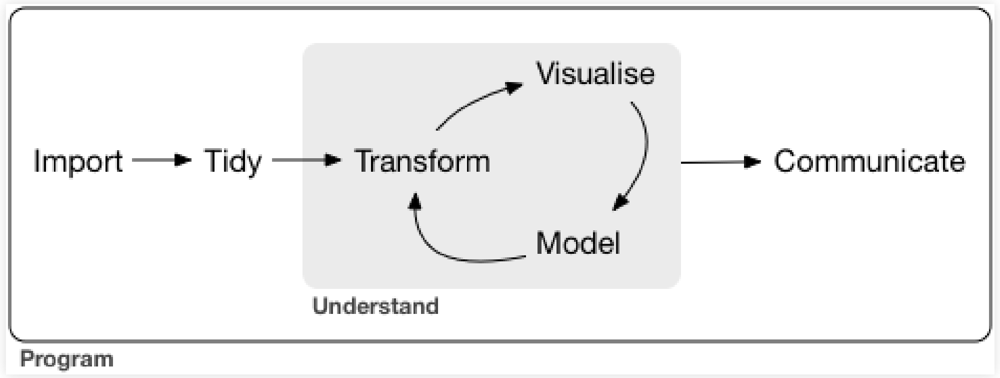
R for Data Science

Alejandro Schuler, adapted from Steve Bagley and based on R for Data Science by Hadley Wickham
2019

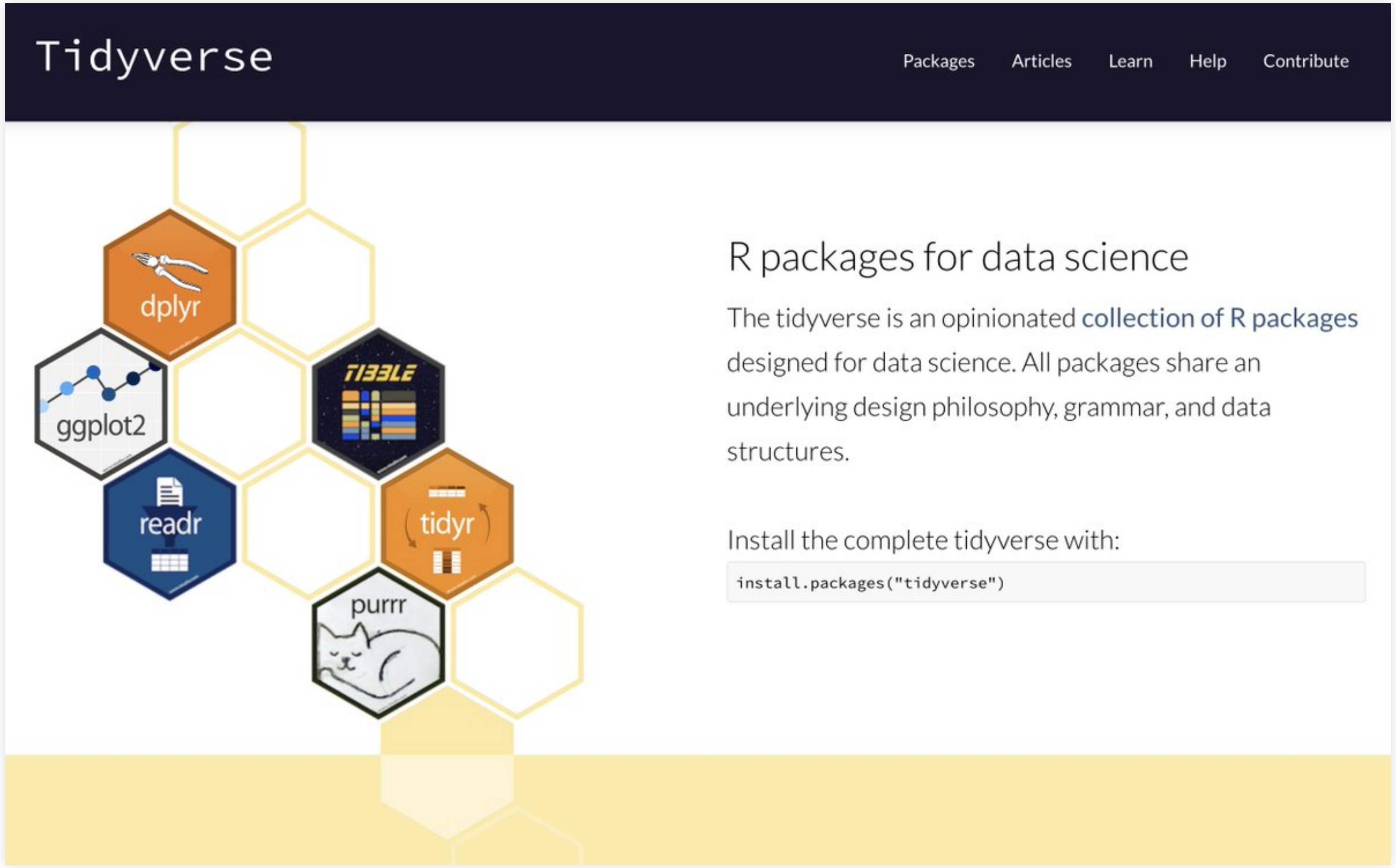
Goals of this course

By the end of the course you should be able to…

- comfortably use R through the Rstudio interface
- read and write tabular data between R and flat files
- subset, transform, summarize, join, and plot data
- write reusable and readable programs
- seek out, learn, and integrate new packages into your analyses



Tidyverse



R packages for data science

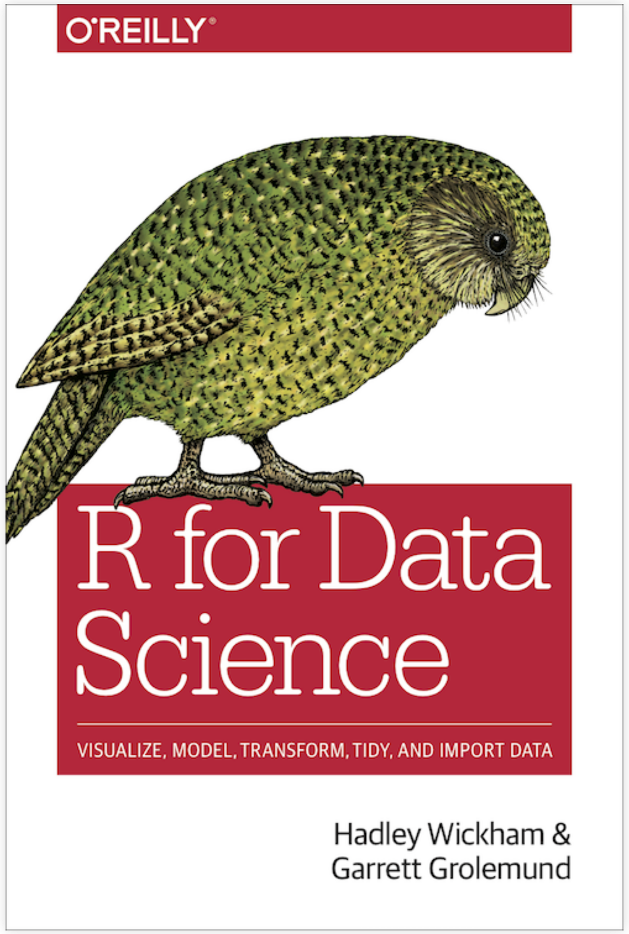
The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

Resources for this course

R for Data Science (R4DS): https://r4ds.had.co.nz


O'REILLY

R for Data Science

VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA

Hadley Wickham & Garrett Grolemund

Cheatsheets: https://www.rstudio.com/resources/cheatsheets/

Website: https://github.com/alejandroschuler/r4ds-courses/tree/advance-2021

Basics and Plotting

Learning Goals:

- issue commands to R using the Rstudio REPL interface
- load a package into R
- read some tabluar data into R
- visualize tabluar data using ggplot geoms, aesthetics, and facets

Basics

The basics of interaction using the console window

The R console window is the left (or lower-left) window in RStudio. The R console uses a “read, eval, print” loop. This is sometimes called a REPL.

