Data Visualization with ggplot2

Medical Biophysics Tech Talks

Antoine Beauchamp
antoine.beauchamp@mail.utoronto.ca
November 9th, 2018

Outline

- The Tidyverse
- The Grammar of Graphics
  - The Data Layer
  - The Aesthetics Layer
  - The Geometries Layer
  - The Facets Layer
  - The Statistics Layer
  - The Coordinates Layer
  - The Themes Layer
- Summary
- Resources

The Tidyverse

- A suite of R packages designed for data science
- Initially developed by Hadley Wickham

The Tidyverse

- A suite of R packages designed for data science
- Initially developed by Hadley Wickham

- Core packages:
  - `readr` ------- Rapid methods for reading rectangular data
  - `tibble` ------ Wrapper to improve data frame performance
  - `tidyr` ------- Functions to tidy your data
  - `dplyr` ------- Grammar of functions for data manipulation
  - `stringr` ---- Functions for string manipulation
  - `purrr` ------- Improved functional programming tools
  - `ggplot2` ---- Powerful data visualization framework

The Tidyverse

- A suite of R packages designed for data science
- Initially developed by Hadley Wickham

- Core packages:
  - `readr` ------- Rapid methods for reading rectangular data
  - `tibble` ------ Wrapper to improve data frame performance
  - `tidyr` ------- Functions to tidy your data
  - `dplyr` ------- Grammar of functions for data manipulation
  - `stringr` ---- Functions for string manipulation
  - `purrr` ------- Improved functional programming tools
  - `ggplot2` ---- Powerful data visualization framework

The Tidyverse

The Tidyverse can be installed all at once

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

Or package by package:

```
install.packages("ggplot2")
```

Packages are loaded in the standard way:

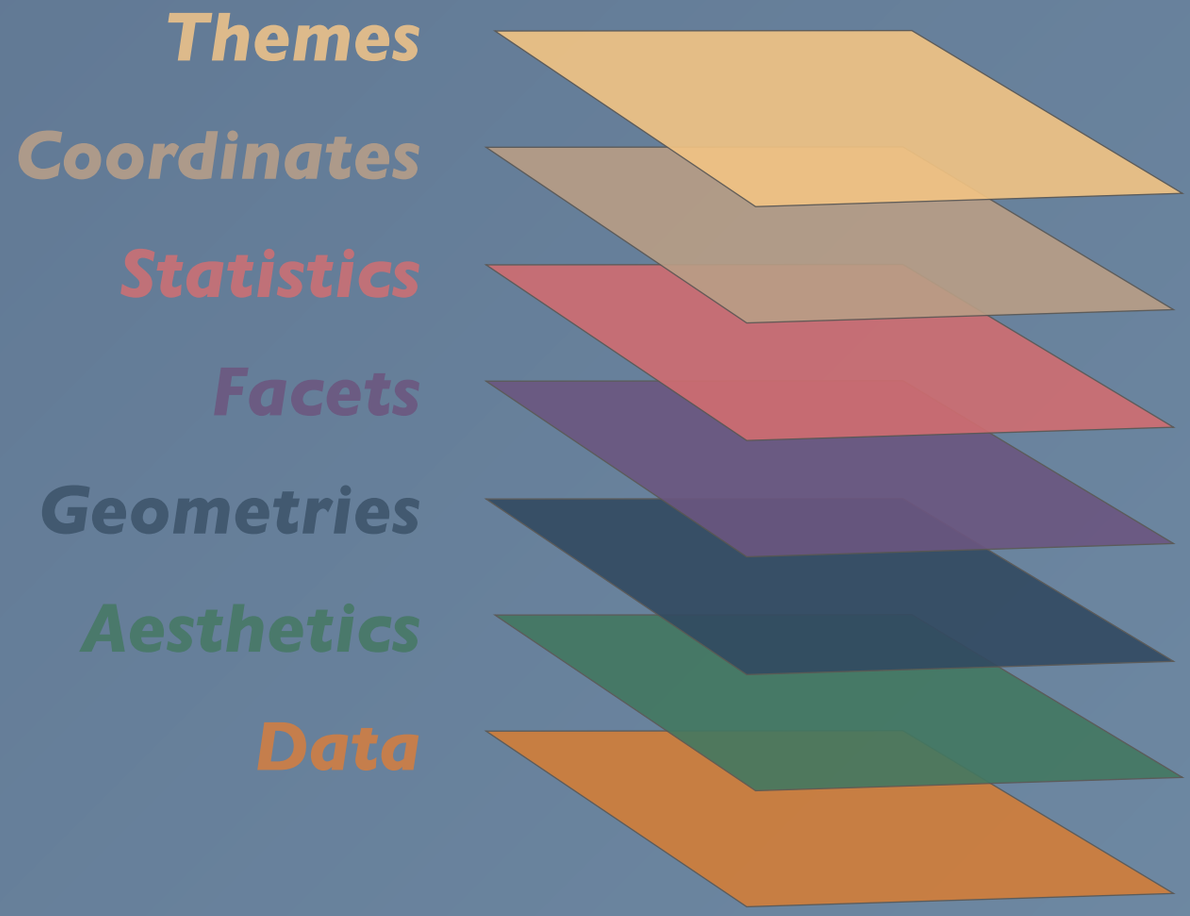
```
library(tidyverse)
library(ggplot2)
```

The Grammar of Graphics

- The "gg" in ggplot2
- Plotting framework developed by Leland Wilkinson:
  - Graphics are made up of layers of "grammatical" elements
  - Meaningful plots are built around aesthetic mappings

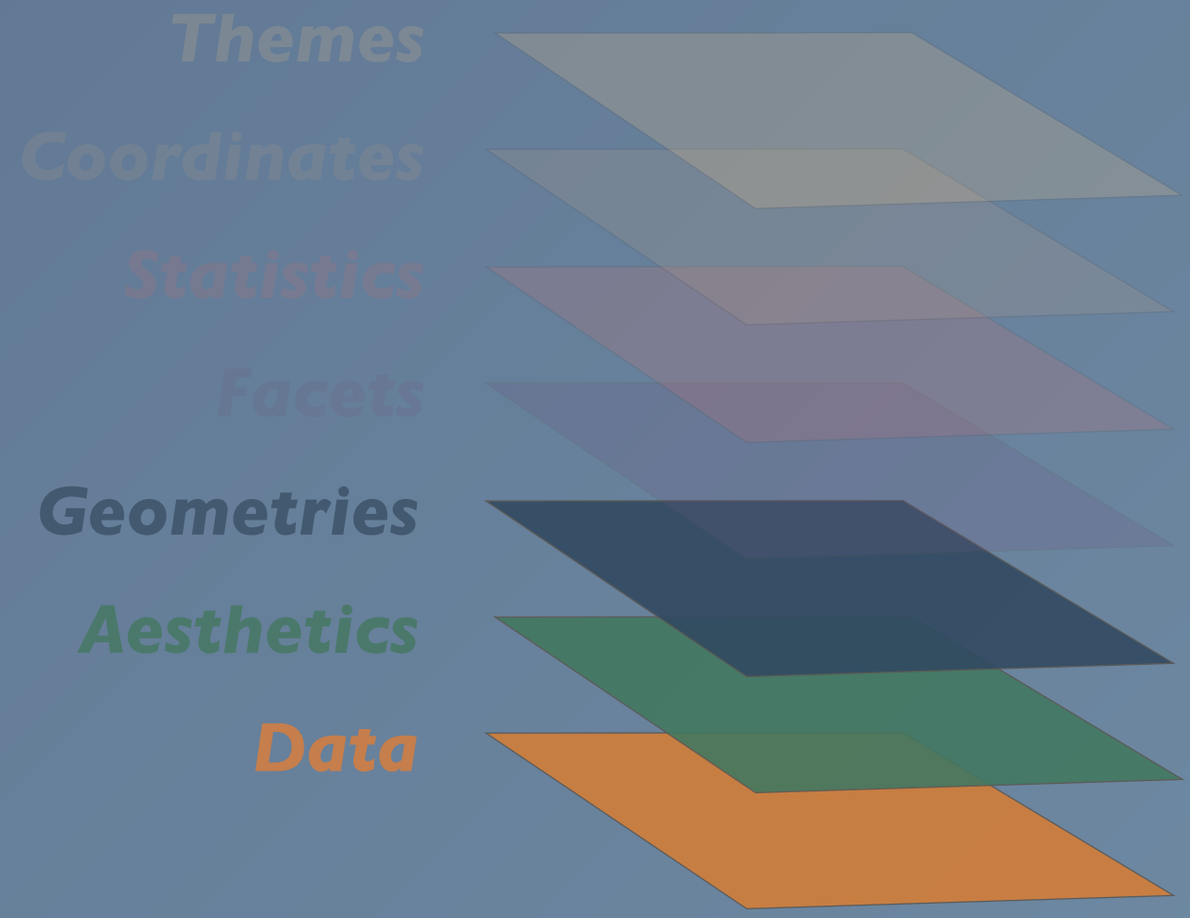
The Grammar of Graphics

- The "gg" in ggplot2
- Plotting framework developed by Leland Wilkinson:
  - Graphics are made up of layers of "grammatical" elements
    - Meaningful plots are built around aesthetic mappings
- 7 grammatical elements/layers
- 3 essential layers to make a plot
- In ggplot we create a graphic by stacking grammatical elements



The Grammar of Graphics

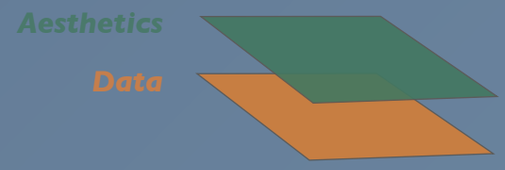
- The "gg" in ggplot2
- Plotting framework developed by Leland Wilkinson:
  - Graphics are made up of layers of "grammatical" elements
    - Meaningful plots are built around aesthetic mappings
- 7 grammatical elements/layers
- 3 essential layers to make a plot
- In ggplot we create a graphic by stacking grammatical elements



The Grammar of Graphics

Data and Aesthetics

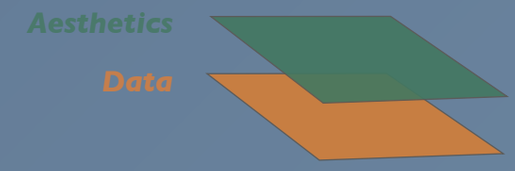
```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))
```



The Grammar of Graphics

Data and Aesthetics

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



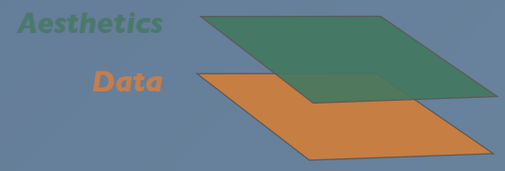
Data

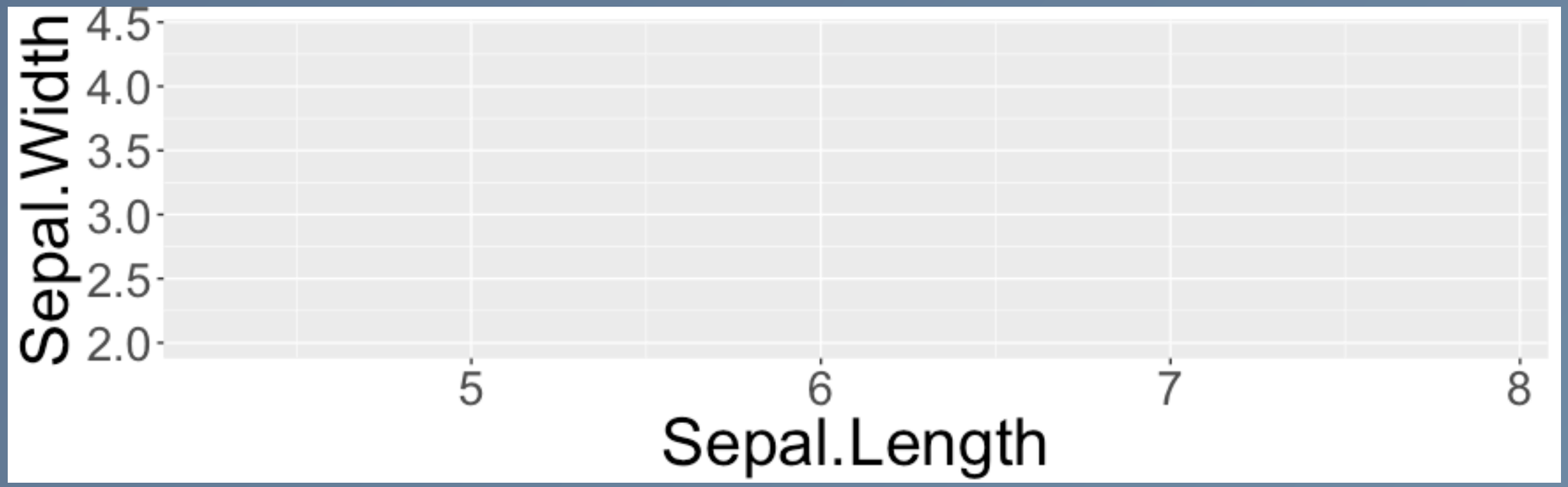
Aesthetics

The Grammar of Graphics

Data and Aesthetics

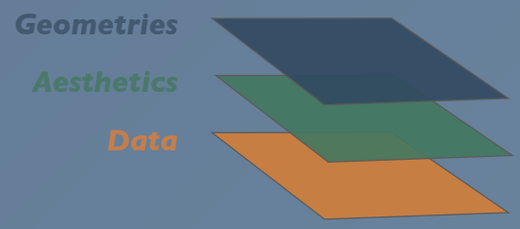
```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))
```



