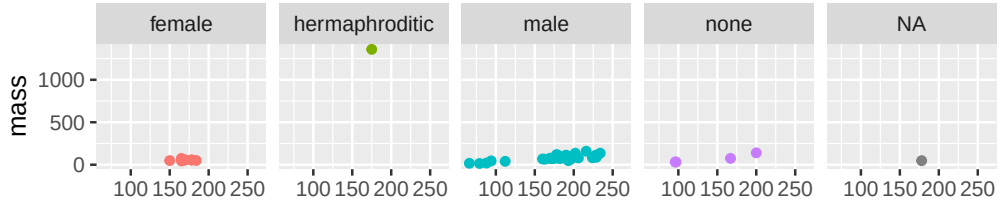
```
ggplot(data = starwars, aes(x = height, y = mass, color = sex)) +
  facet_grid(sex~.) +
  geom_point() +
  theme(legend.position = "none")
```



Facetting by Column

- To separate by column, place the variable name on the right hand side of the ~ and a period on the left hand side.

```
ggplot(data = starwars, aes(x = height, y = mass, color = sex)) +
  facet_grid(.~sex) +
  geom_point() +
  theme(legend.position = "none")
```

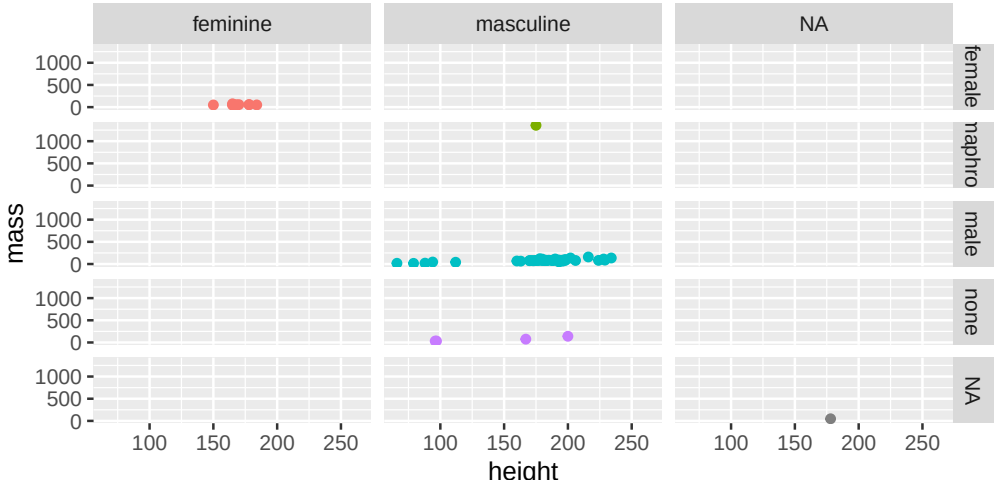


Facetting by Row and Column 1

- You can facet by more than one variable!

```
ggplot(data = starwars, aes(x = height, y = mass, color = sex)) +
  facet_grid(sex~gender) +
  geom_point() +
  theme(legend.position = "none")
```

Facetting by Row and Column 2

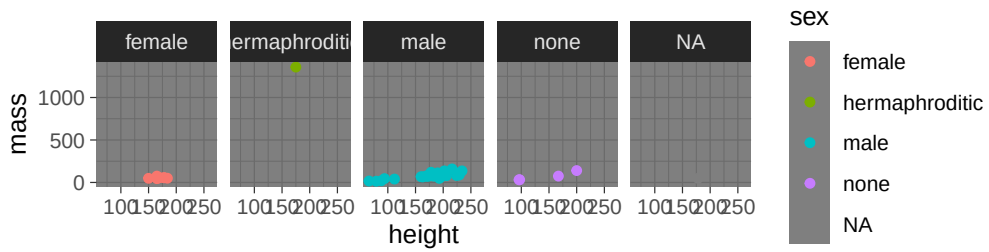


Themes 1

- If you want all of your graphs to have a consistent appearance, you might want to use different themes.
- Themes control the "non-data" aspects of your graph.
  - `theme_grey()`, `theme_bw()`, `theme_linedraw()`, `theme_light()`, `theme_dark()`, `theme_minimal()`, `theme_classic()`, `theme_void()`, `theme_test()`
- They work the same as adding geoms.

Themes 2


```
ggplot(data = starwars, aes(x = height, y = mass, color = sex)) +
  facet_grid(.~sex) +
  geom_point() +
  theme_dark()
```



Themes 3

- You can also control ALL of the non-data aspects (for example, legend position, text size, label justification, etc.) with `theme()`–see `legend.position = "none"` in the previous slides.
- Other arguments include:
  - `line` (all line elements)
  - `rect` (all rectangular elements)
  - `text` (all text elements)
  - `title` (all title elements)
- There are also a ton of "sub" elements! See `?theme()`. As a note, you are unlikely to have these memorized–I have to look up exactly how to use the different arguments most times.

