

# Data Visualization Using R & ggplot2

---

Naupaka Zimmerman (@naupakaz) and Andrew Tredennick (@ATredennick)  
(and Karthik Ram (@\_inundata))

August 6, 2017

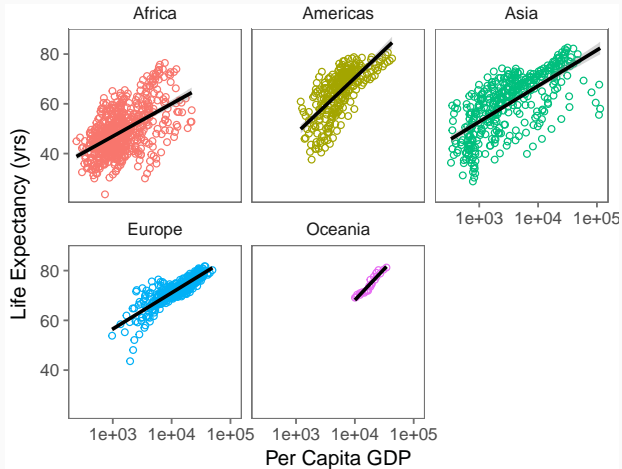
# Some housekeeping

Install some packages

```
install.packages("ggplot2", dependencies = TRUE)  
install.packages("ggthemes")  
install.packages("tidyr")  
install.packages("dplyr")
```

# Why ggplot?

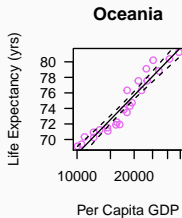
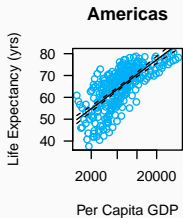
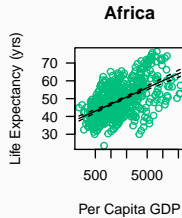
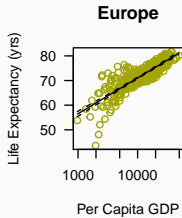
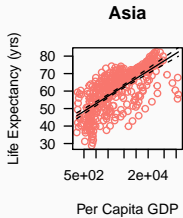
---



# 1 minute

```
library(ggplot2)
library(gapminder)
library(ggthemes)

ggplot(gapminder, aes(x = gdpPercap, y = lifeExp)) +
  geom_point(shape = 1, aes(color = continent)) +
  stat_smooth(method = "lm", size = 1, color = "black") +
  scale_x_log10() +
  xlab("Per Capita GDP") +
  ylab("Life Expectancy (yrs)") +
  facet_wrap(~continent) +
  theme_few() +
  guides(color = FALSE)
```



```
library(scales)
library(gapminder)
gapminder <- as.data.frame(gapminder)
conts <- unique(gapminder[, "continent"])
cols <- scales::hue_pal()(length(conts))
par(mfrow = c(2,3))
counter <- 1
for (i in conts) {
  plot(gapminder[which(gapminder$continent == i), "gdpPercap"],
       gapminder[which(gapminder$continent == i), "lifeExp"], col = cols[counter],
       xlab = "Per Capita GDP", ylab = "Life Expectancy (yrs)",
       main = i, las = 1, log = "x")
  fit <- lm(gapminder[which(gapminder$continent == i), "lifeExp"] ~
```

But wait, there's more...

```
log(gapminder[which(gapminder$continent == i), "gdpPercap"]))  
  pred <- predict(fit, interval = "confidence")  
  lines(sort(gapminder[which(gapminder$continent == i), "gdpPercap"]),  
        lines(sort(gapminder[which(gapminder$continent == i), "gdpPercap"]),  
        lines(sort(gapminder[which(gapminder$continent == i), "gdpPercap"]),  
        counter <- counter + 1  
}
```



# Why ggplot?

---

# Why ggplot?

- More elegant and compact code than with base graphics
- More aesthetically pleasing defaults than lattice
- Very powerful for exploratory data analysis

# Why ggplot?

- `gg` is for **grammar of graphics** (term by Lee Wilkinson)
- A set of terms that defines the basic components of a plot
- Used to produce figures using coherent, consistent syntax

## Why ggplot2?

- Supports a continuum of expertise
- Easy to get started, plenty of power for complex figures

# The Grammar

---

### Data

- Must be a data frame (`data.frame()`, `as.data.frame()`)
- Gets pulled into the `ggplot()` object

# A quick example

## The iris dataset

```
head(iris)
```

##	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
## 1	5.1	3.5	1.4	0.2	setosa
## 2	4.9	3.0	1.4	0.2	setosa
## 3	4.7	3.2	1.3	0.2	setosa
## 4	4.6	3.1	1.5	0.2	setosa
## 5	5.0	3.6	1.4	0.2	setosa
## 6	5.4	3.9	1.7	0.4	setosa

Helps your data play nice with ggplot



```
iris[1:2, ]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1           3.5           1.4           0.2  setosa
## 2           4.9           3.0           1.4           0.2  setosa
```

```
library(tidyr)
df <- gather(iris, key = flower_attribute,
              value = measurement, -Species)
df[1:2, ]
```

```
##   Species flower_attribute measurement
## 1  setosa      Sepal.Length           5.1
## 2  setosa      Sepal.Length           4.9
```

# Aesthetics

---

## aesthetics

- How your data are represented visually
- a.k.a. mapping
- which data on the x
- which data on the y
- but also: **color**, **size**, shape, transparency

## An example

```
myplot <- ggplot(data = iris, aes(x = Sepal.Length,  
                                  y = Sepal.Width))  
  
summary(myplot)
```

```
## data: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width,  
##   Species [150x5]  
## mapping:  x = Sepal.Length, y = Sepal.Width  
## faceting: <ggproto object: Class FacetNull, Facet>  
##   compute_layout: function  
##   draw_back: function  
##   draw_front: function  
##   draw_labels: function  
##   draw_panels: function  
##   finish_data: function  
##   init_scales: function  
##   map: function  
##   map_data: function  
##   params: list
```