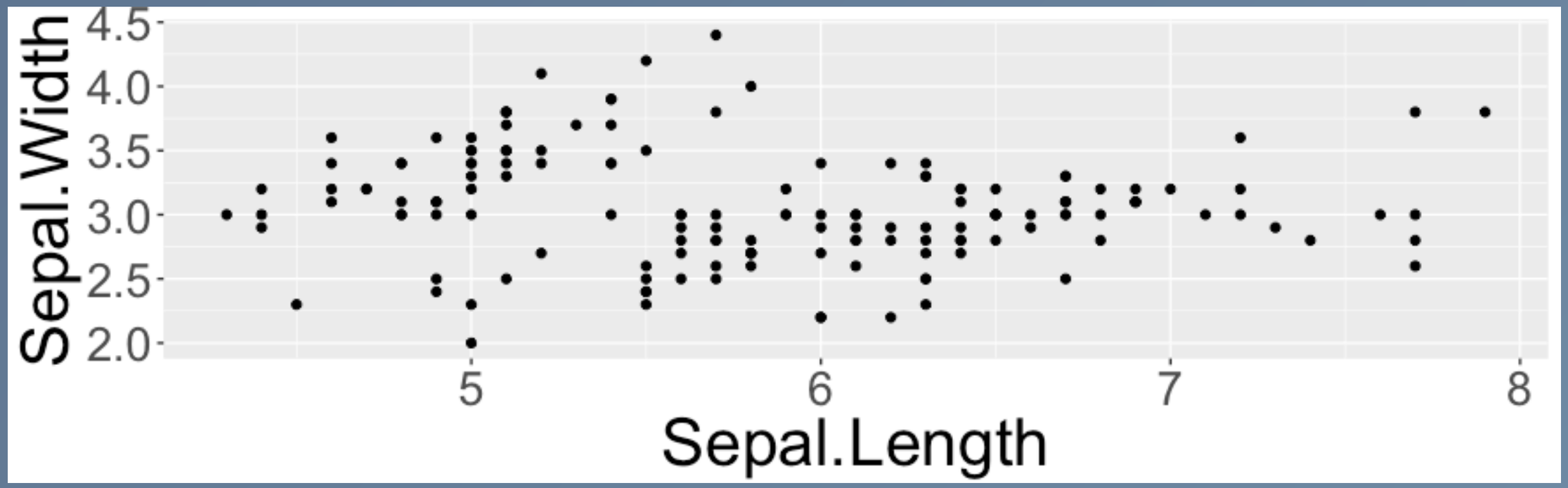


The Grammar of Graphics

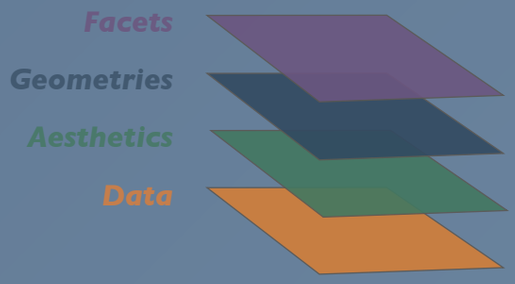
Geometries




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

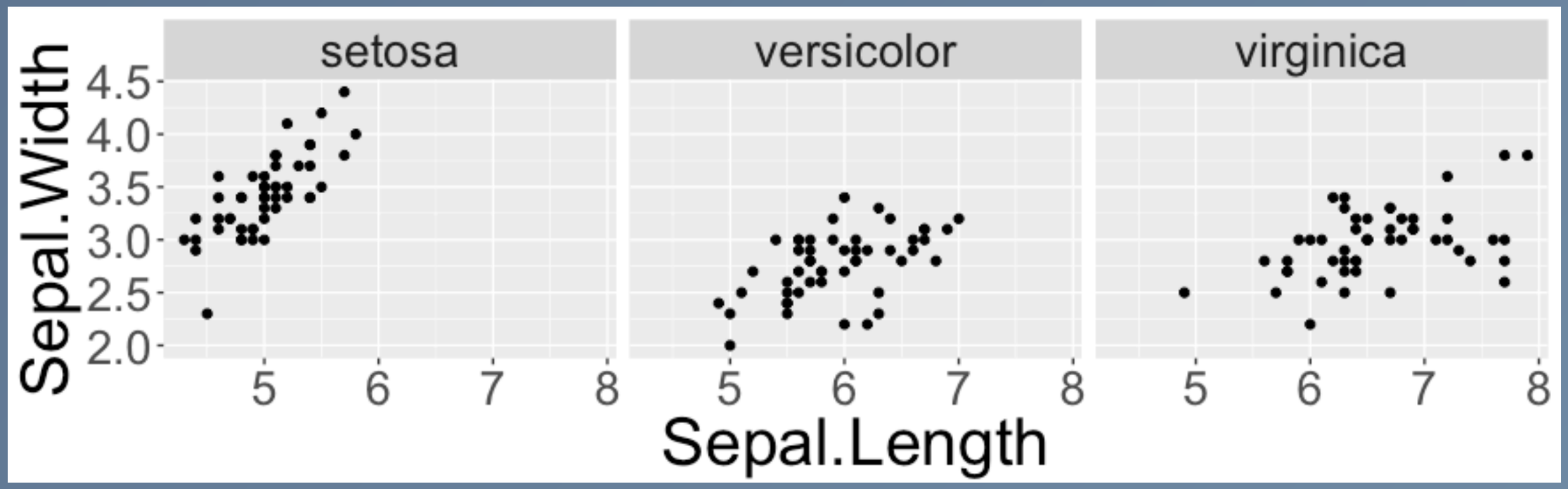


The Grammar of Graphics



Facets

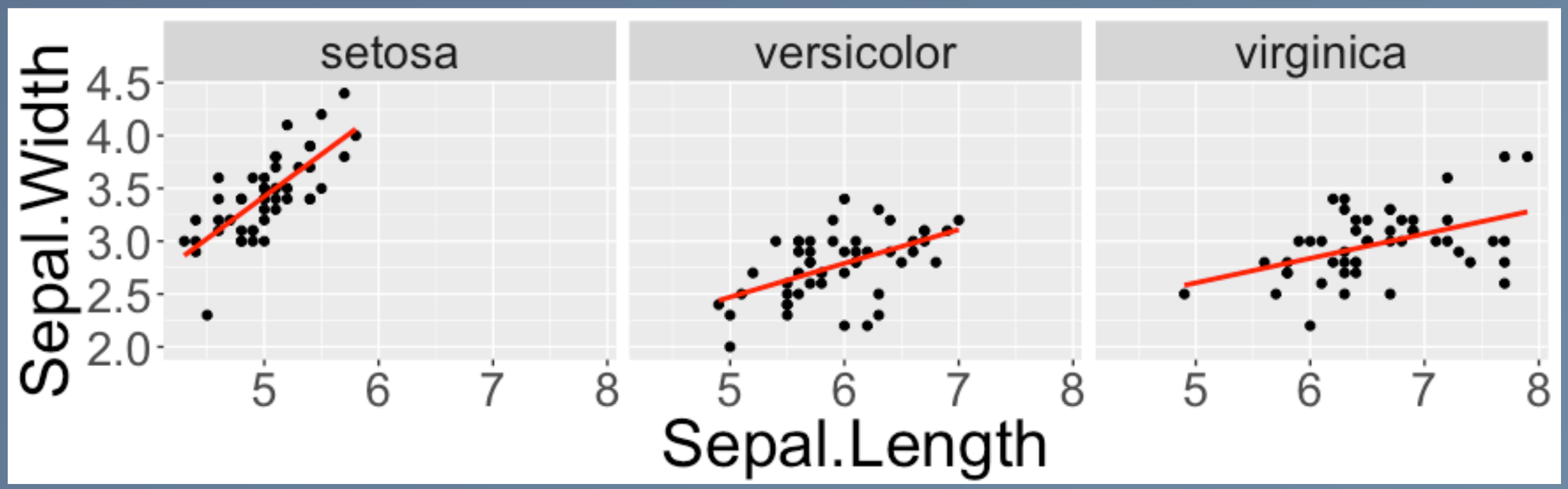
```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point() +
  facet_grid(.~Species)
```



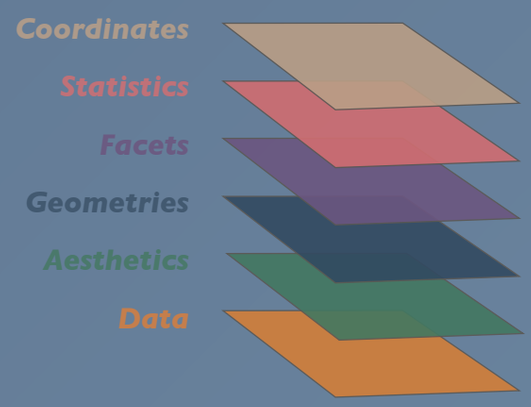
The Grammar of Graphics

Statistics

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point() +
  facet_grid(.~Species) +
  stat_smooth(method = "lm",
              se = F,
              col = "red")
```

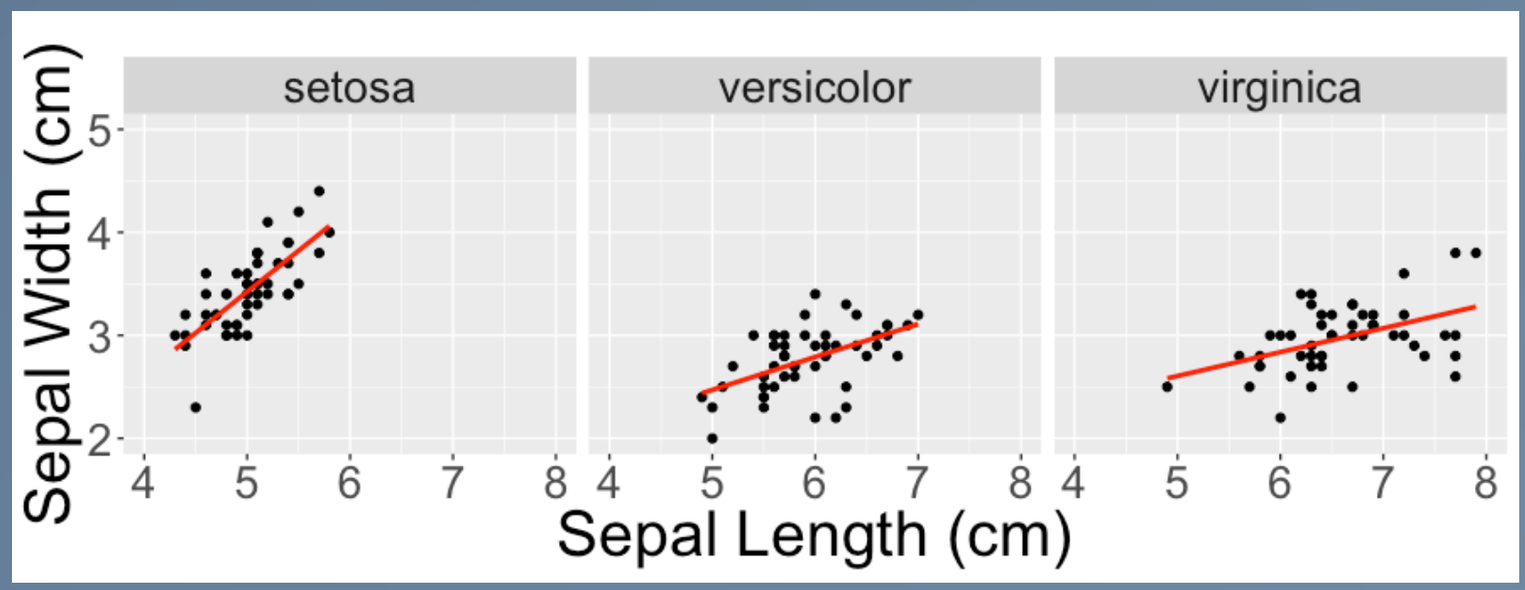


The Grammar of Graphics

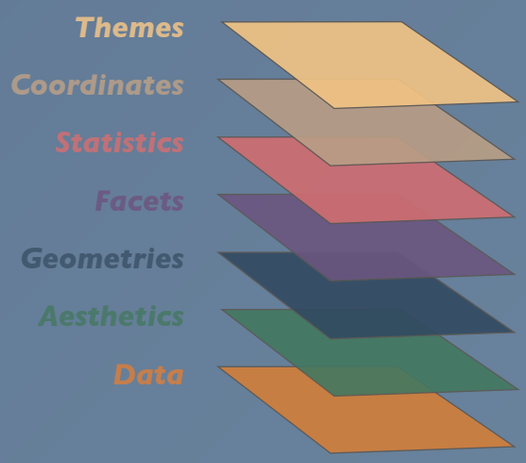


Coordinates

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point() +
  facet_grid(.~Species) +
  stat_smooth(method = "lm",
              se = F,
              col = "red") +
scale_y_continuous("Sepal Width (cm)”, limits = c(2,5)) +
scale_x_continuous("Sepal Length (cm)”, limits = c(4,8)) +
coord_equal()
```