- Read: R reads what you type …
- Eval: R evaluates it …
- Print: R prints the result …
- Loop: (repeat forever)

A simple example in the console

- The box contains an expression that will be evaluated by R, followed by the result of evaluating that expression.

```
1 + 2
[1] 3
```

- `3` is the answer
- Ignore the `[1]` for now.

- R performs operations (called *functions*) on data and values

- These can be composed arbitrarily

```
log(1 + 3)
[1] 1.386294
paste("The answer is", log(1 + 3))
[1] "The answer is 1.38629436111989"
```

How do I…

- typing `?function_name` gives you information about what the function does
- Google is your friend. Try “function_name R language” or “how do I X in R?”. I also strongly recommend using “tidyverse” in your queries or the name of a tidyverse package (more in a moment) that has related functions
- stackoverflow is your friend. It might take some scrolling, but you will eventually find what you need

Quadratic Equation

Solutions to a polynomial equation $ax^2 + bx + c = 0$ are given by

Packages

- The amazing thing about programming is that you are not limited to what is built into the language
- Millions of R users have written their own functions that you can use
- These are bundled together into *packages*
- To use functions that aren't built into the “base” language, you have to tell R to first go download the relevant code, and then to load it in the current session

```
install.packages("tidyverse")  # go download the package called 'tidyverse'- only have to do this once
library("tidyverse")  # load the package into the current R session - do this every time you use R and need functions from this package
```

Packages

- The `tidyverse` package has a function called `read_csv()` that lets you read csv (comma-separated values) files into R.
- csv is a common format for data to come in, and it's easy to export csv files from microsoft excel, for instance.

```
# I have a file called 'lupusGenes.csv' in a folder called data
genes = read_csv("../data/LupusGenes.csv")
Error in read_csv("../data/LupusGenes.csv"): could not find function "read_csv"
```

- This fails because I haven't yet loaded the `tidyverse` package

```
library(tidyverse)
```

```
genes = read_csv("../data/LupusGenes.csv")
```

- Now there is no error message

Packages

- packages only need to be loaded once per R session (session starts when you open R studio, ends when you shut it down)
- once the package is loaded it doesn't need to be loaded again before each function call

```r
poly = read_csv("../data/poly.csv")  # reading another csv file
```

Using R to look at your data

Data analysis workflow

1. Read data into R (done!)
2. Manipulate data
3. Get results, **make plots and figures**

Getting your data in R

- Getting your data into R is easy. We already saw, for example:

```
genes = read_csv("https://raw.githubusercontent.com/alejandroschuler/r4ds-
courses/advance-2021/data/lupusGenes.csv")
```

- `read_csv()` requires you to tell it where to find the file you want to read in
  - Windows, e.g.: `"C:\Users\me\Desktop\myfile.csv"`
  - Mac, e.g.: `"/Users/me/Desktop/myfile.csv"`
  - Internet, e.g.: `"http://www.mywebsite.com/myfile.csv"`
- If your data is not already in csv format, google “covert X format to csv” or “read X format data in R”
- We'll learn the details of this later, but this is enough to get you started!

Looking at data

- `genes` is now a dataset loaded into R. To look at it, just type

```
genes
# A tibble: 59 x 11
   sampleid   age gender ancestry  phenotype FAM50A ERCC2 IFI44 EIF3L  RSAD2
   <chr>    <dbl> <chr>  <chr>     <chr>      <dbl> <dbl> <dbl> <dbl>  <dbl>
 1 GSM3057…    70 F      Caucasi… SLE         18.6  4.28  18.0 182.    25.5
 2 GSM3057…    78 F      Caucasi… SLE         20.3  3.02  21.1 157.    37.2
 3 GSM3057…    64 F      Caucasi… SLE         21.4  4.00 488.  169.   792. 
 4 GSM3057…    32 F      Asian    SLE         17.1  4.49  34.0 149.    60.7
 5 GSM3057…    33 F      Caucasi… SLE         20.9  5.00  34.4 224.    60.8
 6 GSM3057…    46 M      Maori    SLE         15.8  3.96 466.  111.  1382. 
 7 GSM3057…    45 F      Asian    SLE         18.9  6.04 299.  157.   926. 
 8 GSM3057…    67 M      Caucasi… SLE         27.6  4.77  21.8 265.    20.6
 9 GSM3057…    33 F      Caucasi… SLE         15.4  3.88 700.   98.6 1652. 
10 GSM3057…    28 F      Caucasi… SLE         19.9  7.21 278.  217.   972. 
# … with 49 more rows, and 1 more variable: VAPA <dbl>
```

This is a **data frame**, one of the most powerful features in R (a “tibble” is a kind of data frame).

- Similar to an Excel spreadsheet.
- One row ~ one instance of some (real-world) object.
- One column ~ one variable, containing the values for the corresponding instances.
- All the values in one column should be of the same type (a number, a category, text, etc.), but different columns can be of different types.

The Dataset

```
genes
# A tibble: 59 x 11
   sampleid   age gender ancestry phenotype FAM50A ERCC2 IFI44 EIF3L  RSAD2
   <chr>    <dbl> <chr>  <chr>    <chr>      <dbl> <dbl> <dbl> <dbl>  <dbl>
 1 GSM3057…    70 F      Caucasi… SLE         18.6  4.28  18.0 182.    25.5
 2 GSM3057…    78 F      Caucasi… SLE         20.3  3.02  21.1 157.    37.2
 3 GSM3057…    64 F      Caucasi… SLE         21.4  4.00 488.  169.   792. 
 4 GSM3057…    32 F      Asian    SLE         17.1  4.49  34.0 149.    60.7
 5 GSM3057…    33 F      Caucasi… SLE         20.9  5.00  34.4 224.    60.8
 6 GSM3057…    46 M      Maori    SLE         15.8  3.96 466.  111.  1382. 
 7 GSM3057…    45 F      Asian    SLE         18.9  6.04 299.  157.   926. 
 8 GSM3057…    67 M      Caucasi… SLE         27.6  4.77  21.8 265.    20.6
 9 GSM3057…    33 F      Caucasi… SLE         15.4  3.88 700.   98.6 1652. 
10 GSM3057…    28 F      Caucasi… SLE         19.9  7.21 278.  217.   972. 
# … with 49 more rows, and 1 more variable: VAPA <dbl>
```

This is a subset of a real RNA-seq (GSE112087) dataset comparing RNA levels in blood between lupus (SLE) patients and healthy controls.