# Geometries

---

### geometry

- The geometric objects in the plot
- points, lines, polygons, etc.
- functions: `geom_point()`, `geom_bar()`, `geom_line()`

# Basic structure

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point()
```

*# Equivalently...*

```
myplot <- ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width))  
myplot + geom_point()
```

- Specify the data and variables inside the `ggplot()` function.
- Anything else that goes in here becomes a *global* setting.
- Then add layers: geometric objects, statistical models, and facets.

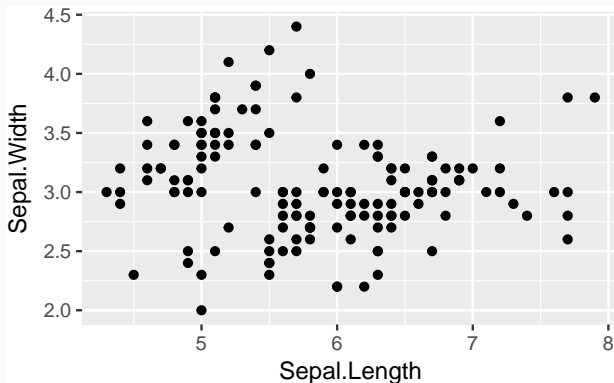
# Don't be tempted!

- **NEVER** use `qplot()` - short for quick plot.
- You'll end up unlearning and relearning a good bit.



## Let's try an example

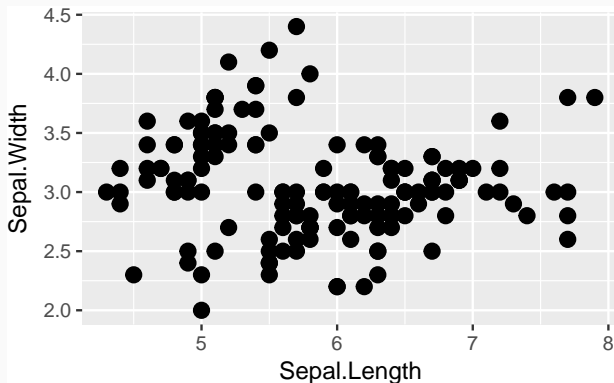
```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point()
```



# Changing the **aesthetics** of a geom

## Increase the size of points

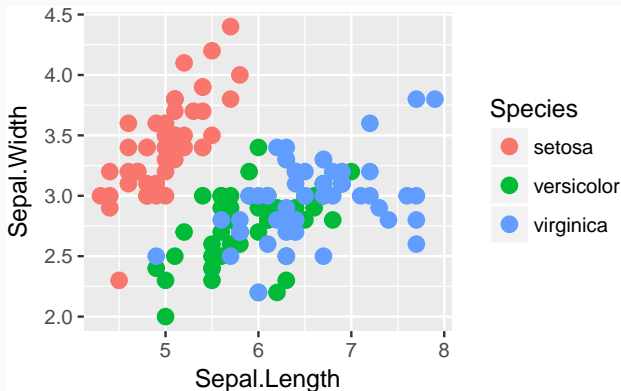
```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point(size = 3)
```



# Changing the **aesthetics** of a geom

## Color by species (*mapping!*)

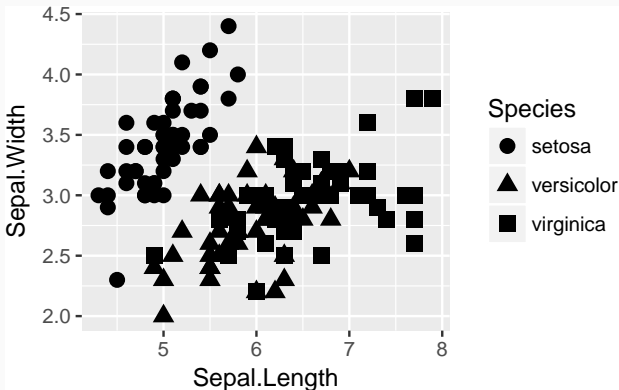
```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width,  
                        color = Species)) +  
  geom_point(size = 3)
```



# Changing the **aesthetics** of a geom

## Differentiate points by shape

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point(size = 3, aes(shape = Species))
```



## Exercise 1

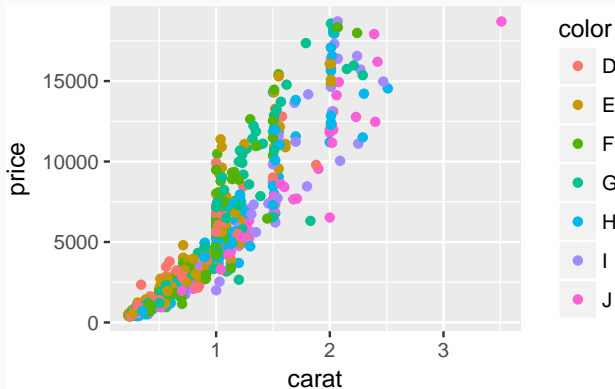
---

# Exercise 1

*# Make a small sample of the diamonds dataset*

```
d2 <- diamonds[sample(x = 1:nrow(diamonds), size = 1000), ]
```

The make this plot:



## Stats

---

### stats

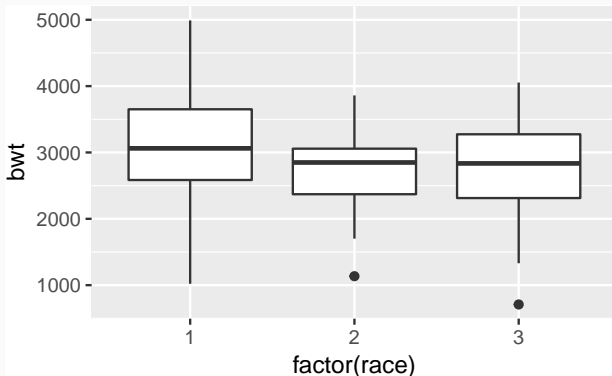
- Statistical transformations and data summary
- All geoms have associated default stats, and vice versa
- e.g. binning (the stat) for a histogram or fitting a linear model



## Built-in stat example

See `?geom_boxplot` for options

```
library(MASS)
ggplot(birthwt, aes(factor(race), bwt)) +
  geom_boxplot()
```



## Built-in stat example

```
myplot <- ggplot(birthwt, aes(factor(race), bwt)) +  
  geom_boxplot()  
summary(myplot)
```

```
## data: low, age, lwt, race, smoke, ptl, ht, ui, ftv, bwt [189x10]  
## mapping: x = factor(race), y = bwt  
## faceting: <ggproto object: Class FacetNull, Facet>  
##   compute_layout: function  
##   draw_back: function  
##   draw_front: function  
##   draw_labels: function  
##   draw_panels: function  
##   finish_data: function  
##   init_scales: function  
##   map: function  
##   map_data: function  
##   params: list  
##   render_back: function
```

# Facets

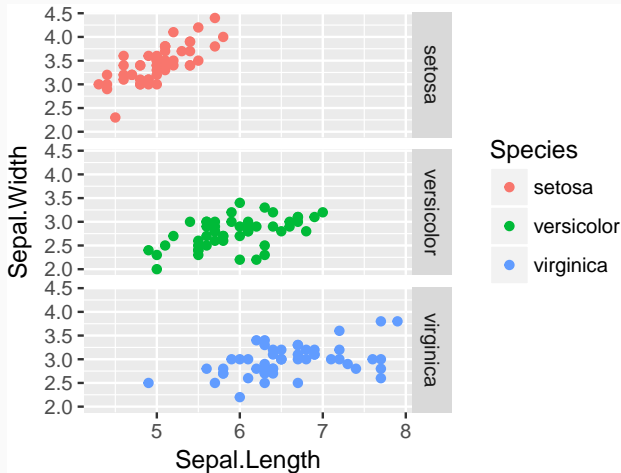
---

## facets

- Subsetting data to make lattice plots
- Really powerful
- I use in almost every publication

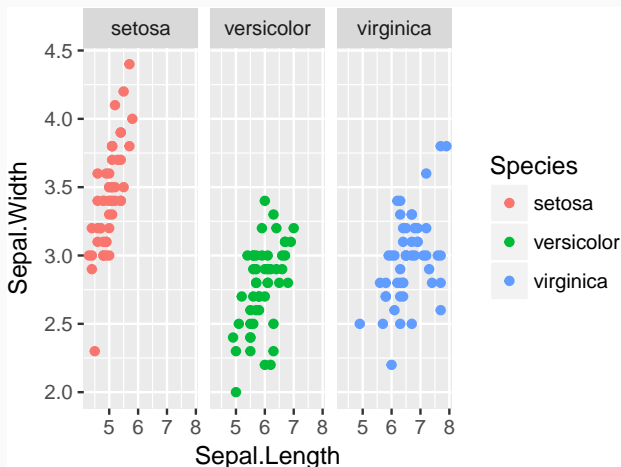
## Faceting: single column, multiple rows

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point() +  
  facet_grid(Species ~ .)
```



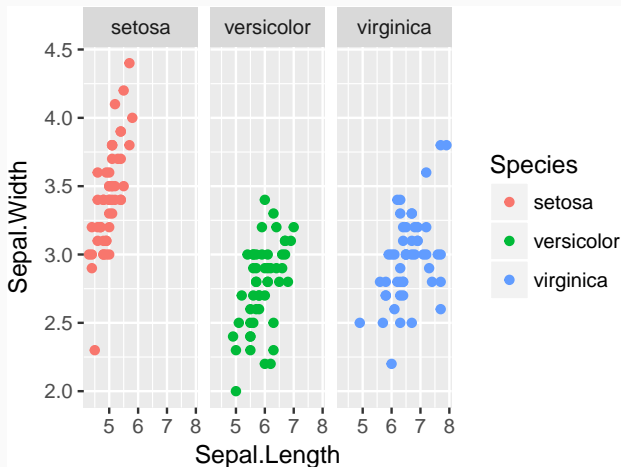
## Faceting: single row, multiple columns

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point() +  
  facet_grid(. ~ Species)
```



## Or, just wrap them

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point() +  
  facet_wrap(~ Species) # notice lack of .
```



# Scales

---

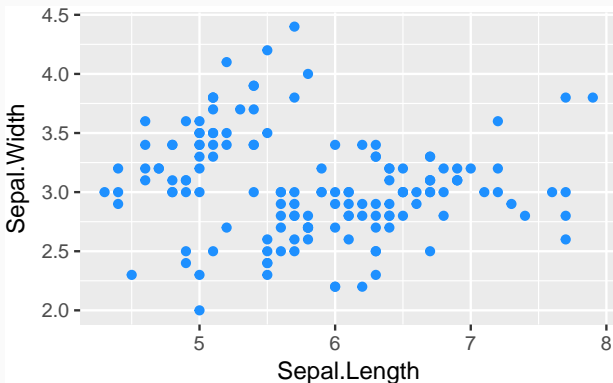


## scales

- Control the *mapping* from data to aesthetics
- Often used for adjusting color mapping (i.e., setting colors manually)

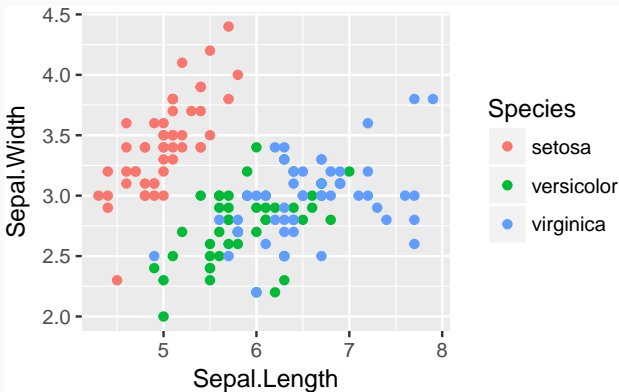
## Setting color

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(color = "dodgerblue")
```