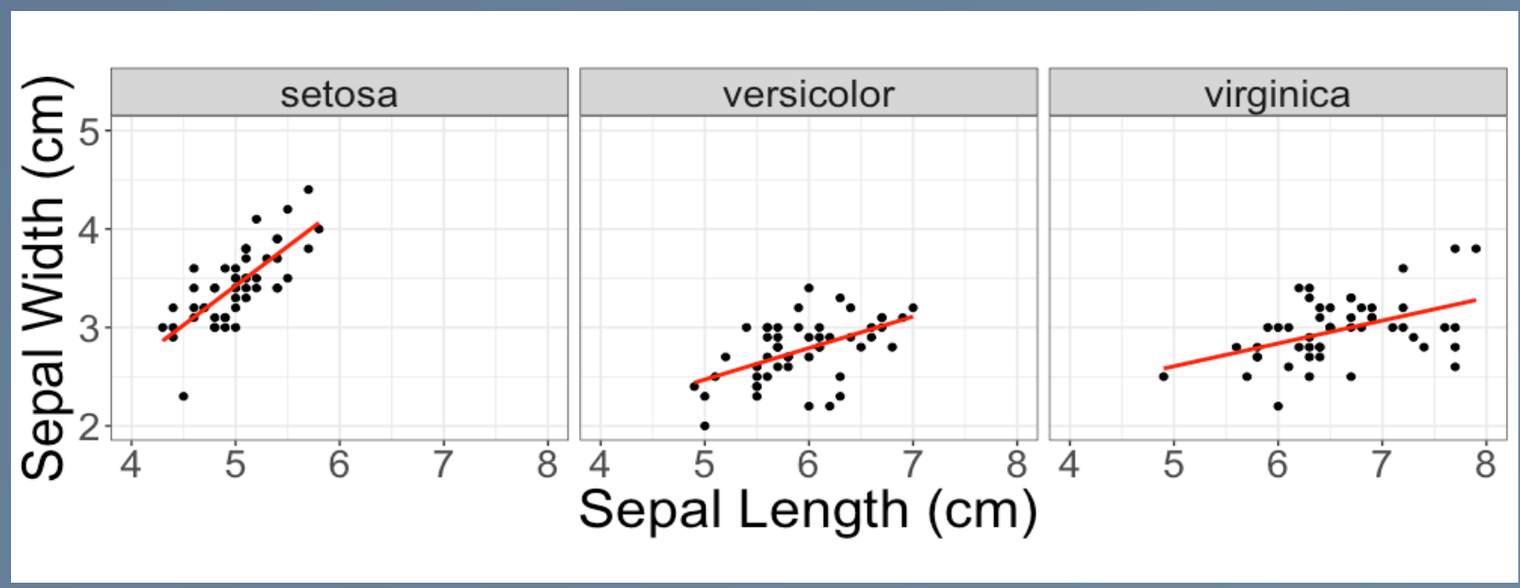


The Grammar of Graphics



Themes

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point() +
  facet_grid(.~Species) +
  stat_smooth(method = "lm",
              se = F,
              col = "red") +
  scale_y_continuous("Sepal Width (cm)", limits = c(2,5)) +
  scale_x_continuous("Sepal Length (cm)", limits = c(4,8)) +
  coord_equal() +
  theme_bw()
```



The Data Layer

The Data layer is simply the data set that holds the variables and observations that you want to plot

The `iris` data set:

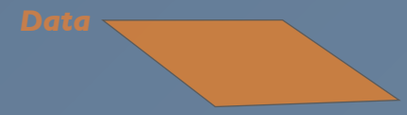
- Measurements of sepal and petal width and length from 150 flowers of the *Iris* genus
- Data from three species: *Setosa*, *versicolor*, and *virginica*

Iris setosa
http://www.twofrog.com/images/iris38a.jpg

Iris versicolor
https://www.aquaportail.com/aquabdd/photos/iris-versicolor.jpg

Iris virginica
https://upload.wikimedia.org/wikipedia/commons/9/9f/Iris_virginica.jpg

The Data Layer



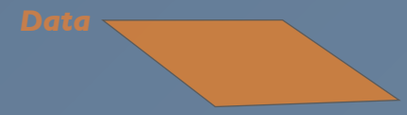
Data Frames

- Table/spreadsheet data format in R
- Built on top of R lists
- Each column is an atomic vector, i.e. all elements have the same data class
- Different columns can have different data classes

The `iris` data frame

	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
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa

The Data Layer



Data Frames

- Table/spreadsheet data format in R
- Built on top of R lists
- Each column is an atomic vector, i.e. all elements have the same data class
- Different columns can have different data classes

Tidy Data

- Each column represents a variable
- Each row is an observation of the variables
- The `tidyr` package provides tools for tidying your data frames

The `iris` data frame

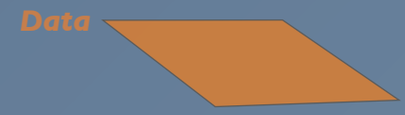
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa

The Data Layer

Question: Is the iris data frame in "tidy" format?

	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
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa

The Data Layer



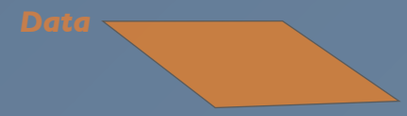
No! "Sepal" and "Petal" are observations of a variable that describes the part of the flower.

	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
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa

tidyr

	Flower	Species	Part	Length	Width
1	1	setosa	Petal	1.4	0.2
2	1	setosa	Sepal	5.1	3.5
3	2	setosa	Petal	1.4	0.2
4	2	setosa	Sepal	4.9	3.0
5	3	setosa	Petal	1.3	0.2
6	3	setosa	Sepal	4.7	3.2
7	4	setosa	Petal	1.5	0.2
8	4	setosa	Sepal	4.6	3.1
9	5	setosa	Petal	1.4	0.2
10	5	setosa	Sepal	5.0	3.6
11	6	setosa	Petal	1.7	0.4
12	6	setosa	Sepal	5.4	3.9
13	7	setosa	Petal	1.4	0.3
14	7	setosa	Sepal	4.6	3.4
15	8	setosa	Petal	1.5	0.2
16	8	setosa	Sepal	5.0	3.4
17	9	setosa	Petal	1.4	0.2
18	9	setosa	Sepal	4.4	2.9
19	10	setosa	Petal	1.5	0.1
20	10	setosa	Sepal	4.9	3.1

The Data Layer



How do we construct a plot?

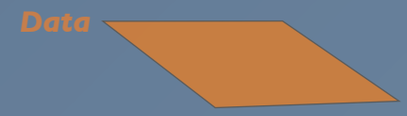
The basic plotting command in ggplot2 is `ggplot()`.
The first argument is the data frame containing your data:

```
ggplot(iris, …)
```

The `ggplot()` command also includes the basic aesthetic mappings.
These are defined within `aes()`:

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

The Data Layer



How do we construct a plot?

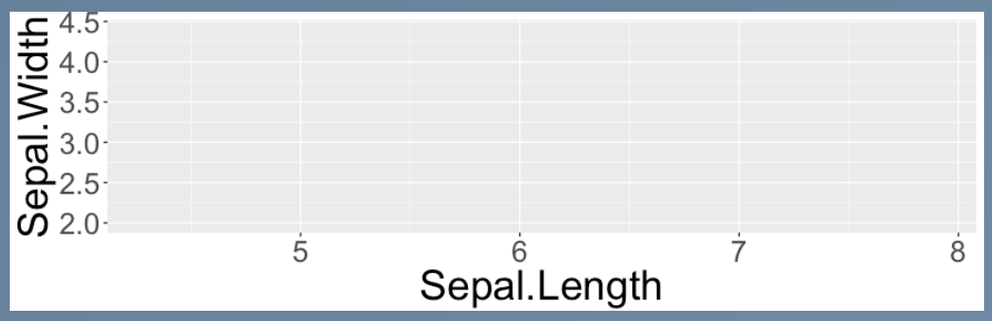
The basic plotting command in ggplot2 is `ggplot()`.
The first argument is the data frame containing your data:

```
ggplot(iris, …)
```

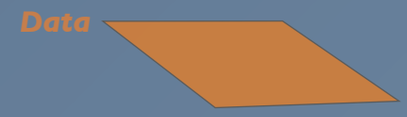
The `ggplot()` command also includes the basic aesthetic mappings.
These are defined within `aes()`:

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

But this on its own doesn't generate a plot.



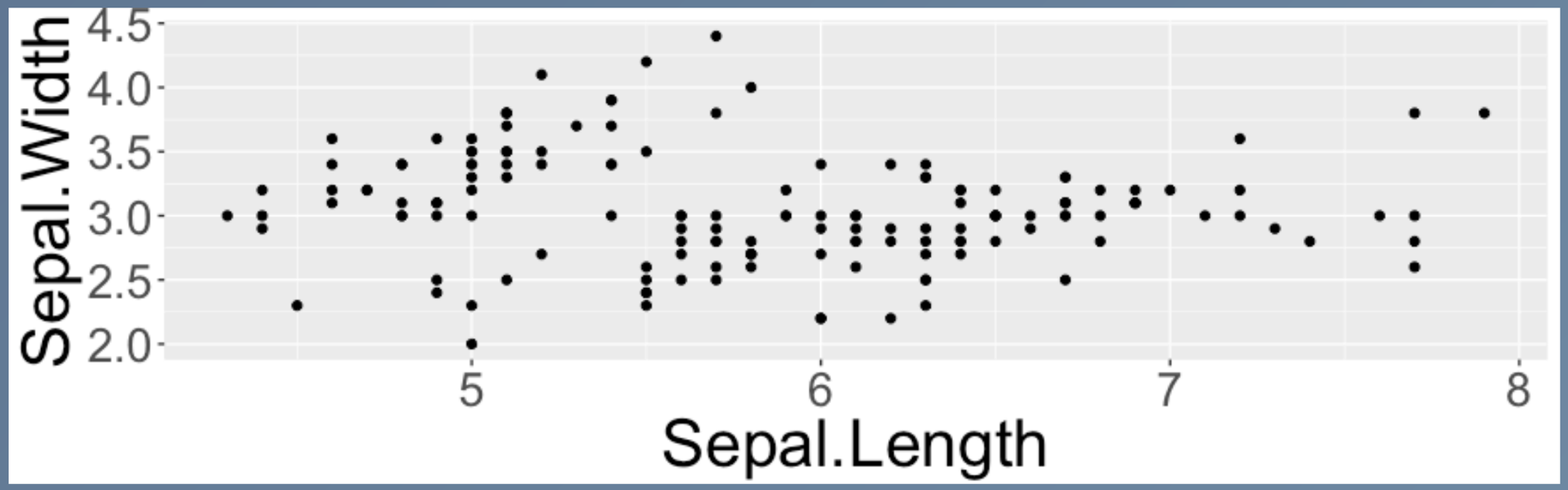
The Data Layer



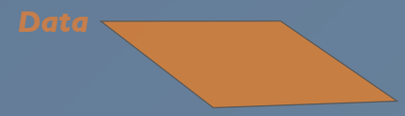
How do we construct a plot?

Remember there are three essential layers to generate a plot:
data, aesthetics, and geometry:

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



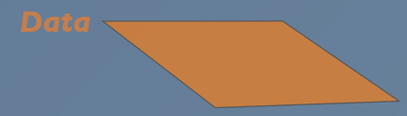
The Data Layer



The importance of data structure

In ggplot2 the structure of your data frame plays an essential role in how your visualization will turn out.

The Data Layer



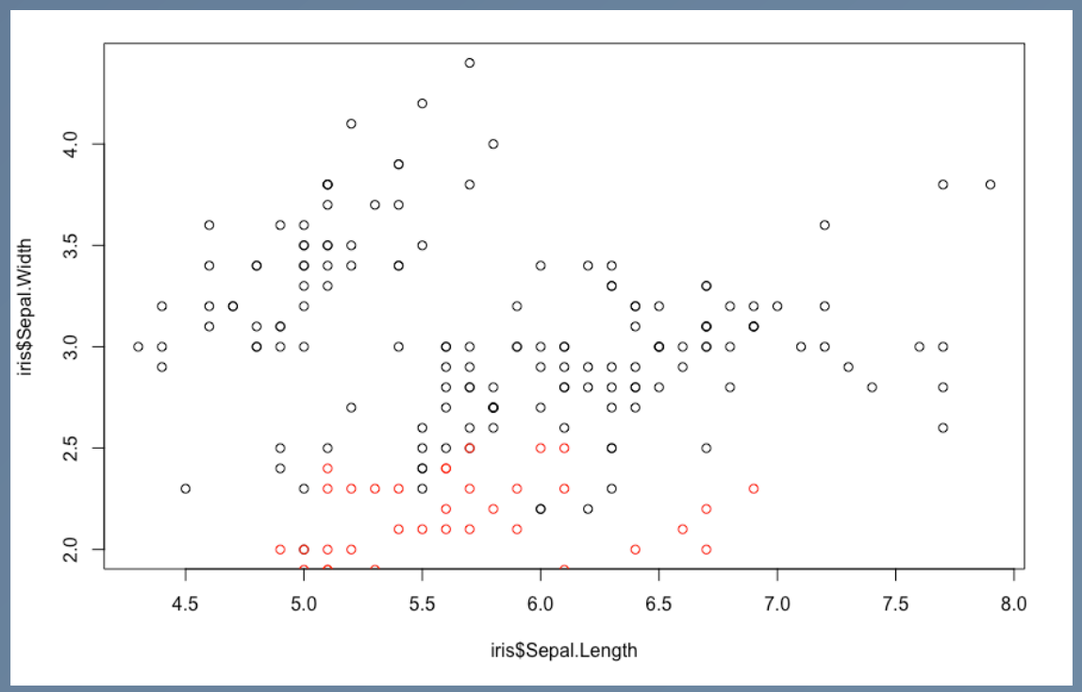
The importance of data structure

In ggplot2 the structure of your data frame plays an essential role in how your visualization will turn out.

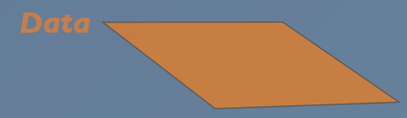
Suppose we want to make a scatter plot of width vs. length but want to include data from both sepal and petal measurements.

In base R we can use the `plot()` and `points()` functions:

```
plot(x = iris$Sepal.Length,
     y = iris$Sepal.Width)

points(x = iris$Petal.Length,
       y = iris$Petal.Width,
       col = "red")
```



The Data Layer

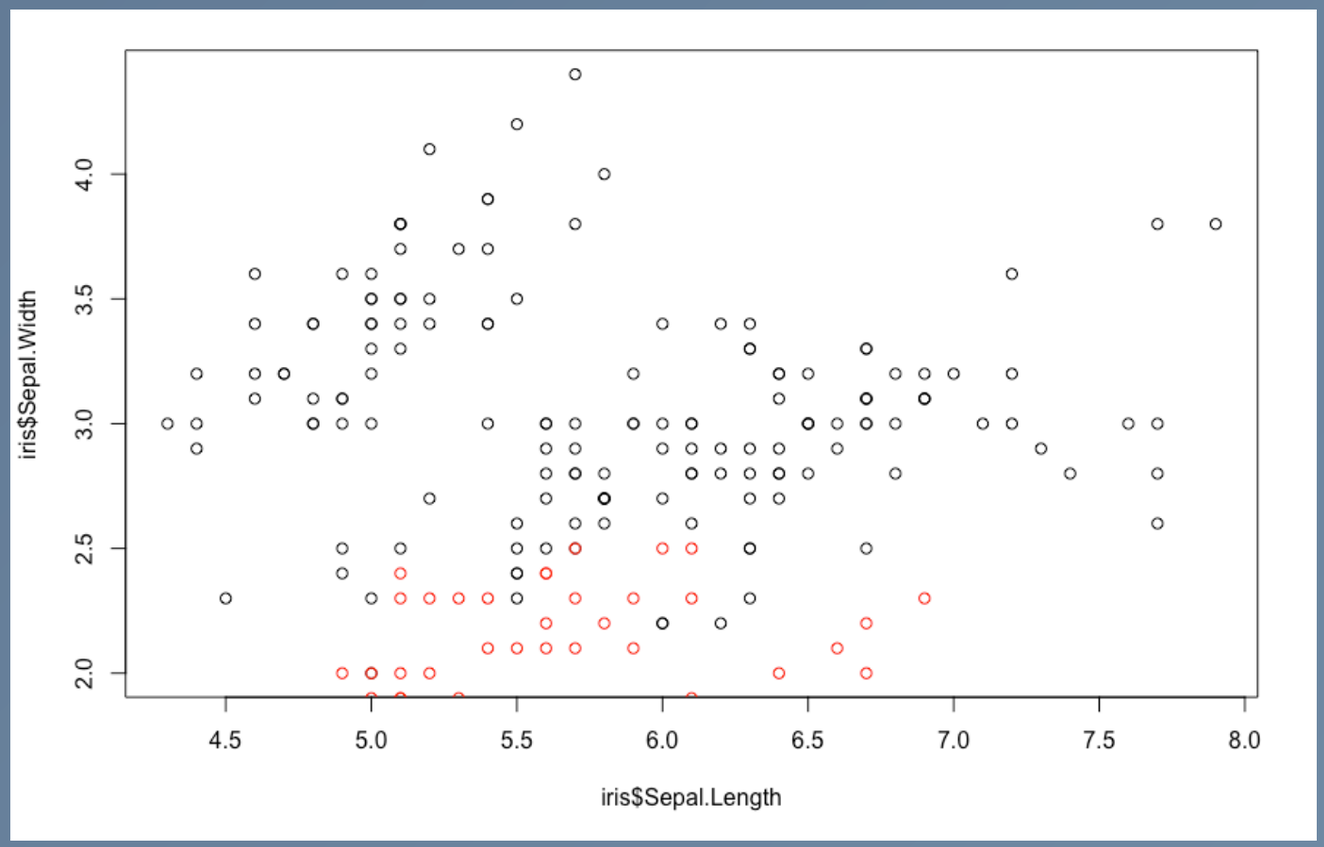


The importance of data structure

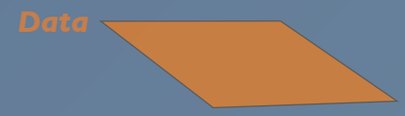
```
plot(iris$Sepal.Length, iris$Sepal.Width)
points(iris$Petal.Length, iris$Petal.Width, col = "red")
```

Some limitations:

1. The plot is not re-drawn with the new data. Original axes are maintained
2. The plot is stored as an image. The new points are added as additional ink
3. There is no legend



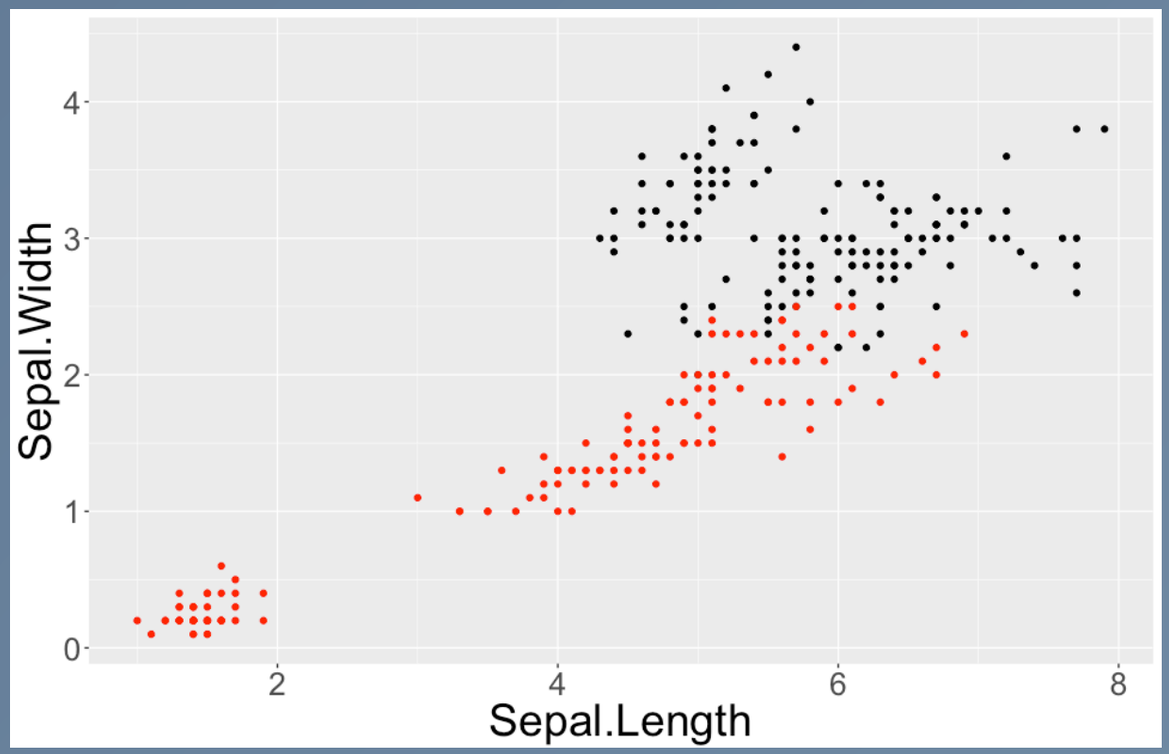
The Data Layer



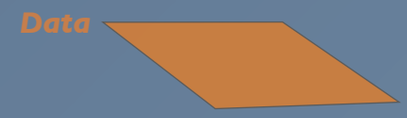
The importance of data structure

This can be reproduced in ggplot by defining a new aesthetic mapping within an additional geometry layer

```
ggplot(iris, aes(x = Sepal.Length,
                 y = Sepal.Width)) +
  geom_point() +
  geom_point(aes(x = Petal.Length,
                 y = Petal.Width),
             col = "red")
```



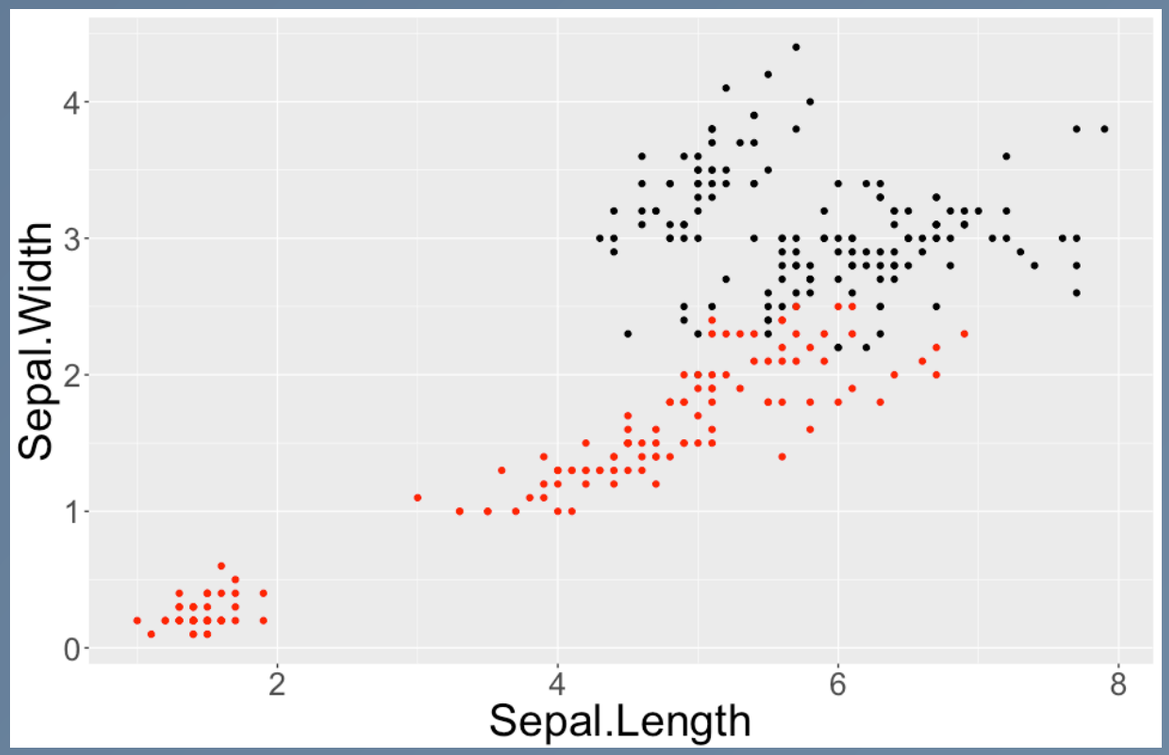
The Data Layer



The importance of data structure

This can be reproduced in ggplot by defining a new aesthetic mapping within an additional geometry layer

```
ggplot(iris, aes(x = Sepal.Length,
                 y = Sepal.Width)) +
  geom_point() +
  geom_point(aes(x = Petal.Length,
                 y = Petal.Width),
             col = "red")
```

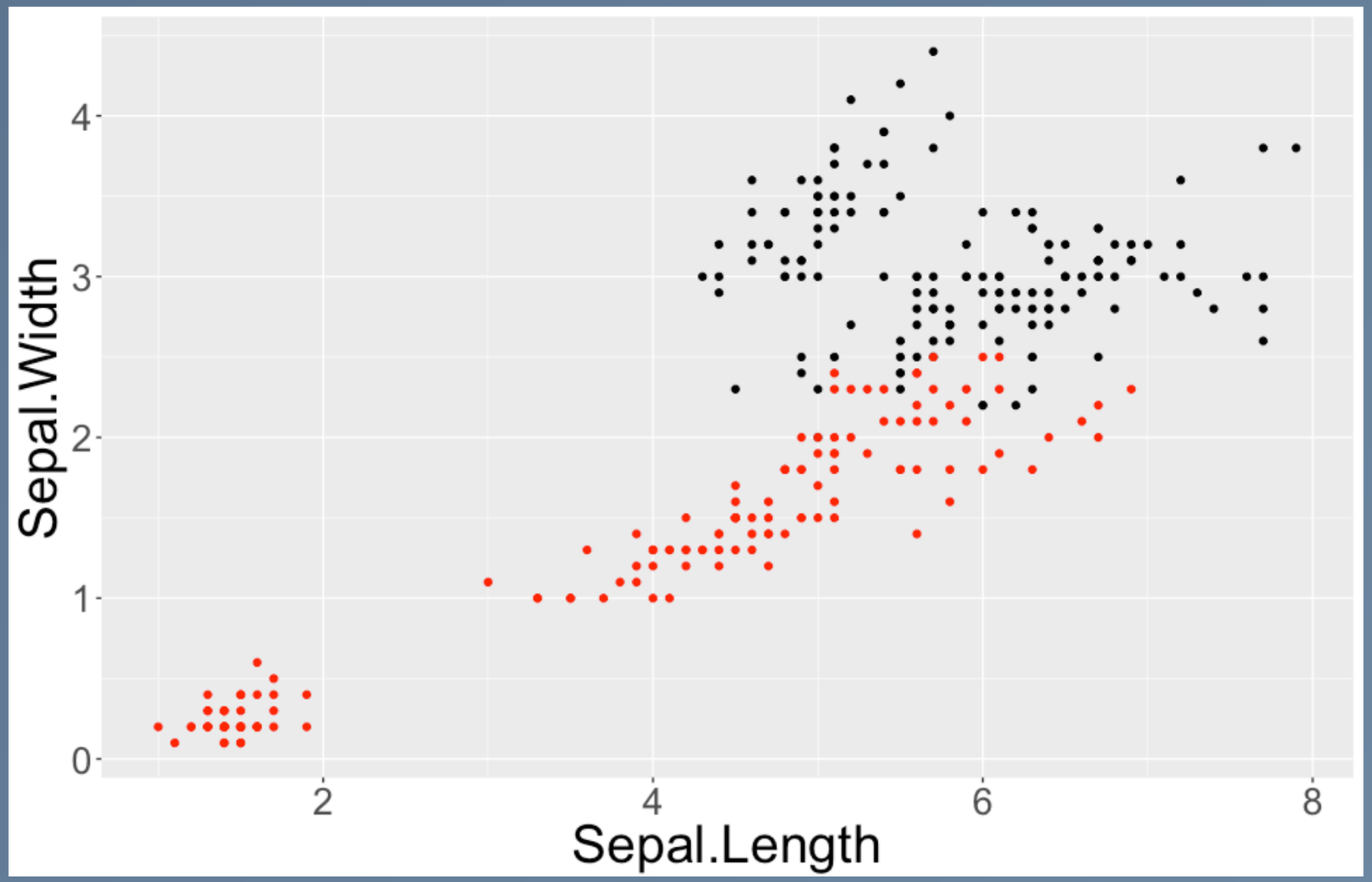


What is wrong with this plot?

The Data Layer

The importance of data structure

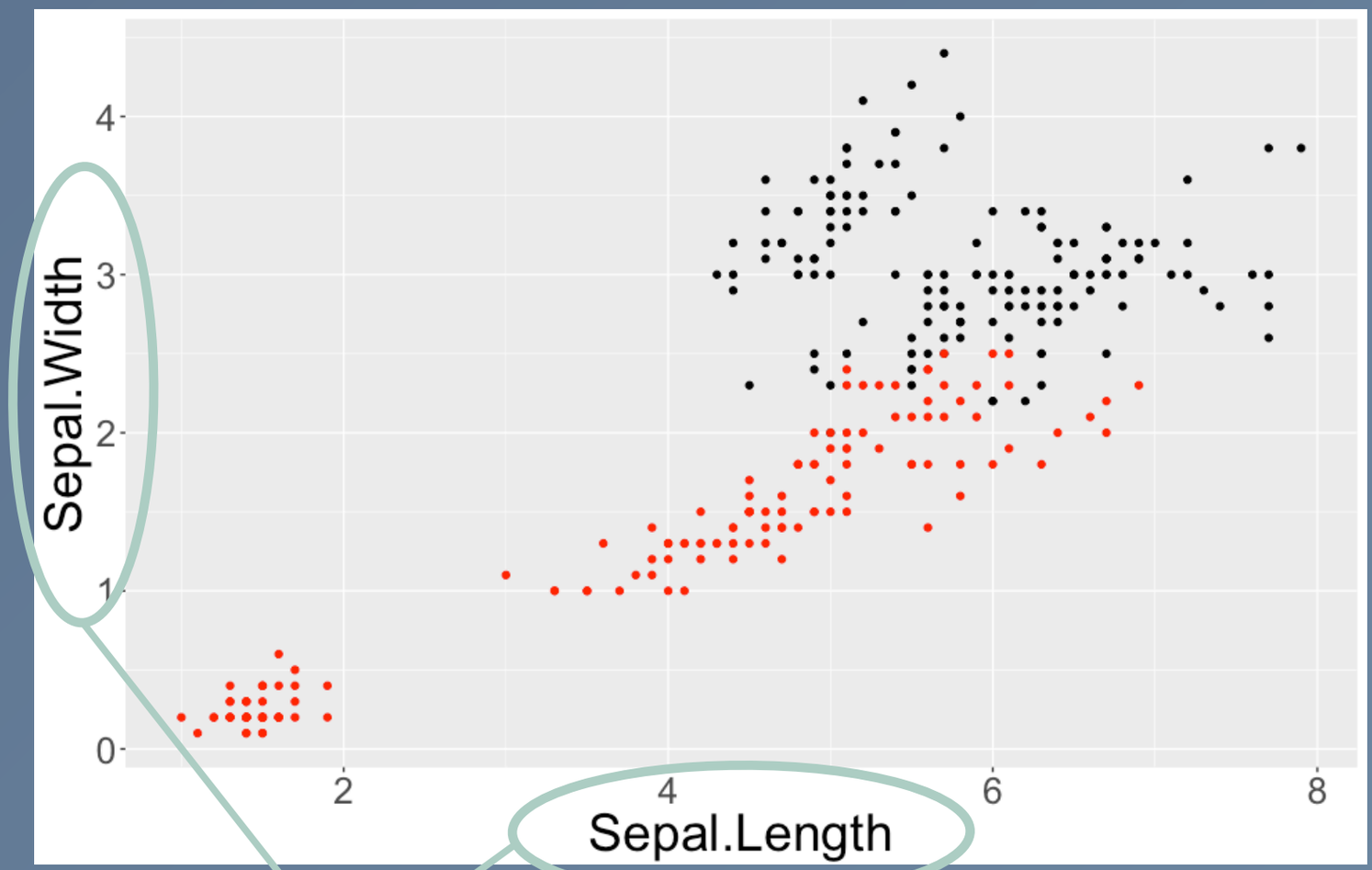
What is wrong with this plot?



The Data Layer

Data

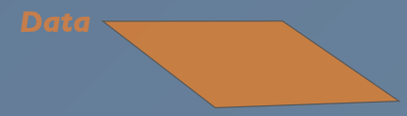
The importance of data structure



Legend?

Incorrect axis titles

The Data Layer



The importance of data structure

We need to re-structure the data frame to get the aesthetic mappings that we want:

```
iris$Flower <- 1:nrow(iris)
```

```
iris.wide <- iris %>%
  gather(key, value, -Species, -Flower) %>%
  separate(key, c("Part", "Measure"), "\\.") %>%
  spread(Measure, value) %>%
  select(Flower, Species, Part, Length, Width)
```

Aside: The Pipe Operator, %>%

The pipe operator comes from the `magrittr` package by Stefan Milton Bache.

It is loaded as part of the `dplyr` package within the Tidyverse.

The pipe assigns the object on the left to the first argument of the function on the right.

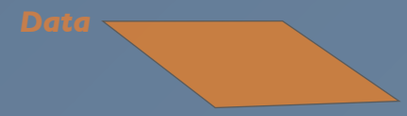
Example:

```
mean(x, na.rm = TRUE)
```

```
x %>% mean(na.rm = TRUE)
```

This is extremely handy for transforming data frames using functions from `dplyr` and `tidyr`.

The Data Layer



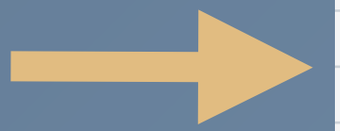
The importance of data structure

We need to re-structure the data frame to get the aesthetic mappings that we want:

```
iris$Flower <- 1:nrow(iris)
```

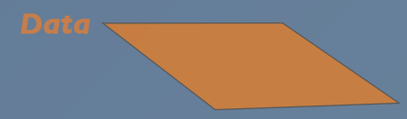
```
iris.wide <- iris %>%
  gather(key, value, -Species, -Flower) %>%
  separate(key, c("Part", "Measure"), "\\.") %>%
  spread(Measure, value) %>%
  select(Flower, Species, Part, Length, Width)
```

	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



	Flower	Species	Part	Length	Width
1	1	setosa	Petal	1.4	0.2
2	1	setosa	Sepal	5.1	3.5
3	2	setosa	Petal	1.4	0.2
4	2	setosa	Sepal	4.9	3.0
5	3	setosa	Petal	1.3	0.2

The Data Layer

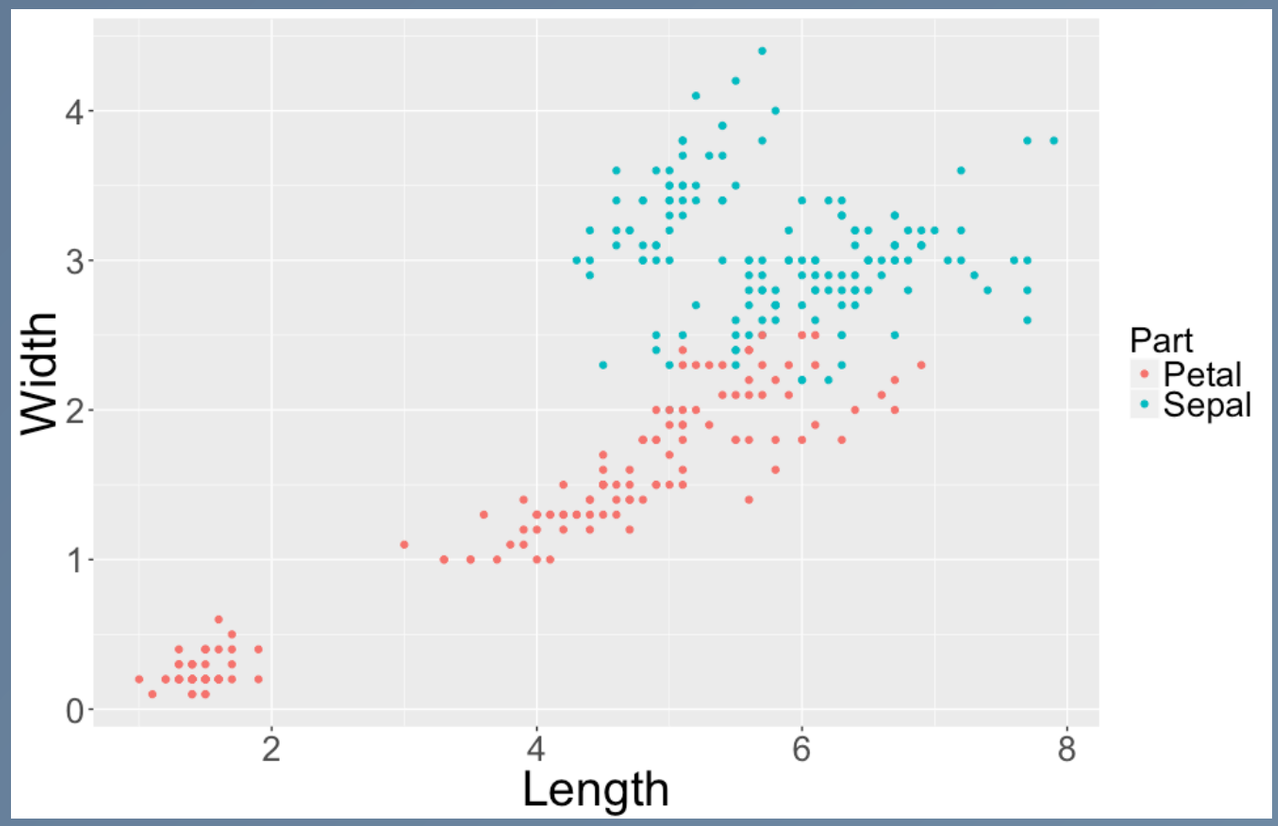


The importance of data structure

We have a new variable, "Part", that we can map to the proper aesthetic, colour

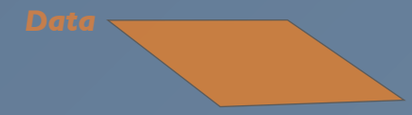
	Flower	Species	Part	Length	Width
1	1	setosa	Petal	1.4	0.2
2	1	setosa	Sepal	5.1	3.5
3	2	setosa	Petal	1.4	0.2
4	2	setosa	Sepal	4.9	3.0
5	3	setosa	Petal	1.3	0.2

```
ggplot(iris.wide,
       aes(x = Length,
           y = Width,
           col = Part)) +
  geom_point()
```



- Correct axis labels
- Automatic legend

The Data Layer

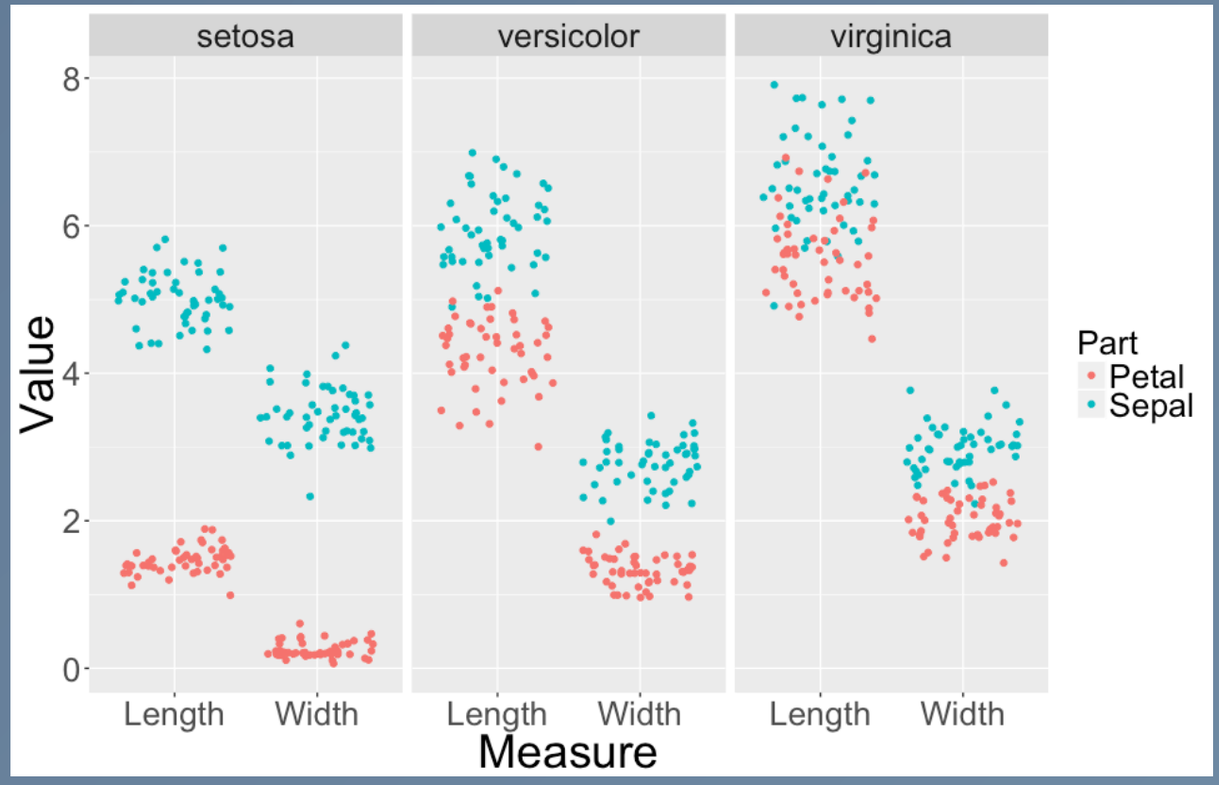


The importance of data structure

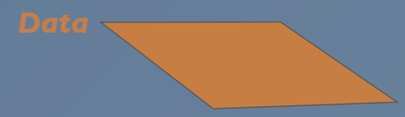
Different organizations of the data allow us to create different plots

	Species	Flower	Part	Measure	Value
1	setosa	1	Sepal	Length	5.1
2	setosa	2	Sepal	Length	4.9
3	setosa	3	Sepal	Length	4.7
4	setosa	4	Sepal	Length	4.6
5	setosa	5	Sepal	Length	5.0