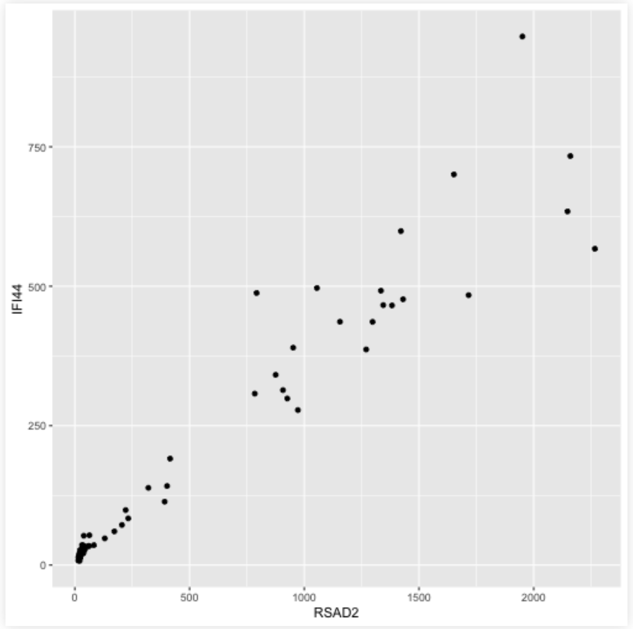
- 29 SLE Patients, 30 Healthy Controls
- We have basic metadata as well as the levels of multiple genes in blood.
- Let's see if we can find anything interesting from this already-generated data!

Investigating a relationship

Let's say we're curious about the relationship between two genes RSAD2 and IFI44.

- Can we use R to make a plot of these two variables?

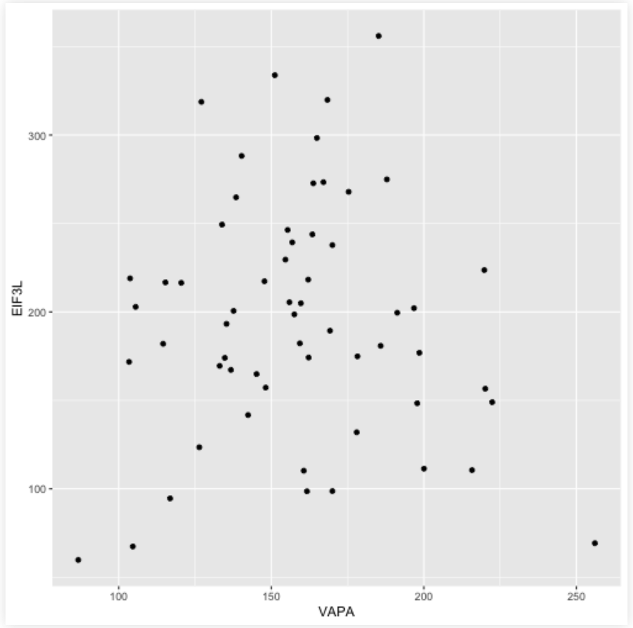
```
ggplot(genes) +
  geom_point(aes(x = RSAD2, y =
IFI44))
```



- `ggplot(dataset)` says "start a chart with this dataset"
- `+ geom_point(...)` says "put points on this chart"
- `aes(x=x_values y=y_values)` says "map the values in the column `x_values` to the x-axis, and map the values in the column `y_values` to the y-axis" (`aes` is short for *aesthetic*)

ggplot

```
ggplot(genes) +
  geom_point(aes(x = VAPA, y = EIF3L))
```



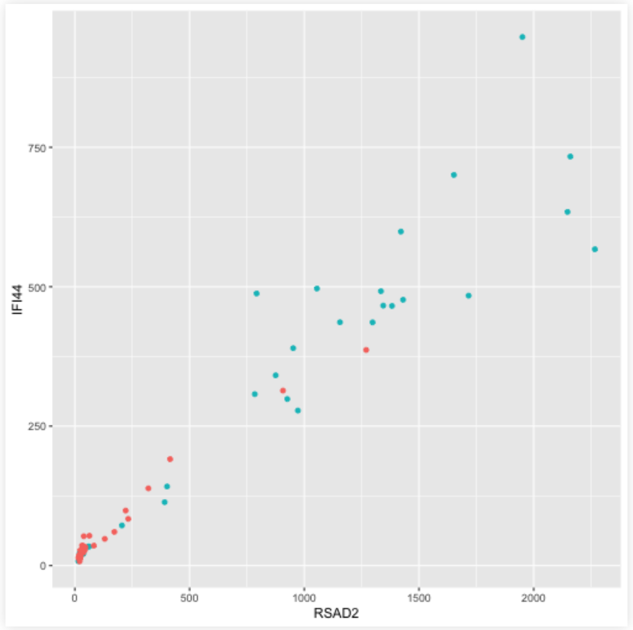
- `ggplot` is short for "grammar of graphics plot"
  - This is a language for describing how data get linked to visual elements
- `ggplot()` and `geom_point()` are functions imported from the `ggplot2` package, which is one of the "sub-packages" of the `tidyverse` package we loaded earlier

Investigating a relationship

Make a scatterplot of `phenotype` vs `IFI44` (another gene in the dataset). The result should look like this:

Investigating a relationship

Let's say we're curious about the relationship between RSAD2 and IFI44.



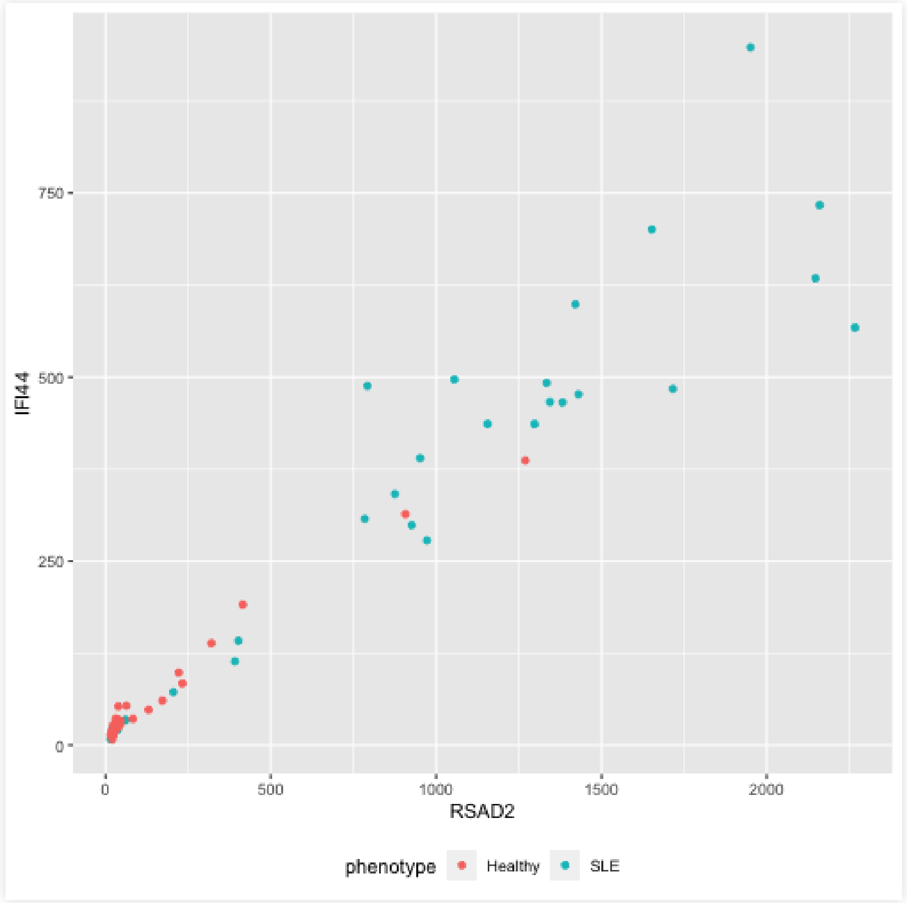
- What's going on here? It seems like there are two clusters.
- What is driving this clustering? Age? Sex? Ancestry? Phenotype?