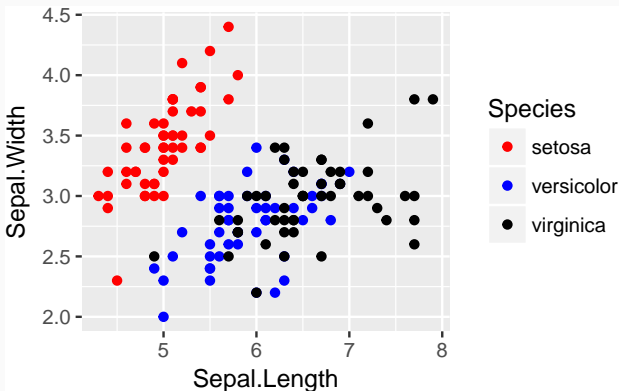
# Mapping color

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(aes(color = Species))
```



# Map custom color

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(aes(color = Species)) +  
  scale_color_manual(values = c("red", "blue", "black"))
```

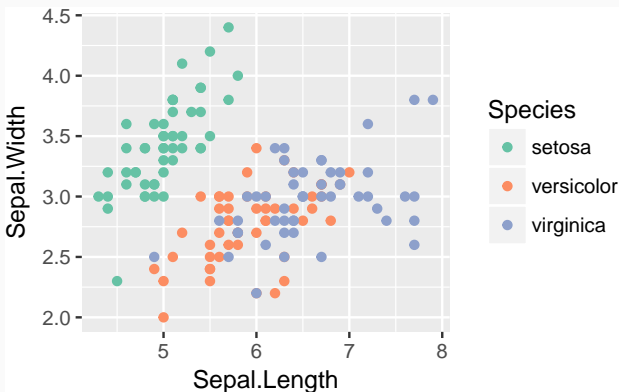


```
library("RColorBrewer")
display.brewer.all()
```



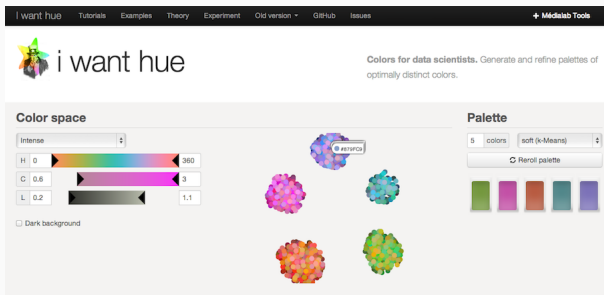
# Map custom color with brewer

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(aes(color = Species)) +  
  scale_color_brewer(palette = "Set2")
```



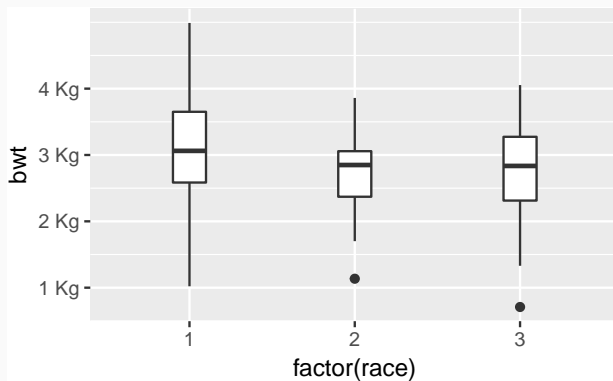
# Refer to a color chart for beautiful visualizations

<http://tools.medialab.sciences-po.fr/iwanthue/>



# Altering a continuous scale

```
library(MASS)
ggplot(birthwt, aes(x = factor(race), y = bwt)) +
  geom_boxplot(width = 0.2) +
  scale_y_continuous(labels = (paste0(1:4, " Kg")),
                     breaks = seq(1000, 4000, by = 1000))
```





## Some common scales

```
scale_fill_discrete(); scale_colour_discrete()  
scale_fill_hue(); scale_color_hue()  
scale_fill_manual(); scale_color_manual()  
scale_fill_brewer(); scale_color_brewer()  
scale_linetype(); scale_shape_manual()
```

# Coordinates

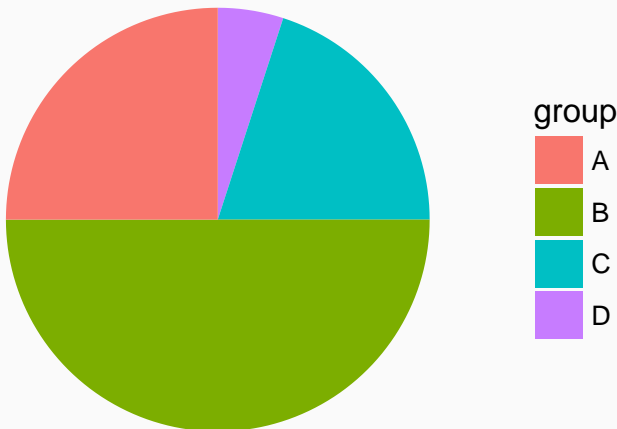
---

## coordinates

- Not going to cover this in detail
- e.g. polar coordinate plots

# Don't try this at home... please!