```
iris %>%
  gather(key,
         value = Value,
         -Species, -Flower) %>%
  separate(key,
           c("Part", "Measure"),
           "\\.") %>%
  ggplot(aes(x = Measure,
             y = Value,
             col = Part)) +
  geom_jitter() +
  facet_grid(.~Species)
```



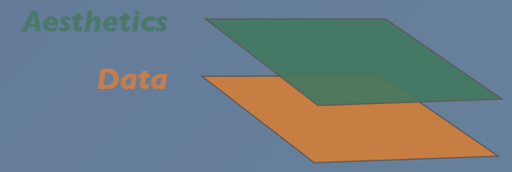
The Data Layer



Summary

- The Data layer consists of the data frame that you want to use for plotting
- Data structure is intimately tied to the kind of plots you can generate
- ggplot works best with tidy data
- Plots are created using aesthetic mappings of variables
- It is good practice to have a single set of aesthetic mappings for any given plot

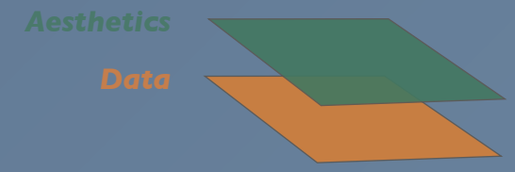
The Aesthetics Layer



Introduction

- In ggplot, aesthetics refer to scales in the graphic that can be used to display information
  - **Example:** The x and y axes of a plot are aesthetics
- Variables, or data columns, are associated with aesthetics via aesthetic mappings
- Aesthetic mappings are created by using `aes()` within the `ggplot()` command
- Aesthetic mappings can also be defined in geometry and statistics layers

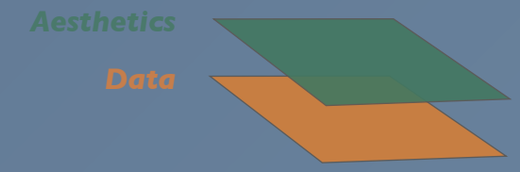
The Aesthetics Layer



Aesthetics in ggplot2

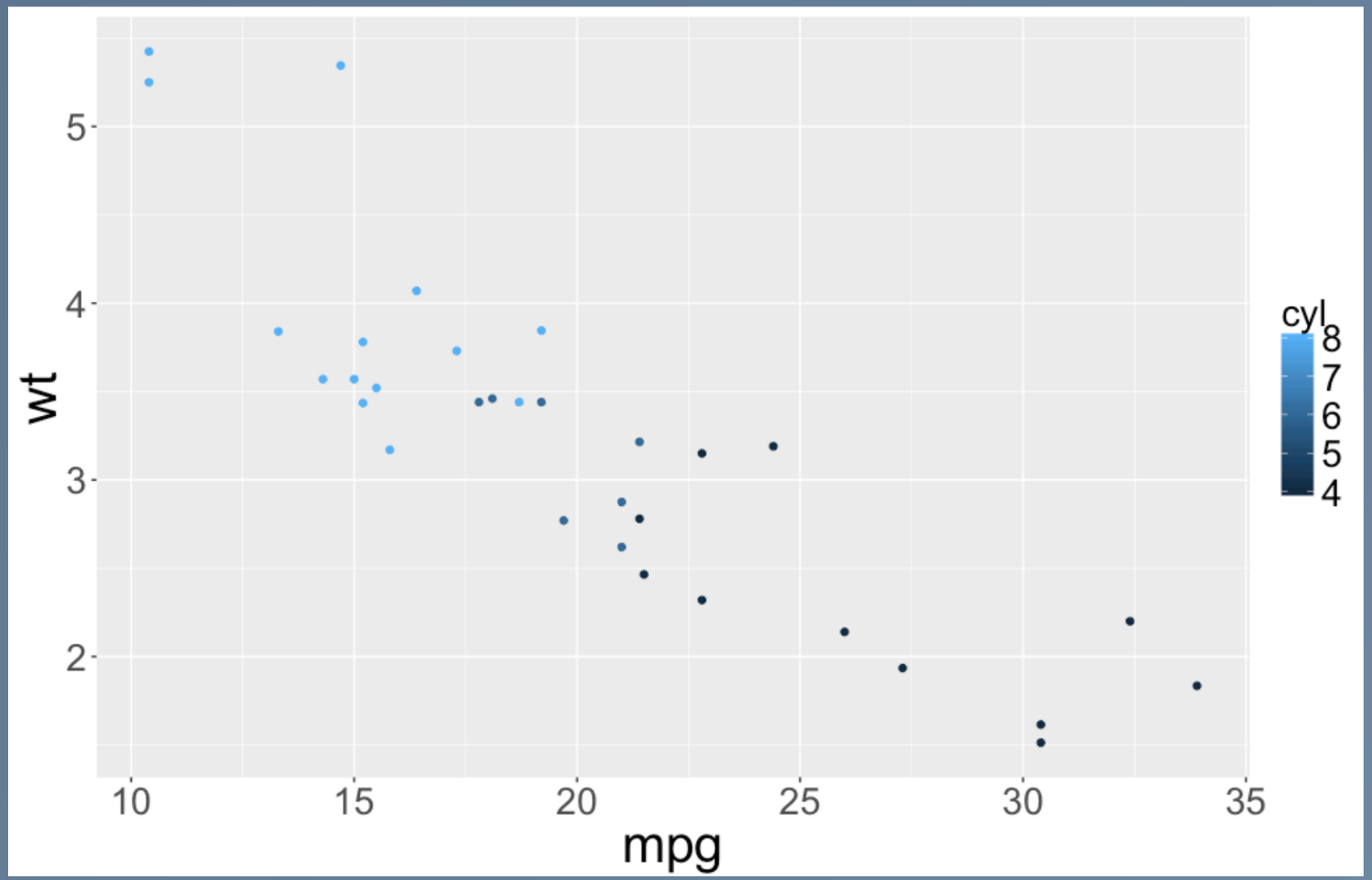
- x ——— The x-axis
- y ——— The y-axis
- col ——— Discrete or continuous colour scales (outline)
- fill ——— Discrete or continuous colour scales (fill)
- size ——— Marker size
- shape ——— Marker shape
- alpha ——— Alpha blending, i.e. transparency
- linetype ——— Line type
- labels ——— Label text

The Aesthetics Layer

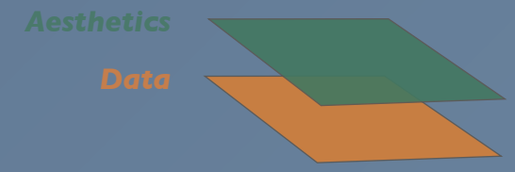


Examples with `mtcars`: x, y and colour (continuous)

```
ggplot(mtcars, aes(x = mpg, y = wt, col = cyl)) +
  geom_point()
```

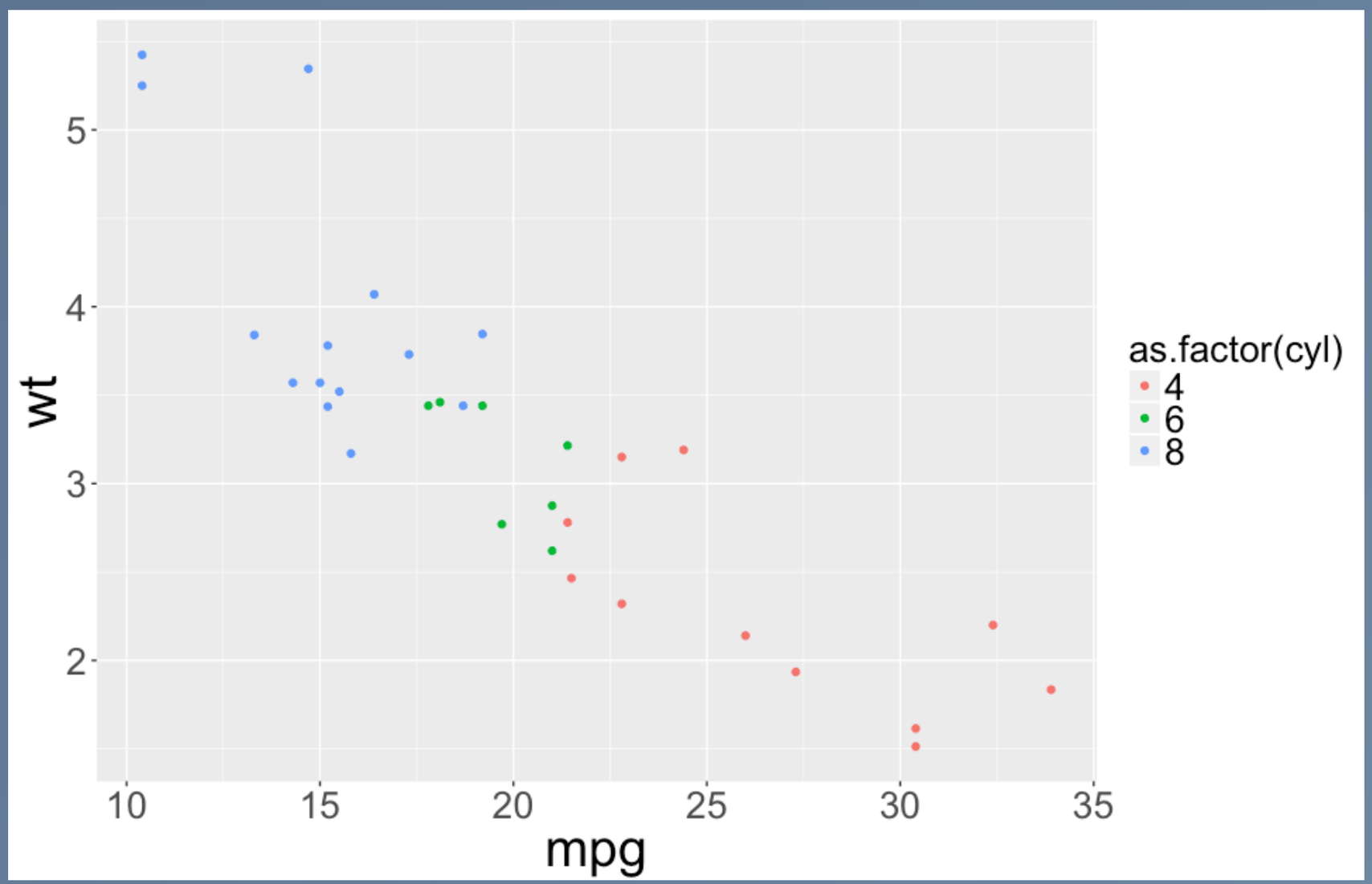


The Aesthetics Layer

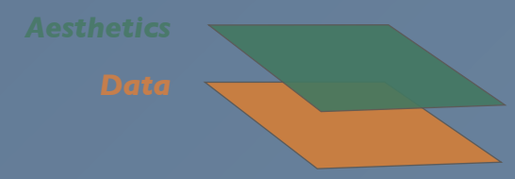


Examples with `mtcars`: x, y and colour (discrete)

```
ggplot(mtcars, aes(x = mpg, y = wt, col = as.factor(cyl))) +
  geom_point()
```

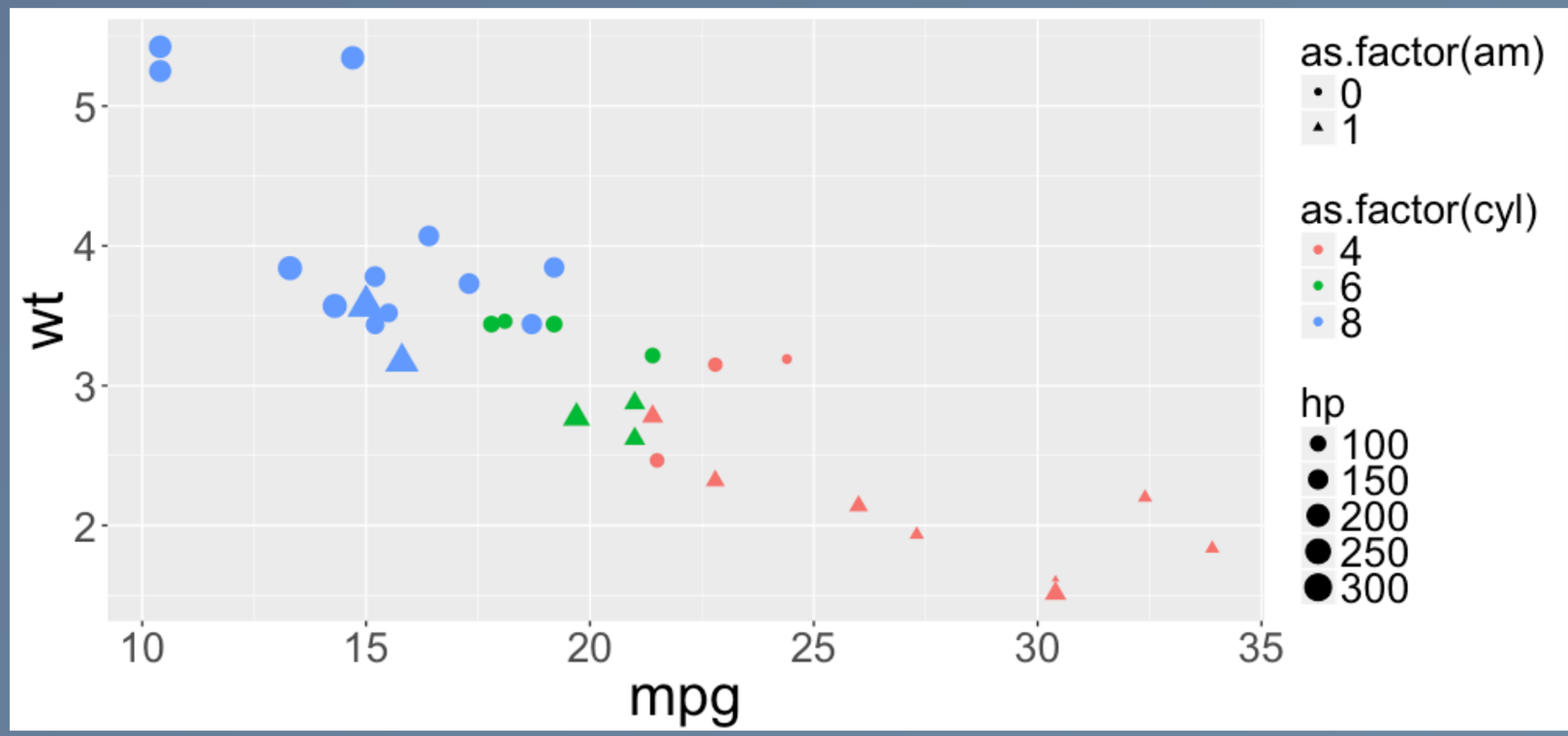


The Aesthetics Layer

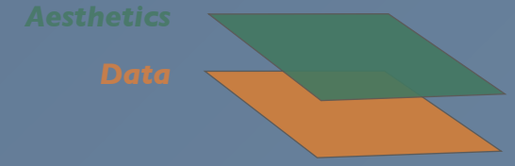


Examples with `mtcars`: x, y, colour, size and shape