Aesthetics

- Aesthetics aren't just for mapping columns to the x- and y-axis
- You can also use them to assign color, for instance

```
ggplot(genes) +
  geom_point(aes(x = RSAD2,
                 y = IFI44,
                 color = phenotype))
```

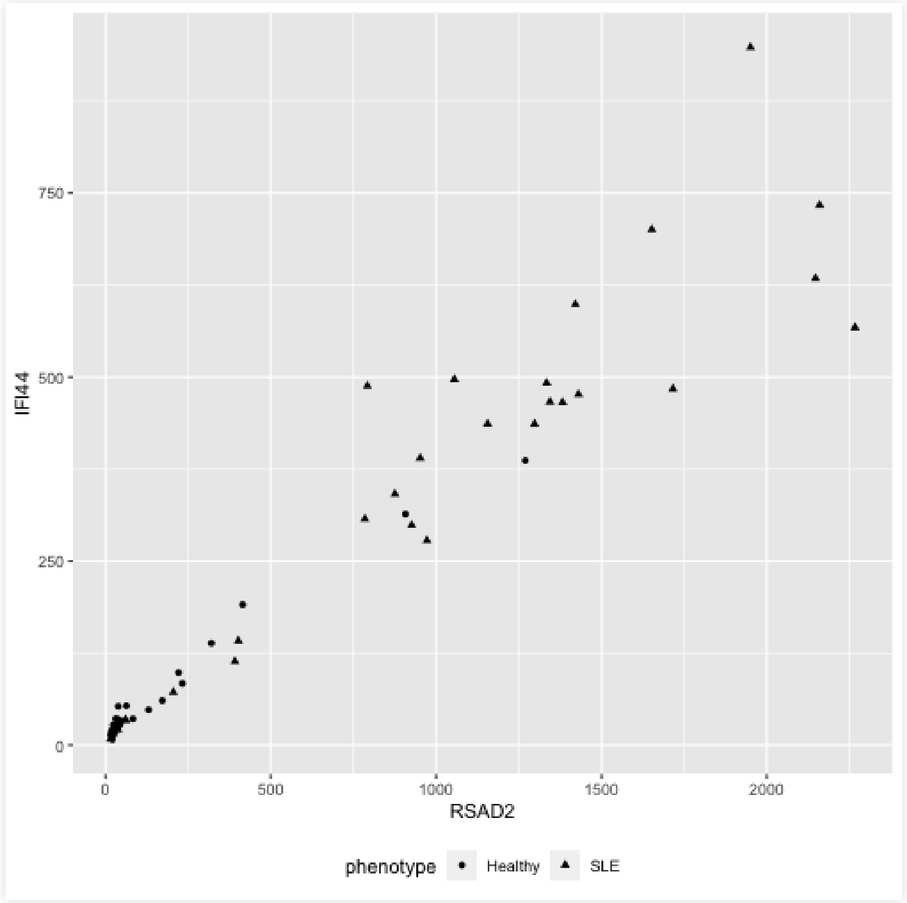
- ggplot automatically gives each value of the column a unique level of the aesthetic (here a color) and adds a legend
- What did we learn about the genes that we are interested in?



Aesthetics

- Aesthetics aren't just for mapping columns to the x- and y-axis
- We could have used a shape

```
ggplot(genes) +
  geom_point(aes(
    x = RSAD2,
    y = IFI44,
    shape=phenotype
  ))
```



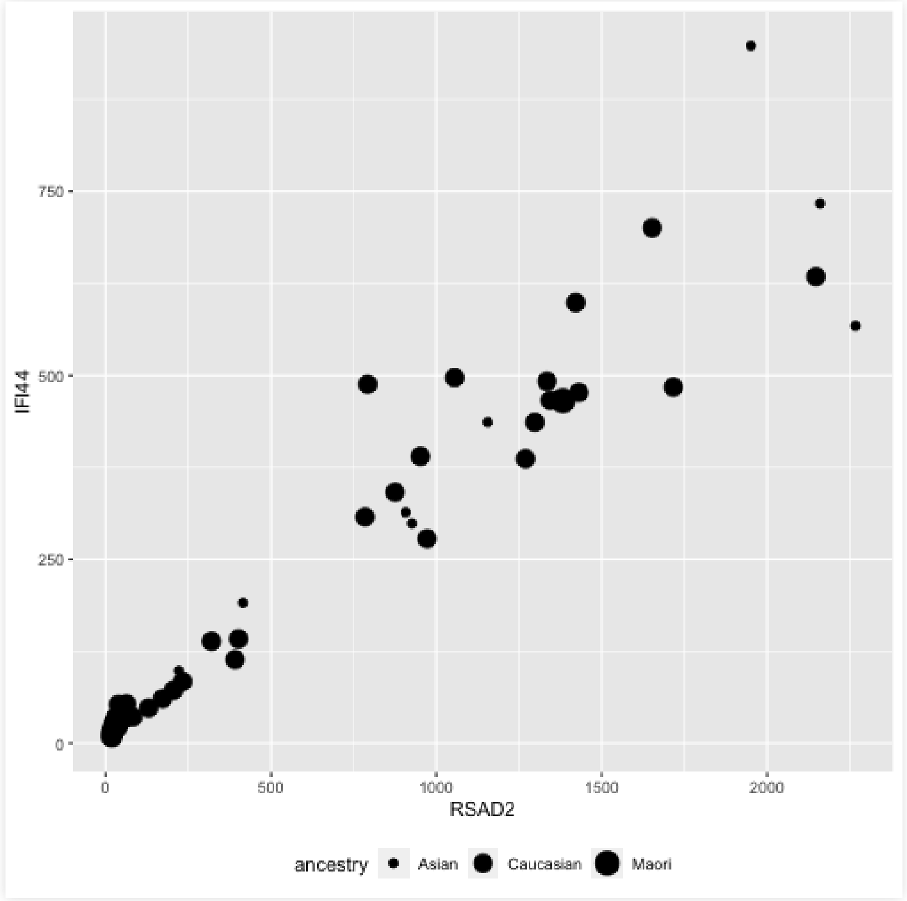
Aesthetics

- Aesthetics aren't just for mapping columns to the x- and y-axis
- Or size

```
ggplot(genes) +
  geom_point(aes(
    x = RSAD2,
    y = IFI44,
    size=ancestry
  ))
```

- This one doesn't really make sense because we're mapping a categorical variable to an aesthetic that can take continuous values that imply some ordering

```
Warning: Using size for a discrete
variable is not advised.
```