```
ggplot(mydata, aes(x = "", y = proportion, fill = group)) +  
  geom_bar(stat = "identity", width = 1) +  
  coord_polar("y", start=0) +  
  theme_void()
```



## Putting it all together with more examples

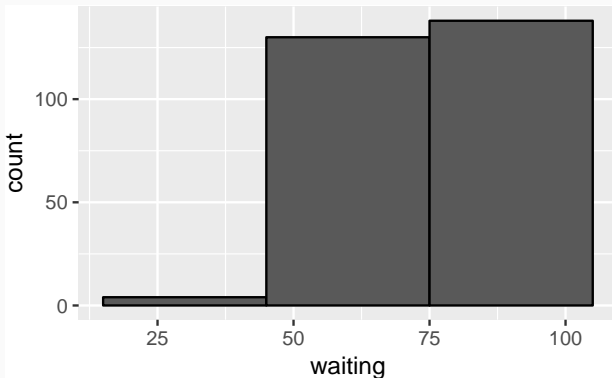
---

# Histograms

---

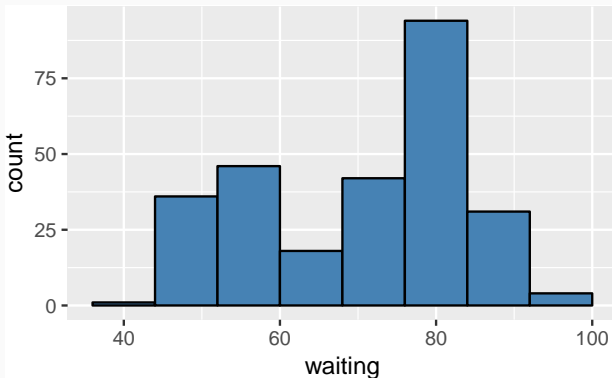
# Simple histogram

```
ggplot(faithful, aes(x = waiting)) +  
  geom_histogram(binwidth = 30, color = "black")
```



# Simple histogram

```
ggplot(faithful, aes(x = waiting)) +  
  geom_histogram(binwidth = 8, color = "black", fill = "steelblue")
```



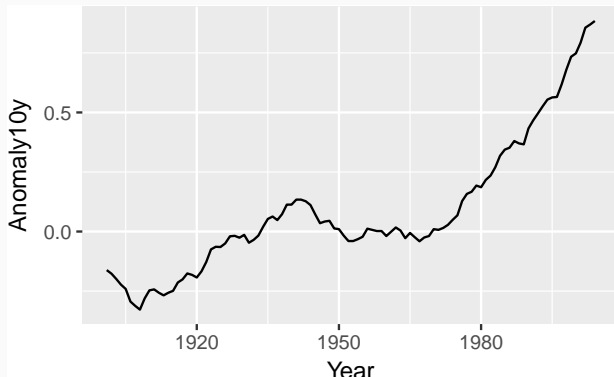


## Line plots

---

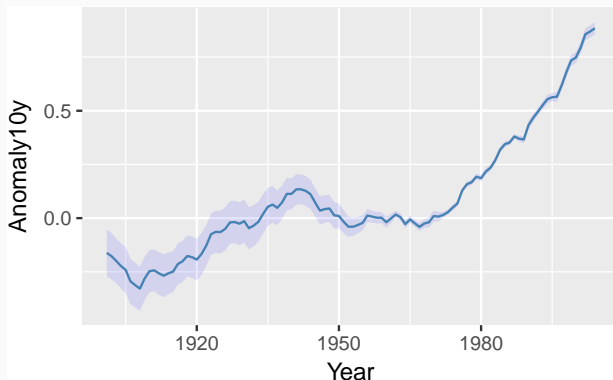
# Line plot

```
climate <- read.csv("../data/climate.csv", header = TRUE)
ggplot(climate, aes(Year, Anomaly10y)) +
  geom_line()
```



# Line with confidence interval

```
ggplot(climate, aes(x = Year, y = Anomaly10y)) +  
  geom_ribbon(aes(ymin = Anomaly10y - Unc10y,  
                ymax = Anomaly10y + Unc10y),  
            fill = "blue", alpha = 0.1) +  
  geom_line(color = "steelblue")
```

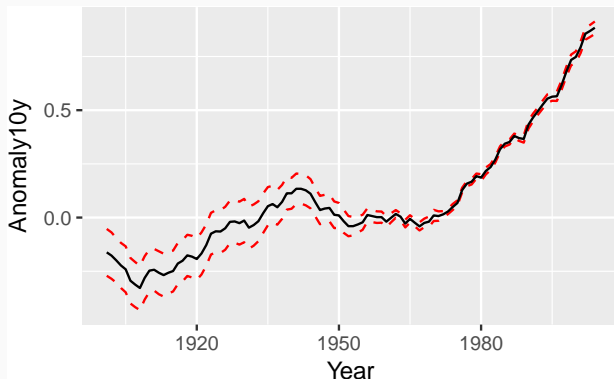


## Exercise 2

---

## Exercise 2

- Modify the last plot and change the ribbon to three separate lines, like below:

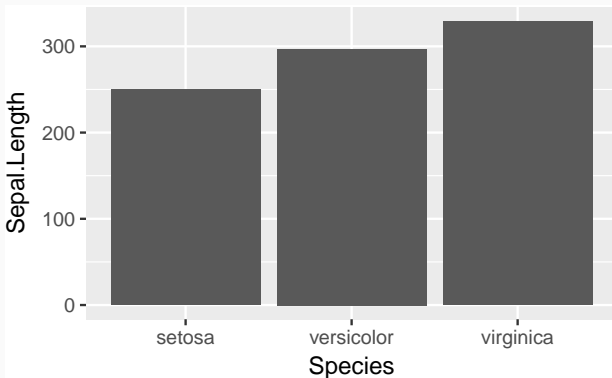


## Bar plots

---

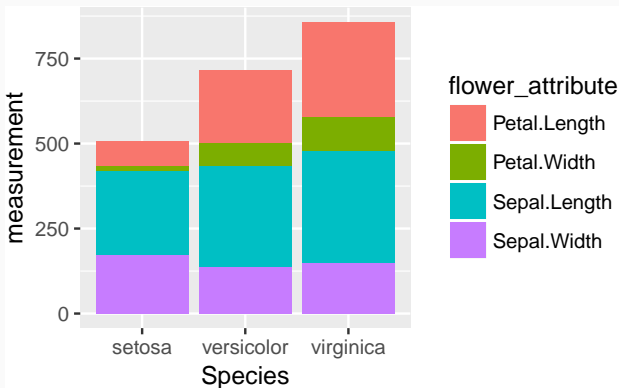
## geom\_bar() can be tricky