```
ggplot(mtcars, aes(x = mpg,
                   y = wt,
                   col = as.factor(cyl),
                   size = hp,
                   shape = as.factor(am))) +
  geom_point()
```

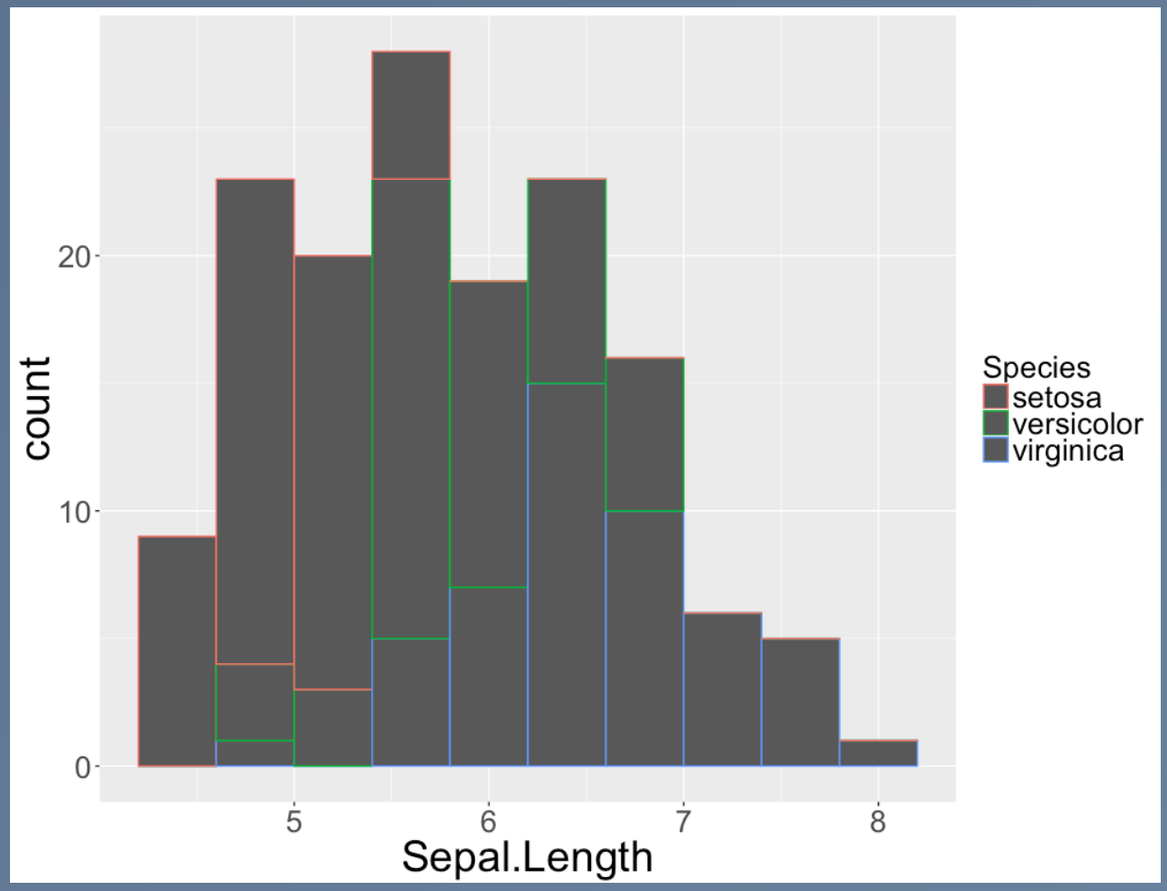


The Aesthetics Layer

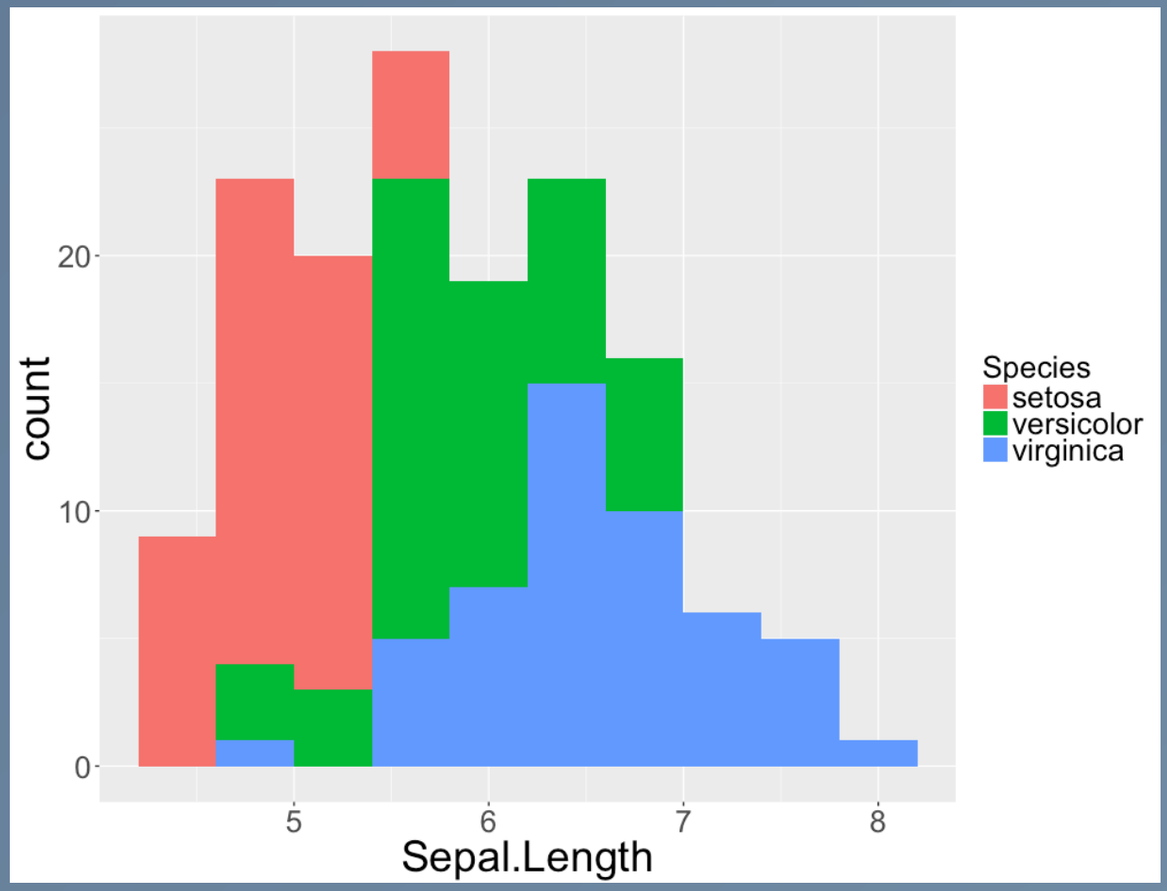


Example with `iris`: col vs. fill

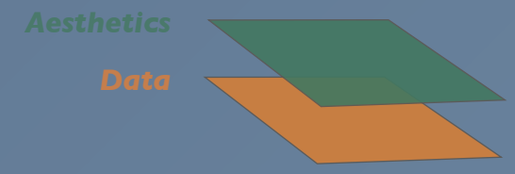
```
ggplot(iris, aes(x = Sepal.Length,
                 col = Species)) +
  geom_histogram(bins = 10)
```




```
ggplot(iris, aes(x = Sepal.Length,
                 fill = Species)) +
  geom_histogram(bins = 10)
```



The Aesthetics Layer



Aesthetics vs. Attributes

Aesthetics

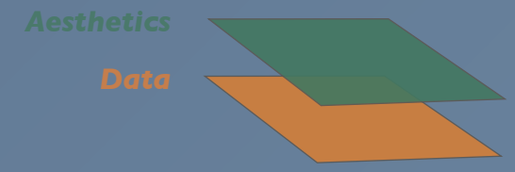
- Aesthetics refer to mappings from the variables in your data frame to scales on the graphic
- Aesthetics are primarily defined in the base `ggplot()` command

Attributes

- Attributes are properties of the visual elements in your graphic
- Attributes are defined in the geometry and statistics layers
- Attributes affect all visual elements in the layer

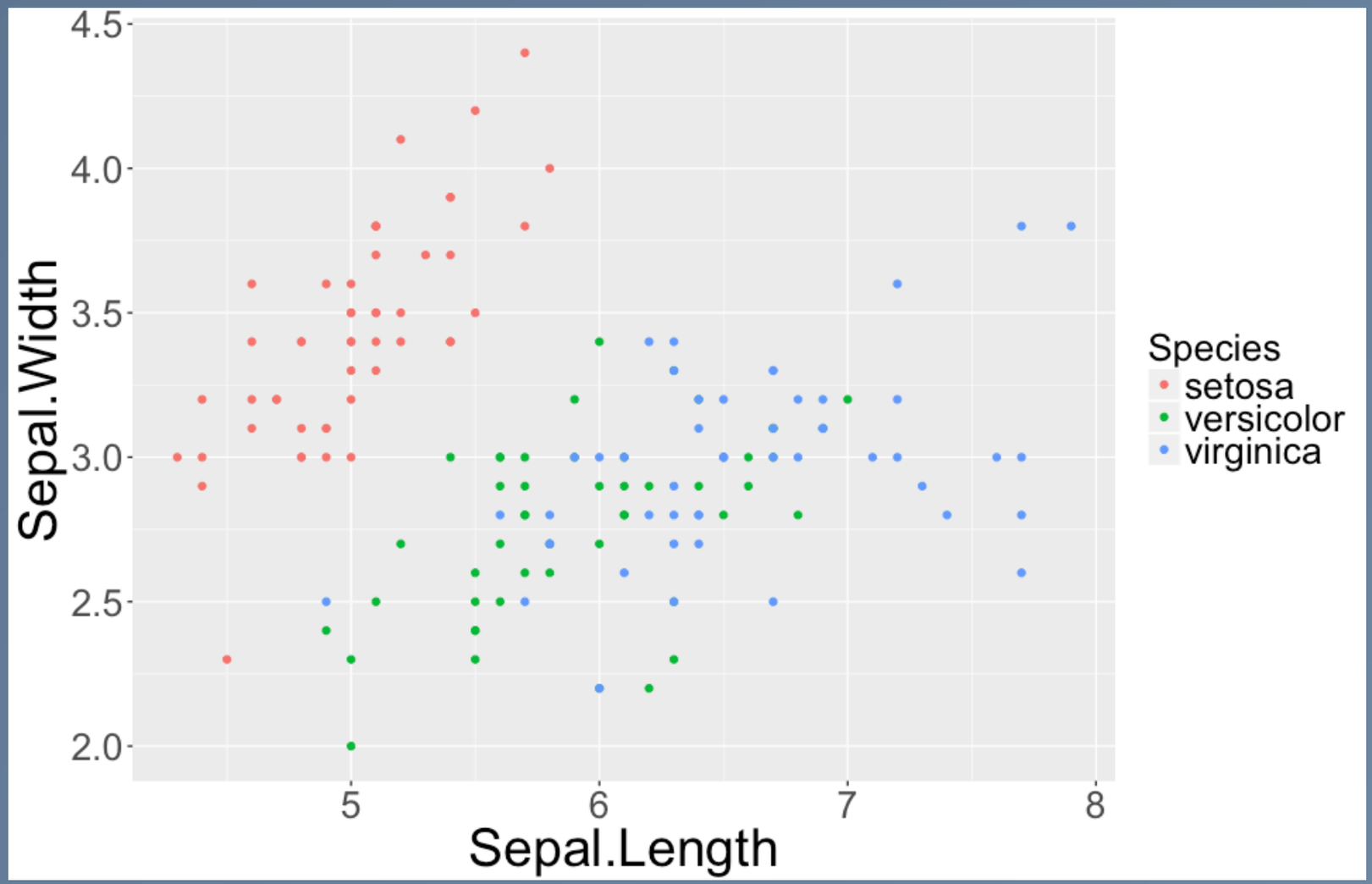
Most aesthetics can also be called as attributes

The Aesthetics Layer

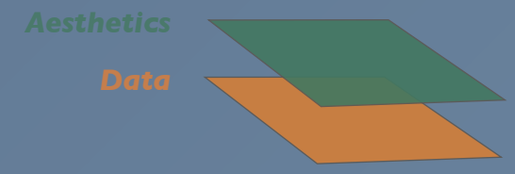


Aesthetics vs. Attributes: Colour

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

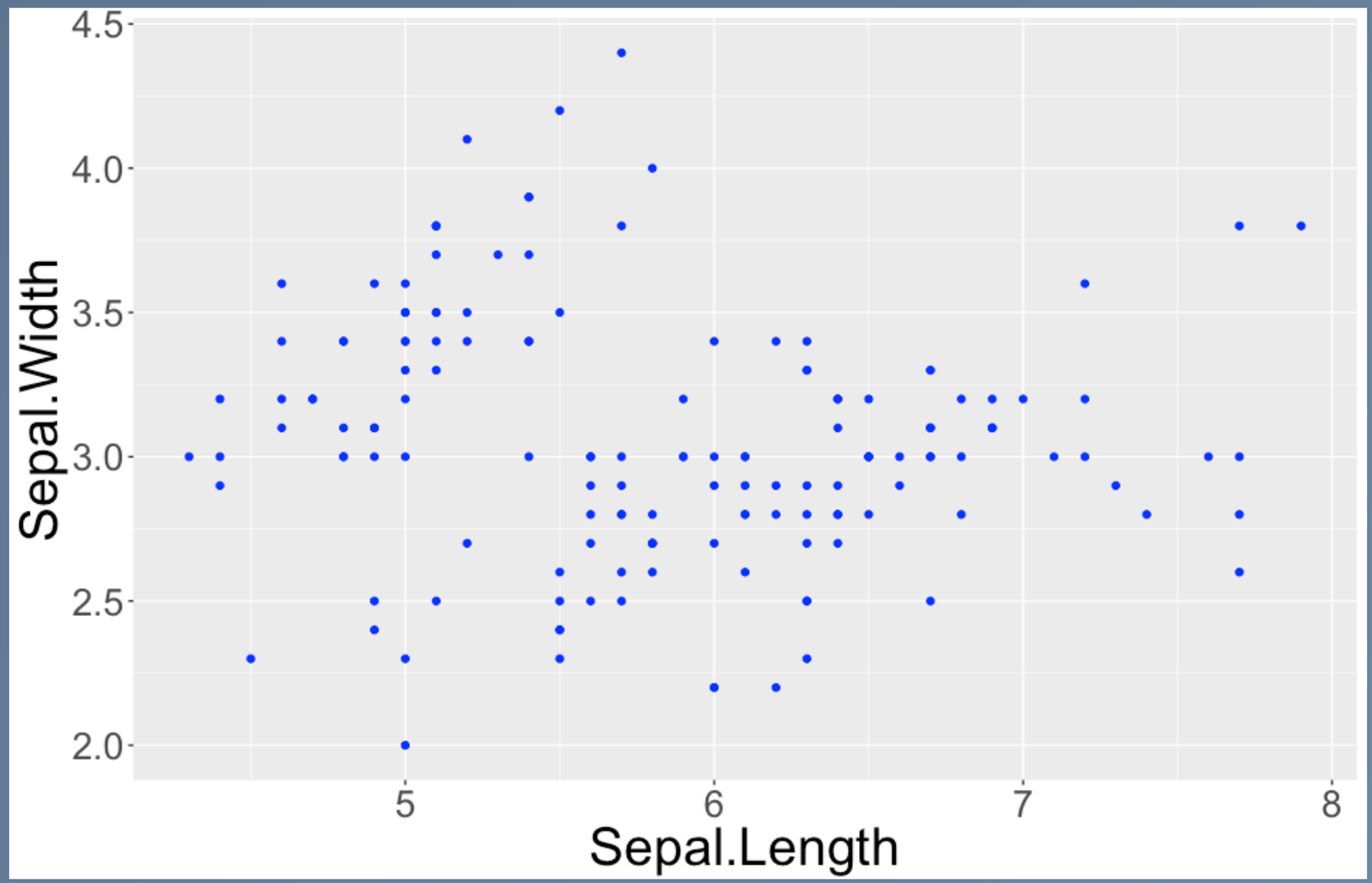


The Aesthetics Layer

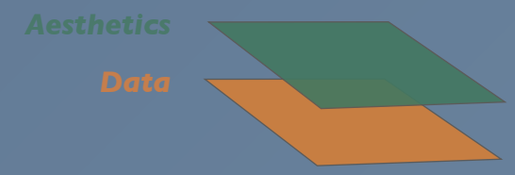


Aesthetics vs. Attributes: Colour

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

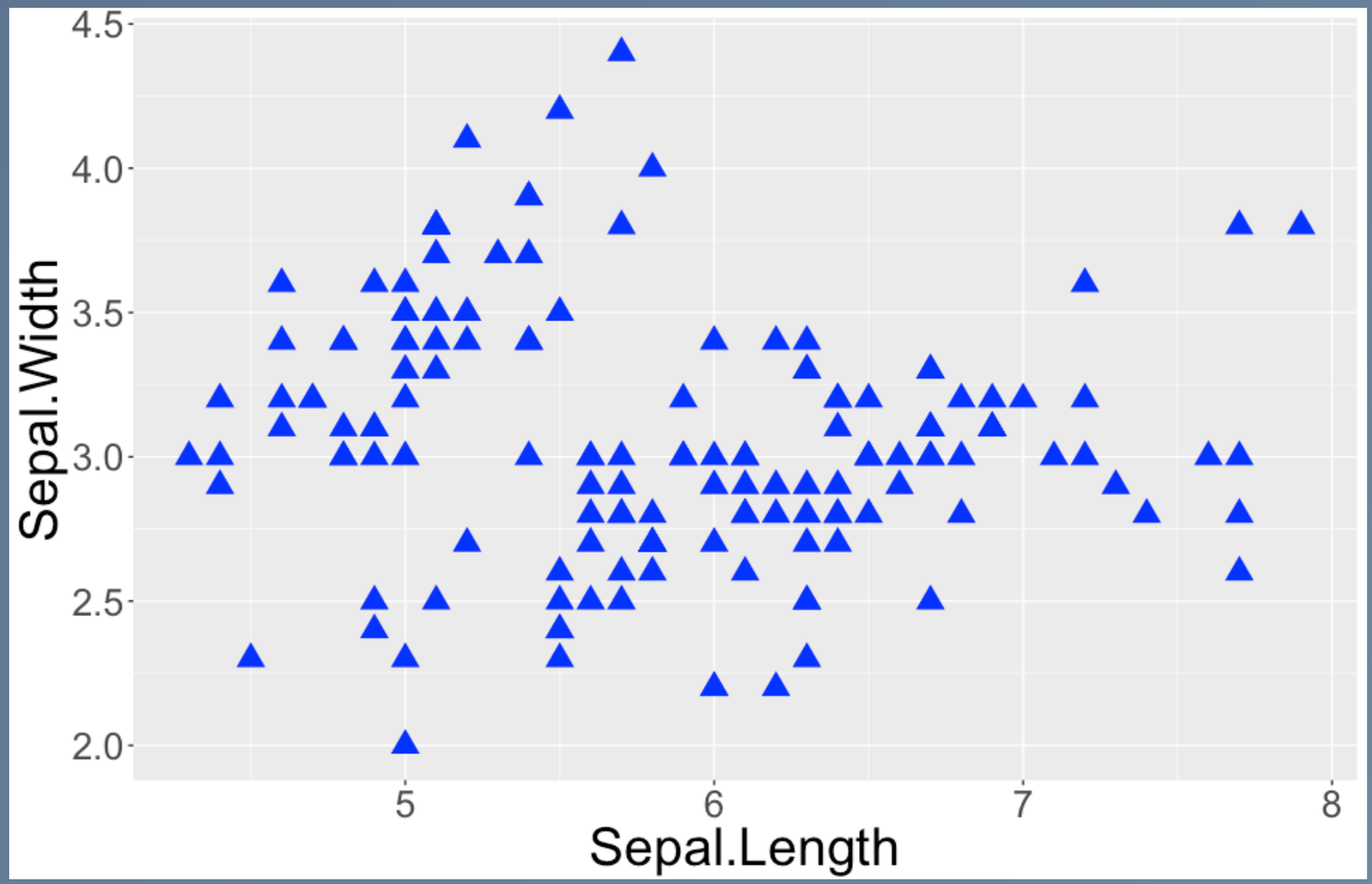


The Aesthetics Layer

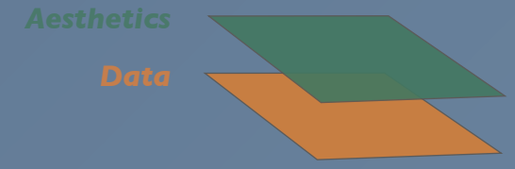


Aesthetics vs. Attributes: Size and shape

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_point(col = “blue”, shape = 17, size = 5)
```