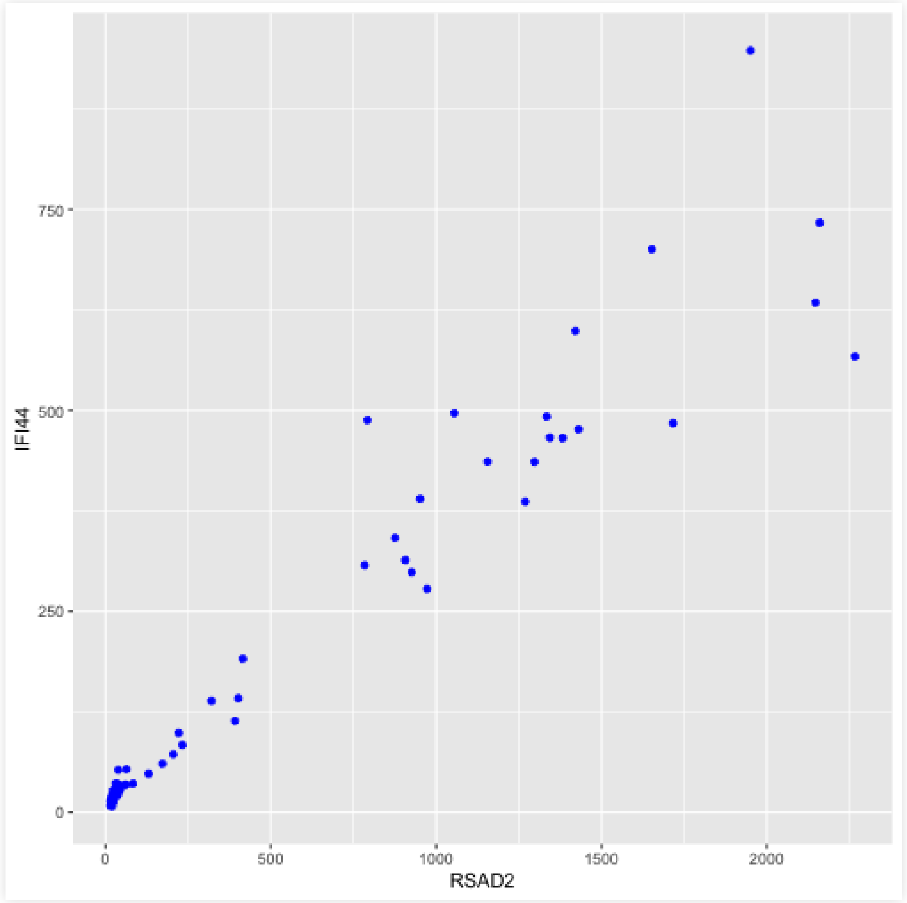


Aesthetics

- If we set a property *outside* of the aesthetic, it no longer maps that property to a column.

```
ggplot(genes) +
  geom_point(
    aes(
      x = RSAD2,
      y = IFI44
    ),
    color = "blue"
  )
```

- However, we can use this to assign fixed properties to the plot that don't depend on the data



Exercise

Can you recreate this plot?

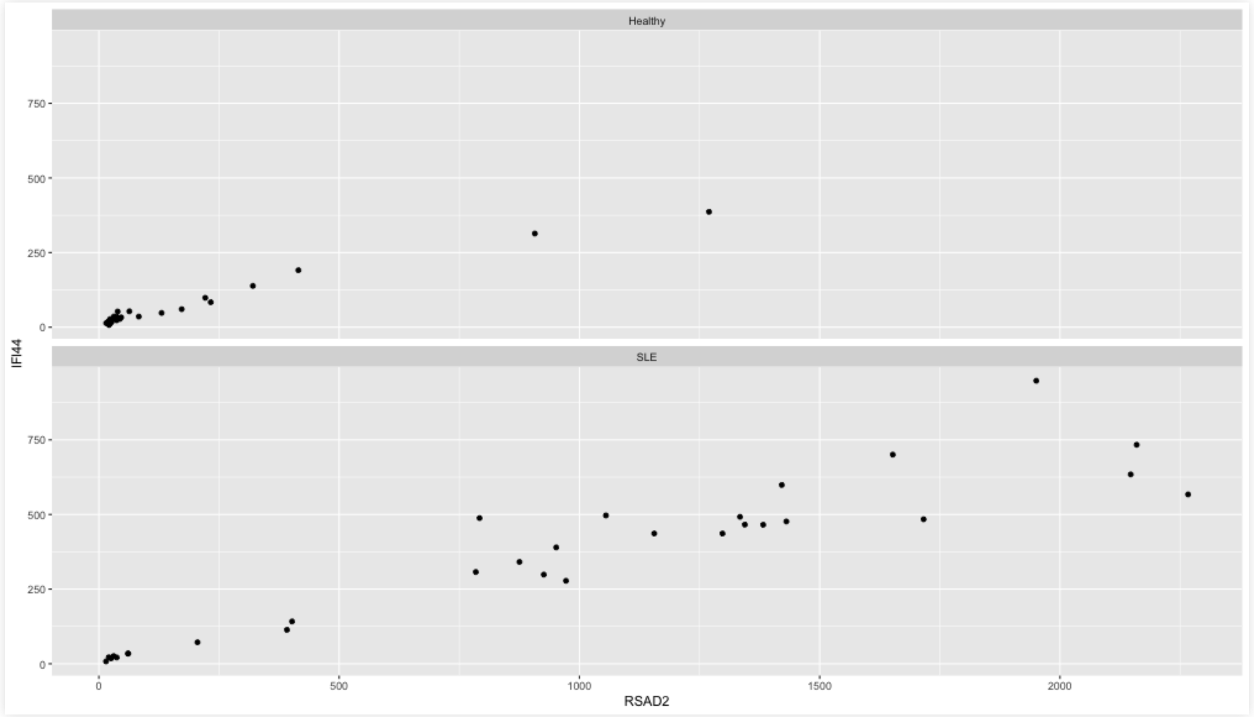
Exercise

What will this do? Why?

Facets

- Aesthetics are useful for mapping columns to particular properties of a single plot
- Use **facets** to generate multiple plots with shared structure

```
ggplot(genes) +
  geom_point(aes(x = RSAD2, y = IFI44)) +
  facet_wrap(~ phenotype, nrow = 2)
```

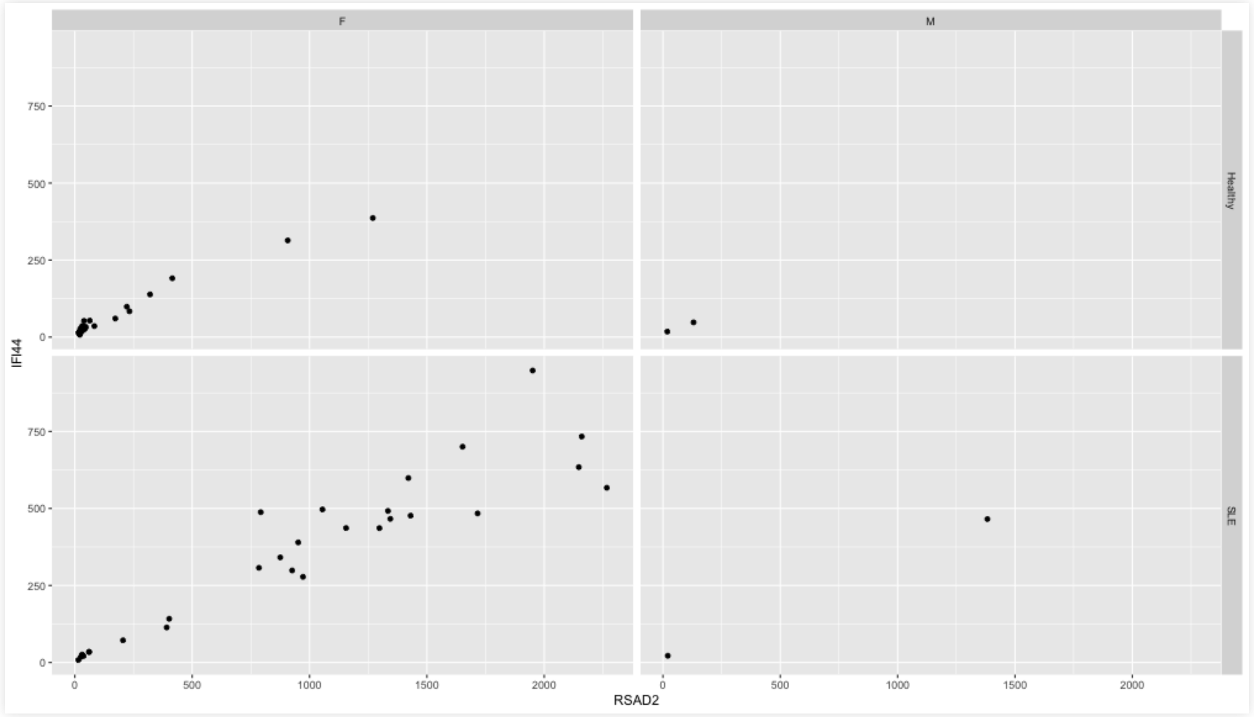


- `facet_wrap` is good for faceting according to unordered categories

Facets

- `facet_grid` is better for ordered categories, and can be used with two variables

```
ggplot(genes) +
  geom_point(aes(x = RSAD2, y = IFI44)) +
  facet_grid(phenotype ~ gender)
```