```
ggplot(iris, aes(Species, Sepal.Length)) +  
  geom_bar(stat = "identity")
```



## geom\_bar() can be tricky

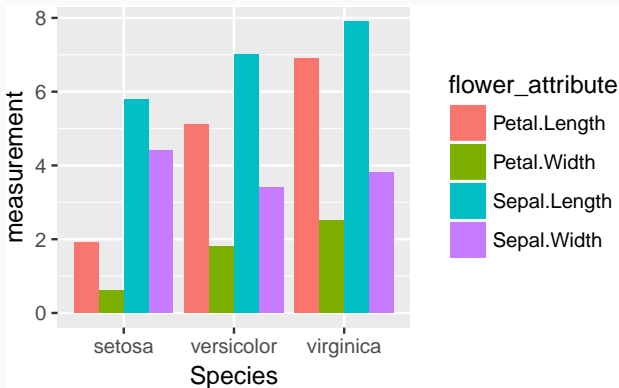
```
ggplot(df, aes(Species, measurement, fill = flower_attribute)) +  
  geom_bar(stat = "identity")
```





## geom\_bar() can be tricky

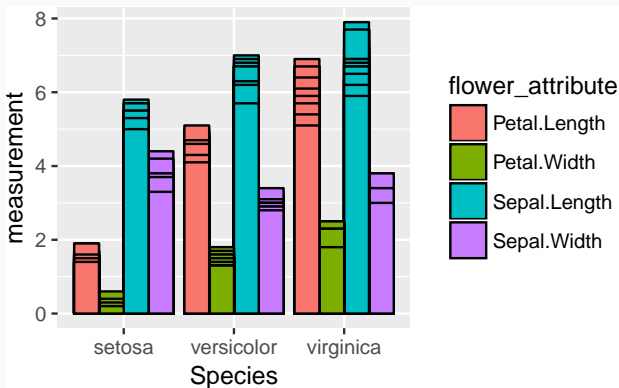
```
ggplot(df, aes(Species, measurement, fill = flower_attribute)) +  
  geom_bar(stat = "identity", position = "dodge")
```



What's up with the y-axis?

## geom\_bar() can be tricky

```
ggplot(df, aes(Species, measurement, fill = flower_attribute)) +  
  geom_bar(stat = "identity", position = "dodge", color = "black")
```



## Summarize the data first, usually

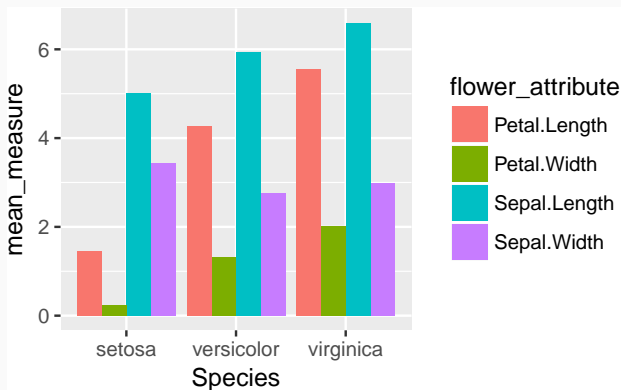
```
df_agg <- df %>%  
  group_by(Species, flower_attribute) %>%  
  summarise(mean_measure = mean(measurement))
```

```
head(df_agg)
```

```
## Source: local data frame [6 x 3]  
## Groups: Species [2]  
##  
## # A tibble: 6 x 3  
##   Species flower_attribute mean_measure  
##   <fctr>      <chr>          <dbl>  
## 1  setosa    Petal.Length      1.462  
## 2  setosa    Petal.Width        0.246  
## 3  setosa    Sepal.Length      5.006  
## 4  setosa    Sepal.Width        3.428  
## 5 versicolor Petal.Length      4.260  
## 6 versicolor Petal.Width        1.326
```

## Then plot the summary with `geom_col`