Introduction to dplyr

- `dplyr` is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges. `dplyr` is part of the `tidyverse`, so you may find it very similar to `ggplot2`.
  - `arrange()` changes the ordering of the rows.
  - `filter()` picks cases based on their values (good for removing "bad" rows).
  - `mutate()` adds new variables that are functions of existing variables (good for changing storage types).
  - `select()` picks variables based on their names (good for removing unnecessary columns).
  - `summarise()` reduces multiple values down to a single summary.
- These all combine naturally with `group_by()`, which allows you to perform any operation "by group".

filter()

- You may have seen that `sex` has at least three values in it–`"male"`, `"none"`, `"female"`.
  - This is not a complete picture of the variable. Using `unique()`, you can see that the values in `starwars$sex` are `"male"`, `"none"`, `"female"`, `"hermaphroditic"`, and `NA`.
  - `"male"` and `"female"` refer to the standard definitions of sex.
  - `"none"` refers to droids like R2-D2 and C-3PO who are manufactured and cannot reproduce.
  - `"hermaphroditic"` refers to one specific character, Jabba the Hutt (members of the Hutt species have multiple sex organs and and need no partners to produce children, see Wookieepedia for more details), and
  - The last value, `NA`, doesn't refer to a level of this variable at all, but instead *indicates a missing value* in the dataframe.

More `filter()`

- To find out more about the missing values, we can use the `filter()` function. Run the code below.

- Let's break down this code:
  - First, we specify that we would like to be looking in the `starwars` dataframe.
  - Second, we link our two statements, `starwars` and `filter()`, with the pipe operator, %>%. This is actually necessary anytime you use a `dplyr` verb.
  - The filter function itself "filters" the data, and prints out only values that satisfy a certain condition. In this case, the condition is that the value for `sex` is missing (you can confirm this in the output).

Logical Operators

- How do I write the conditions? I can use a special type of operator in R called a **logical operator**, a symbol that links two statements and results in a true or false statement.
- You can find out more about other logical operators here.
- Notice in our filter statement, we checked `is.na(sex) == TRUE`–`is.na()` is a special function that will evaluate each value to see if it is missing.
- If the value is missing, `is.na()` returns a `TRUE`, so we pulled all the values of `sex` that are missing by finding the values that are equal to `TRUE`.
- Hopefully, you can see that four characters (Ric Olié, Quarsh Panaka, Sly Moore, and Captain Phasma) all have missing values for `sex`.

Back to `filter()`

- Let's say we want to ignore the missing values for now.
- I can rewrite my `filter()` code and **resave the object** to use for later!

summarize()

- Now, we can begin to summarize the values.
- You will need to know some basic functions, like `mean()`, `sd()` (standard deviation), `min()`, `median()`, and `maximum()`.
- Because we are using `dplyr`, the general syntax is the same!

- Let's break down this code:
  - First, we specify that we would like to be looking in the `starwars2` dataframe.
  - Second, we link our two statements, `starwars` and `summarize()`, with the pipe operator, %>%.
  - The `summarize()` function itself needs a function inside so it knows how to summarize. I have used `mean()`.

More `summarize()`

- What if you want to calculate more than one summary statistic?
- Easy! List the other statistics you would like, separated by commas.

group_by()

- What if we would like to produce output that "matches" our boxplots, in the sense that we have summary statistics for multiple groups.
- Also easy! We can use the `group_by()` function.

- Let's break down this code:
  - First, we have much of the same code from before (lines 1 and 3).
  - Second, we link another line of code with the pipe operator, `%>%`. Every line but the last needs one!
  - The `group_by()` function needs a variable inside so it knows how to group. I have used `sex`.

Combining R and Python 1

If you DON'T want to learn `dplyr`, you can also do all of your data wrangling in Python via R!

```{r}
library(reticulate)
```

```{python}
r.starwars.head()
```

```
              name  ...                        starships
0  Luke Skywalker  ...  [X-wing, Imperial shuttle]
1           C-3PO  ...                               []
2           R2-D2  ...                               []
```

Combining R and Python 2


```
```{python}
starwars = r.starwars
```
```

```
```{r, fig.height = 1.5, eval = FALSE}
ggplot(data = py$starwars, aes(x = height)) +
  geom_histogram()
```
```

Follow this tutorial if you are interested.

R Resources

- Hands-On Programming with R is a nice, friendly introduction to R. You can buy a physical copy, or use the online copy for free.
- R for Data Science is a great resource if you already have a bit of experience with R and want to focus on the `tidyverse`! You can buy a physical copy, or use the online copy for free.
- Once you get the hang of R, there are some great cheat sheets that serve as wonderful, fast references. Here are some of my favorites: Base R, dplyr, ggplot2, LaTeX, R Syntax Comparison, and tidyR. Translations also available!
  - The ggplot2 cheat sheet is especially useful for looking up the appropriate geom you need.
  - The first page of the cheat sheet shows a list of all the possible geoms you can have.