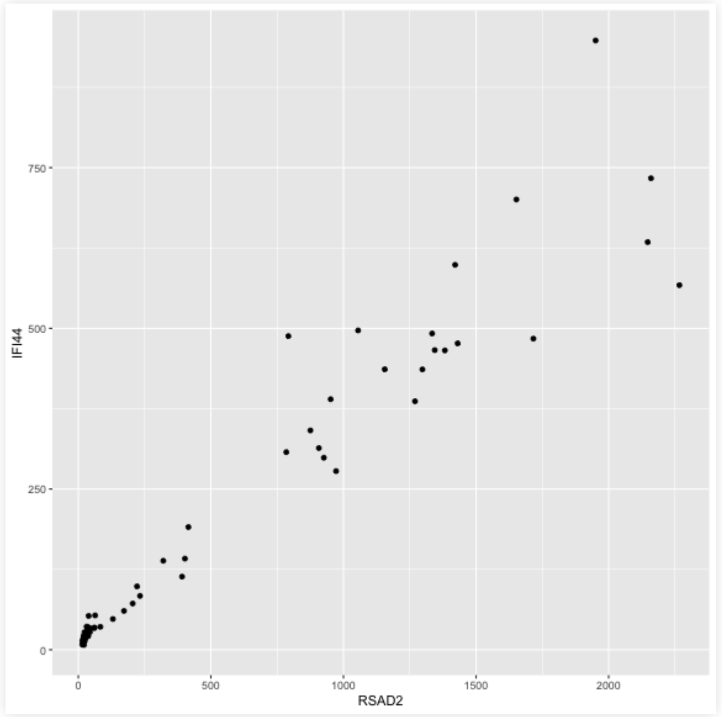
Exercise

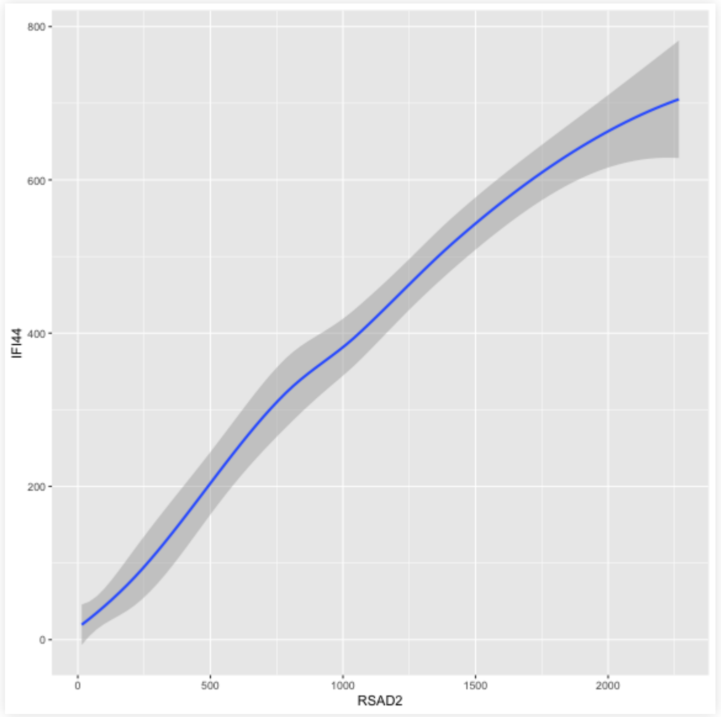
Run this code and comment on what role `.` plays:

Geoms

```
ggplot(genes) +
  geom_point(aes(x = RSAD2, y =
IFI44))
```

```
ggplot(genes) +
  geom_smooth(aes(x = RSAD2, y =
IFI44))
```



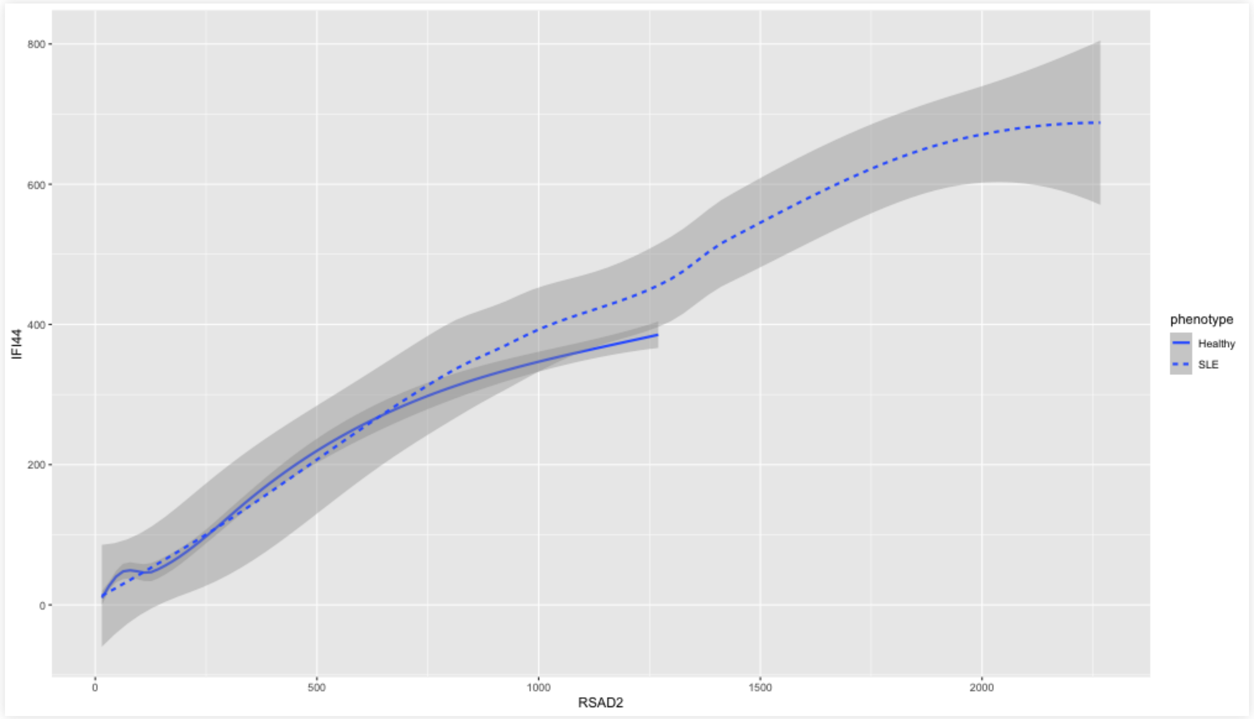


- Both these plots represent the same data, but they use a different geometric representation (“geom”)
- e.g. bar chart vs. line chart, etc.