```
ggplot(df_agg, aes(x = Species, y = mean_measure, fill=flower_attribute))  
  geom_col(position = "dodge")
```

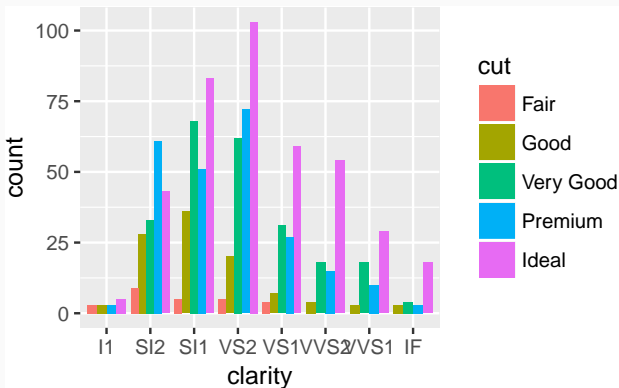


## Exercises 3, 4, & 5

---

## Exercise 3

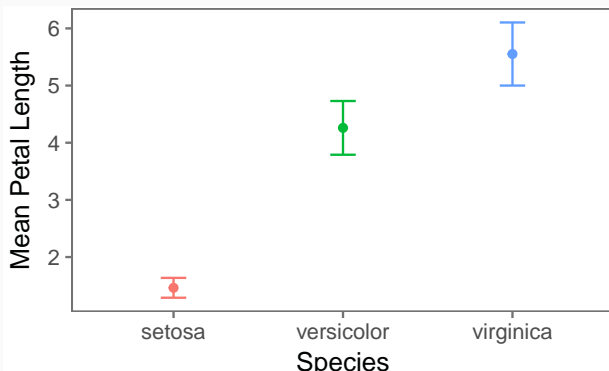
Using the d2 dataset you created earlier, make this plot:



## Exercise 4

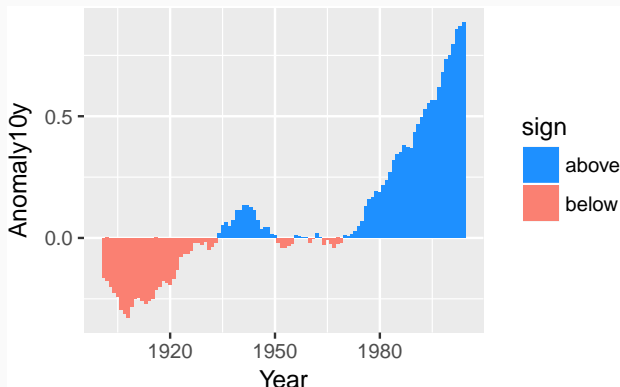
Use dplyr to summarize the iris data, then make the plot below.

```
iris_agg <- iris %>%  
  group_by(Species) %>%  
  summarise(mean_petal_len = mean(Petal.Length),  
            sd_petal_len = sd(Petal.Length))  
# ?geom_errorbar()
```



## Exercise 5

- Using the `climate` dataset, create a new variable called `sign`. Make it categorical (above/below) based on the sign of `Anomaly10y`.
- Plot a bar plot and use `sign` variable as the fill.



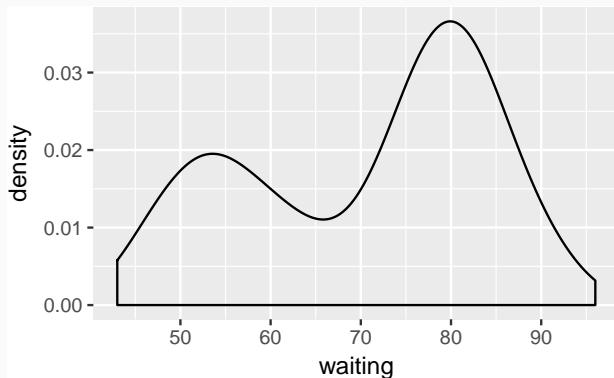


## Density plots

---

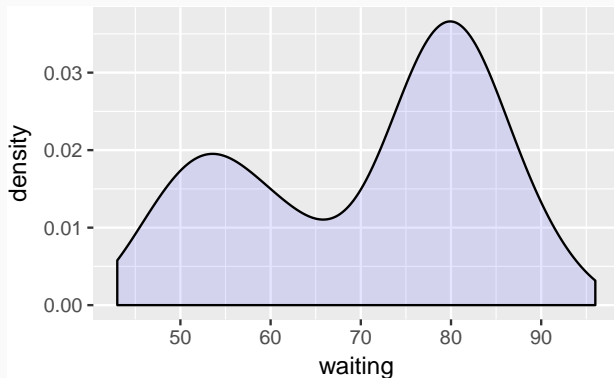
# Density plot

```
ggplot(faithful, aes(waiting)) +  
  geom_density()
```



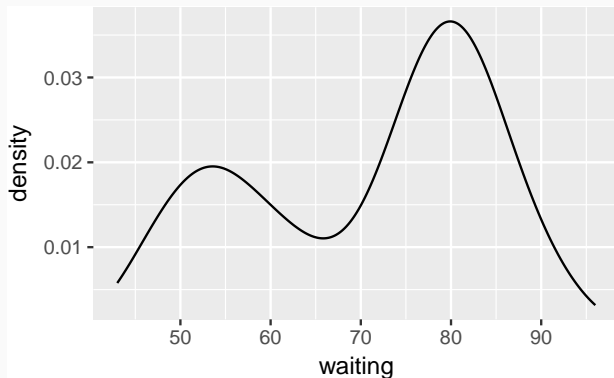
# Density plot

```
ggplot(faithful, aes(waiting)) +  
  geom_density(fill = "blue", alpha = 0.1)
```



# Density plot

```
ggplot(faithful, aes(waiting)) +  
  geom_line(stat = "density")
```

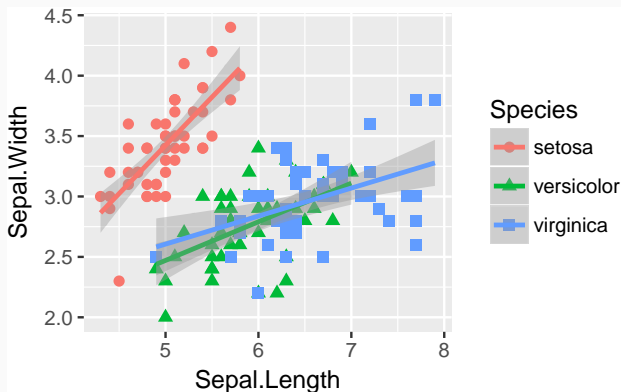


## Adding smoothers

---

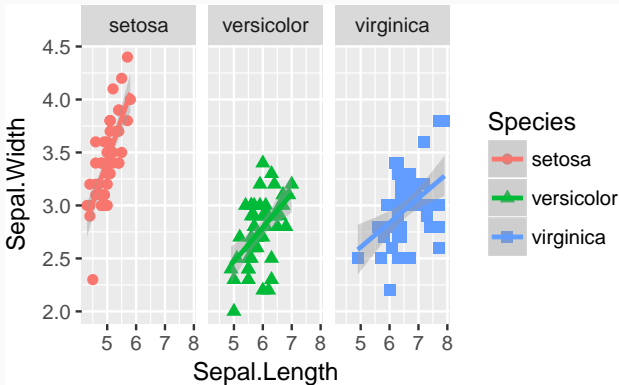
# Linear fits

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(aes(shape = Species), size = 2) +  
  geom_smooth(method = "lm")
```