Geoms

- Different geoms are configured to work with different aesthetics.
- e.g. you can set the shape of a point, but you can't set the "shape" of a line.
- On the other hand, you *can* set the "line type" of a line:

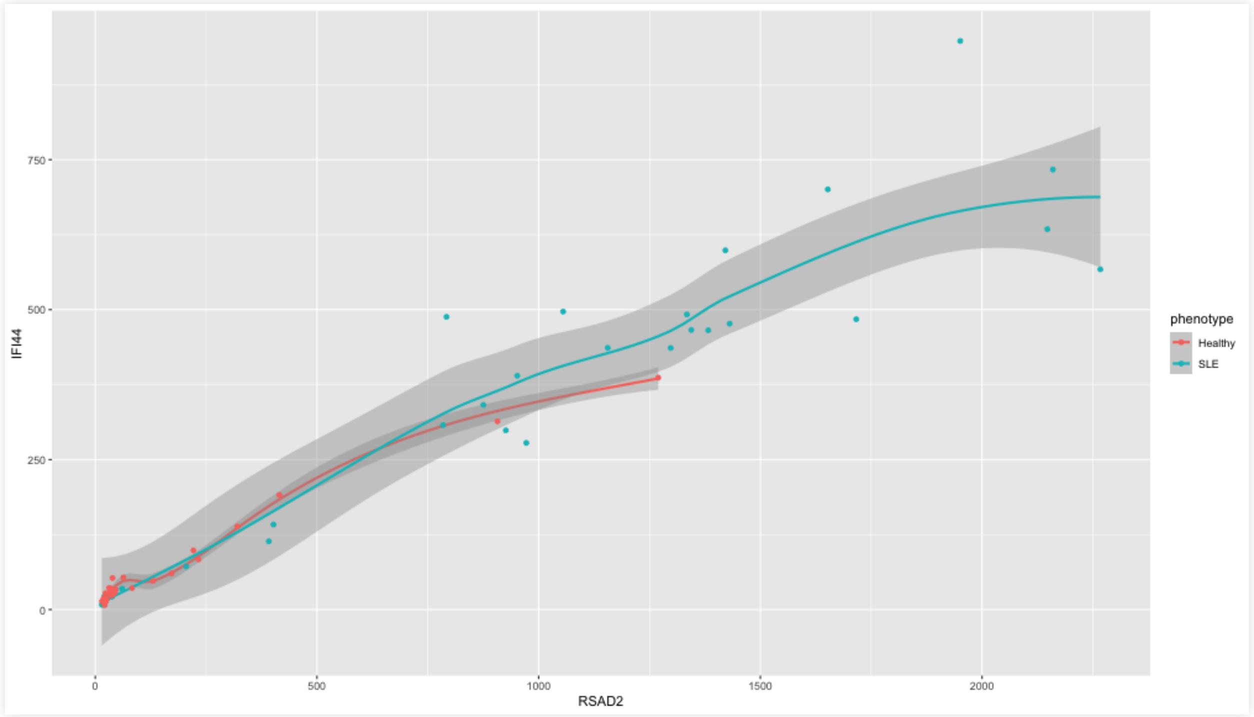
```
ggplot(genes) +
  geom_smooth(aes(x = RSAD2, y = IFI44, linetype = phenotype))
```



Geoms

- It's possible to add multiple geoms to the same plot

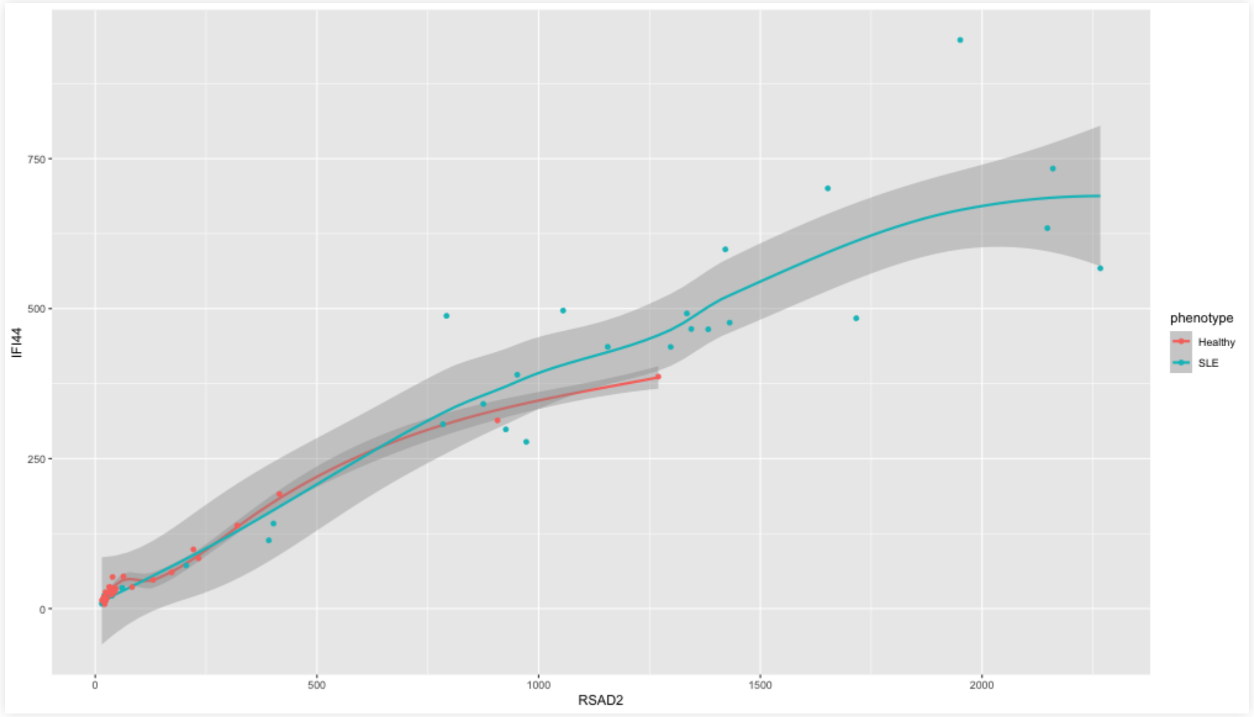
```
ggplot(genes) +
  geom_smooth(aes(x = RSAD2, y = IFI44, color = phenotype)) +
  geom_point(aes(x = RSAD2, y = IFI44, color = phenotype))
```



Geoms

- To assign the same aesthetics to all geoms, pass the aesthetics to the `ggplot` function directly instead of to each geom individually

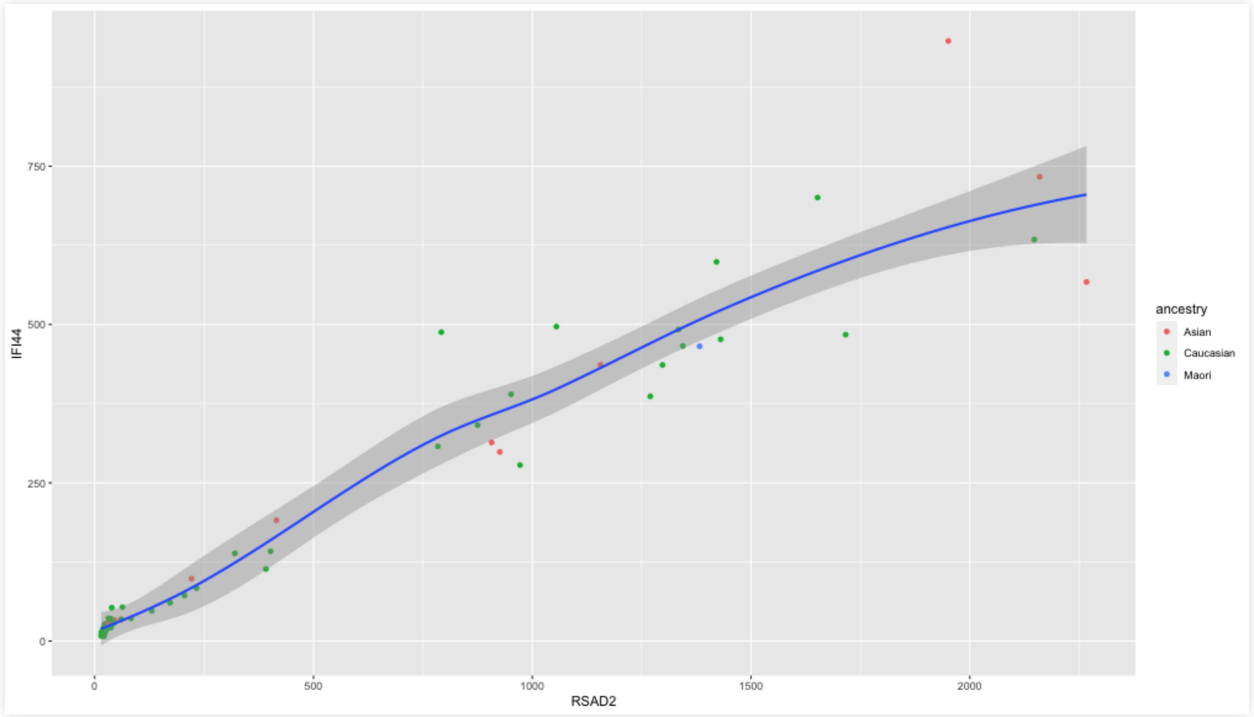
```
ggplot(genes, aes(x = RSAD2, y = IFI44, color = phenotype)) +
  geom_smooth() +
  geom_point()
```



Geoms

- You can also use different mappings in different geoms

```
ggplot(genes, mapping = aes(x = RSAD2, y = IFI44)) +
  geom_point(aes(color = ancestry)) +
  geom_smooth()
```



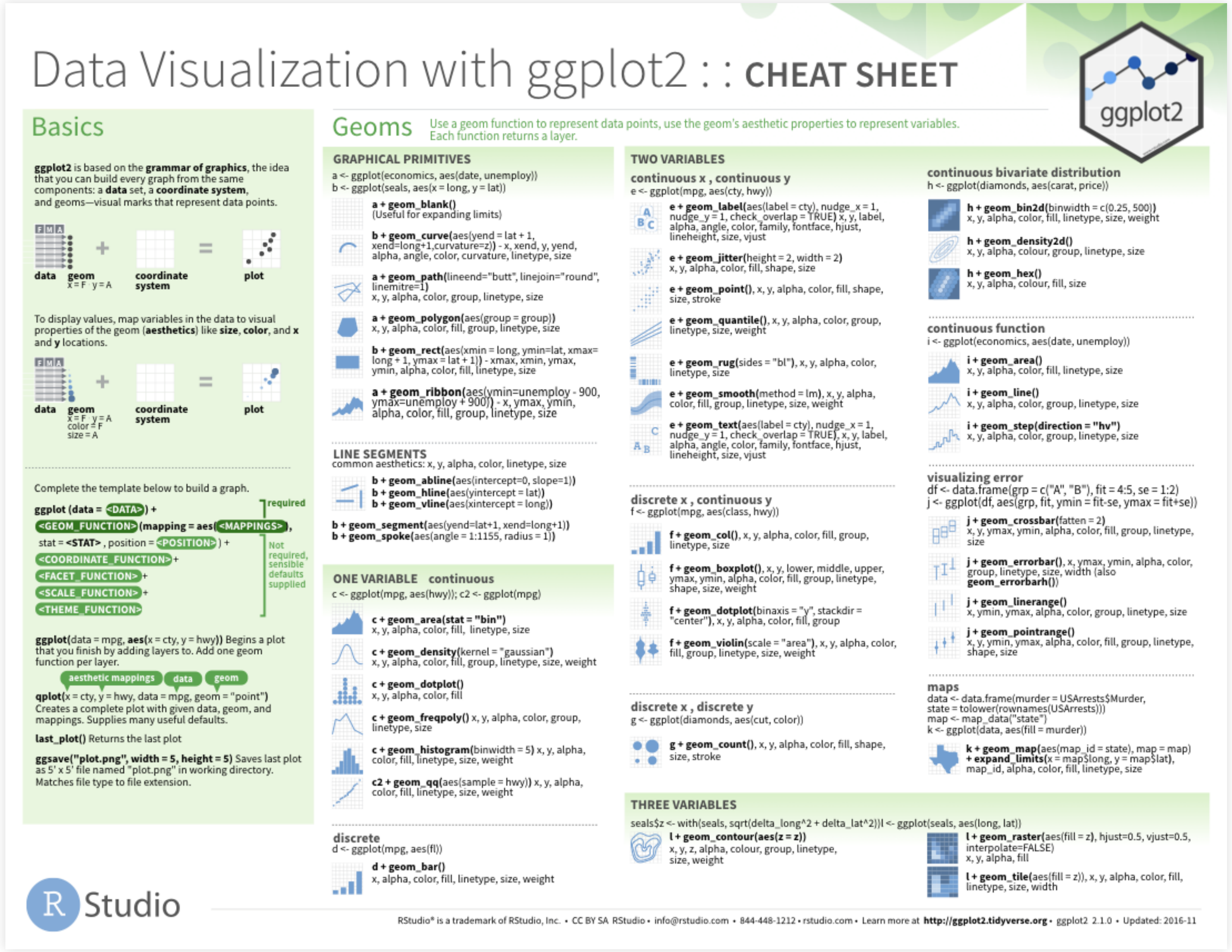
Exercise

Use google or other resources to figure out how to receate this plot in R:

Learning More

From **R for Data Science**:

> If you want to learn more about the mechanics of ggplot2, I'd highly recommend grabbing a copy of the ggplot2 book: https://amzn.com/331924275X. It's been recently updated, so it includes dplyr and tidyr code, and has much more space to explore all the facets of visualisation. Unfortunately the book isn't generally available for free, but if you have a connection to a university you can probably get an electronic version for free through SpringerLink.
>
> Another useful resource is the R Graphics Cookbook by Winston Chang. Much of the contents are available online at http://www.cookbook-r.com/Graphs/.
>
> I also recommend Graphical Data Analysis with R, by Antony Unwin. This is a book-length treatment similar to the material covered in this chapter, but has the space to go into much greater depth.



Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

F M A
data
geom
x = F · y = A
+
coordinate system
=
plot

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

```
ggplot (data = <DATA>) +                                  ] required
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
 stat = <STAT> , position = <POSITION>) +                 ] Not
<COORDINATE_FUNCTION> +                                     required,
<FACET_FUNCTION> +                                          sensible
<SCALE_FUNCTION> +                                          defaults
<THEME_FUNCTION>                                            supplied
```

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings · data · geom

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1,curvature=z)) - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1)
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group))
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1))
b + geom_hline(aes(yintercept = lat))
b + geom_vline(aes(xintercept = long))

b + geom_segment(aes(yend=lat+1, xend=long+1))
b + geom_spoke(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy)) x, y, alpha, color, fill, linetype, size, weight

discrete

d <- ggplot(mpg, aes(fl))

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

e <- ggplot(mpg, aes(cty, hwy))

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2)
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"), x, y, alpha, color, linetype, size

e + geom_smooth(method = lm), x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE), x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

f <- ggplot(mpg, aes(class, hwy))

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"), x, y, alpha, color, fill, group

f + geom_violin(scale = "area"), x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

g <- ggplot(diamonds, aes(cut, color))

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

h <- ggplot(diamonds, aes(carat, price))

h + geom_bin2d(binwidth = c(0.25, 500))
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

i <- ggplot(economics, aes(date, unemploy))

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

j + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))

k + geom_map(aes(map_id = state), map = map)
+ expand_limits(x = map$long, y = map$lat),
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))l <- ggplot(seals, aes(long, lat))

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)
x, y, alpha, fill

l + geom_tile(aes(fill = z)), x, y, alpha, color, fill, linetype, size, width

RStudio