# Linear fits, faceted

```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(aes(shape = Species), size = 2) +  
  geom_smooth(method = "lm") +  
  facet_grid(. ~ Species)
```



# Themes!

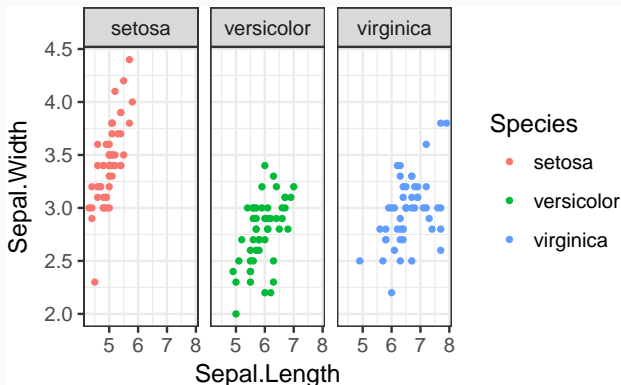
---



- Everything can be customized using `theme()` settings
- <http://ggplot2.tidyverse.org/reference/theme.html>

# Out-of-the-box themes

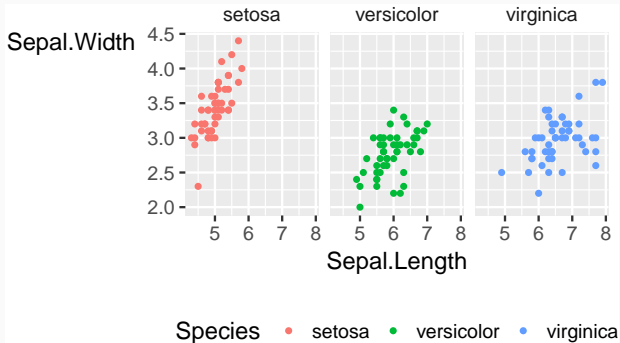
```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(size = 1.2, shape = 16) +  
  facet_wrap( ~ Species) +  
  theme_bw()
```



## A 'themed' plot

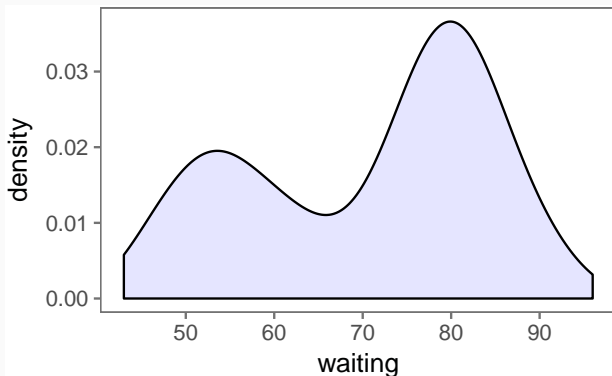
```
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(size = 1.2, shape = 16) +  
  facet_wrap( ~ Species) +  
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +  
  geom_point(size = 1.2, shape = 16) +  
  facet_wrap( ~ Species) +  
  theme(legend.key = element_rect(fill = NA),  
        legend.position = "bottom",  
        strip.background = element_rect(fill = NA),  
        axis.title.y = element_text(angle = 0))
```

## A 'themed' plot



## ggthemes library

```
library(ggthemes)
ggplot(faithful, aes(waiting)) +
  geom_density(fill = "blue", alpha = 0.1) +
  theme_few()
```

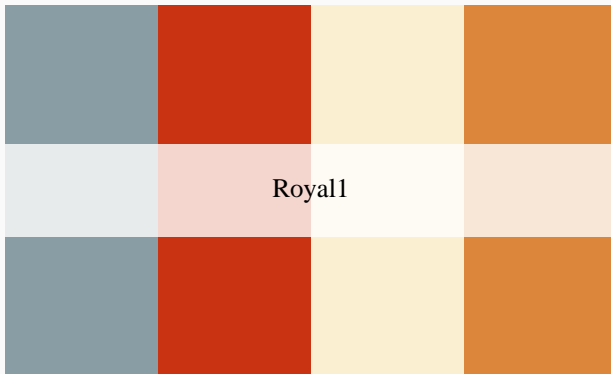


# What about Wes Anderson?



# There's a theme for that!

```
# install.packages('wesanderson')  
library("wesanderson")  
# display a palette  
wes_palette("Roya11")
```



# Save your beautiful plot

---



- If the plot is on the screen

```
ggsave('~ /path/to/figure/filename.png')
```

- If your plot is assigned to a named object

```
ggsave(plot1, '~ /path/to/figure/filename.png')
```

- Specify size

```
ggsave(file = '~ /path/figure.png', width = 4, height = 3, units = "in")
```

- And format

```
ggsave(file = "/path/to/figure/filename.eps")
```

```
ggsave(file = "/path/to/figure/filename.jpg")
```

```
ggsave(file = "/path/to/figure/filename.pdf")
```

# Further help

- We've only scratched the surface
- Practice
- Read: <http://ggplot2.tidyverse.org/index.html>
- Work together