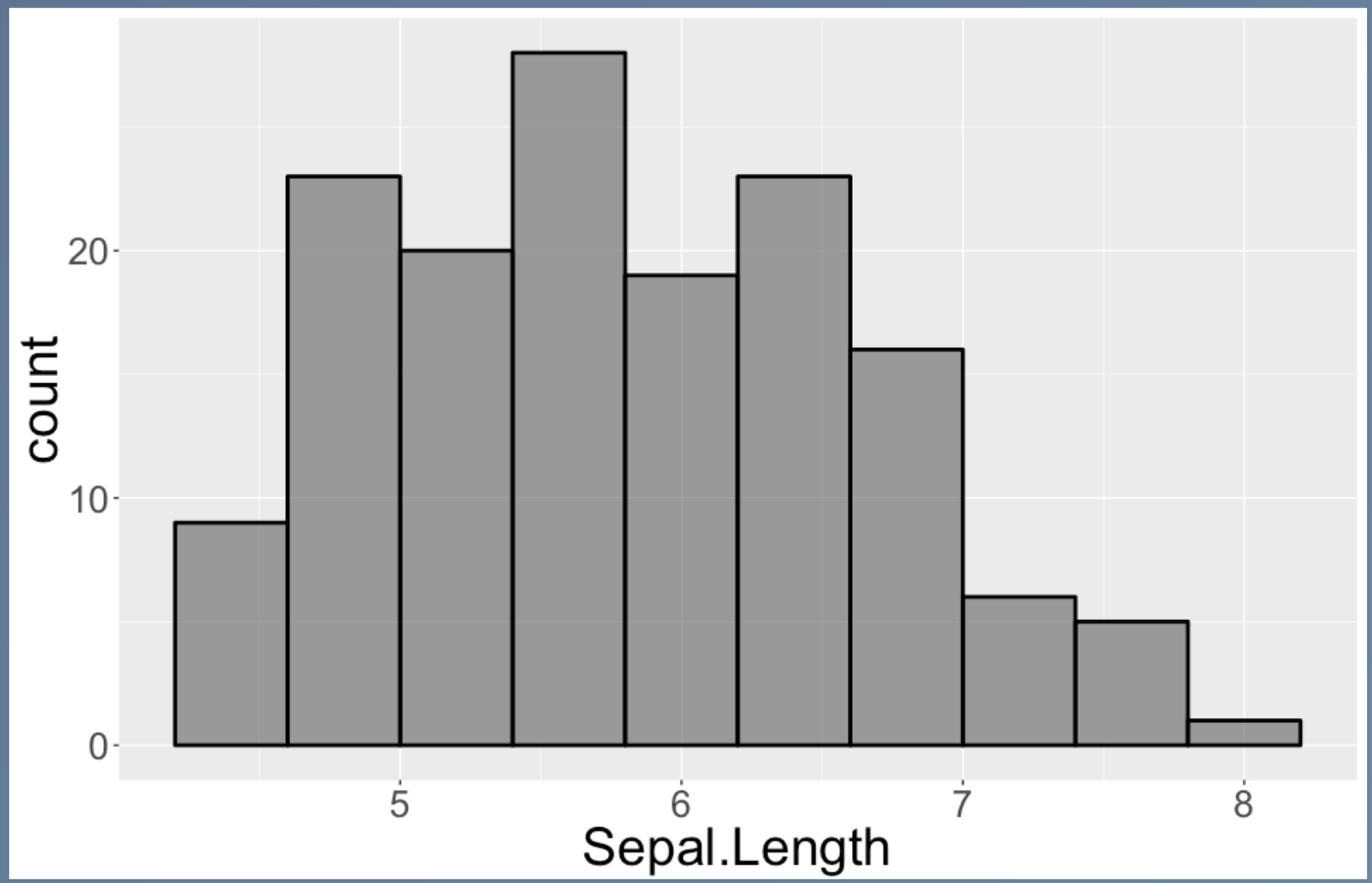


The Aesthetics Layer

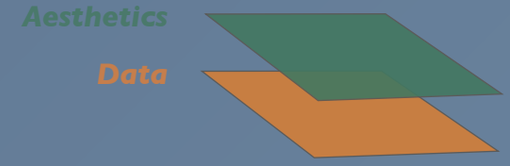


Aesthetics vs. Attributes: Alpha blending

```
ggplot(iris, aes(x = Sepal.Length) +
  geom_histogram(bins = 10, alpha = 0.6, col = “black”, size = 1)
```



The Aesthetics Layer

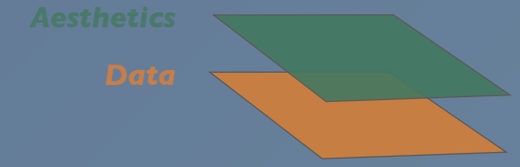


Attribute values

- Colour:
  - R has a huge range of character strings associated with colours
  - Can also use hexadecimal colour values
- Shape:
  - Shapes are identified using shape codes
  - Some shapes have fill and col attributes
- Alpha:
  - Range from 0 to 1 to determine transparency
- Linetype:
  - Can be specified using strings or numbers from 0 to 6
  - Linetypes include “solid”, “dashed”, “dotted”, etc.

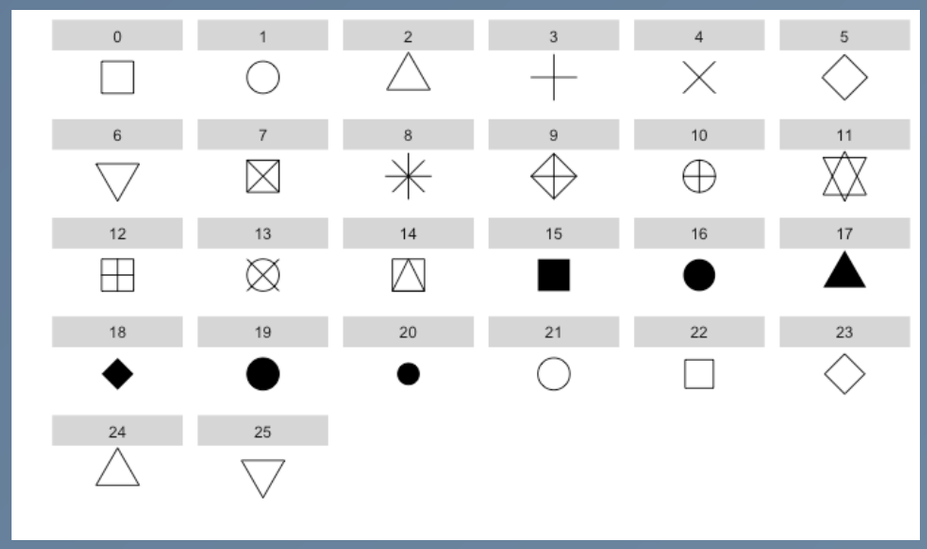
color	name	color	name
	lightpink4		mediumorchid1
	lightsalmon		mediumorchid2
	lightsalmon1		mediumorchid3
	lightsalmon2		mediumorchid4
	lightsalmon3		mediumpurple
	lightsalmon4		mediumpurple1
	lightseagreen		mediumpurple2
	lightskyblue		mediumpurple3
	lightskyblue1		mediumpurple4
	lightskyblue2		mediumseagreen
	lightskyblue3		mediumslateblue
	lightskyblue4		mediumspringgreen
	lightslateblue		mediumturquoise
	lightslategray		mediumvioletred
	lightslategrey		midnightblue
	lightsteelblue		mintcream
	lightsteelblue1		mistyrose
	lightsteelblue2		mistyrose1
	lightsteelblue3		mistyrose2
	lightsteelblue4		mistyrose3
	lightyellow		mistyrose4
	lightyellow1		moccasin
	lightyellow2		navajowhite
	lightyellow3		navajowhite1
	lightyellow4		navajowhite2
	limegreen		navajowhite3
	linen		navajowhite4
	magenta		navy
	magenta1		navyblue
	magenta2		oldlace
	magenta3		olivedrab
	magenta4		olivedrab1
	maroon		olivedrab2
	maroon1		olivedrab3
	maroon2		olivedrab4
	maroon3		orange
	maroon4		orange1
	mediumaquamarine		orange2
	mediumblue		orange3
	mediumorchid		orange4

The Aesthetics Layer



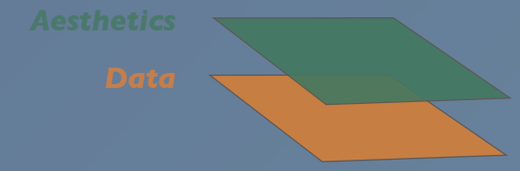
Attribute values

- Colour:
  - R has a huge range of character strings associated with colours
  - Can also use hexadecimal colour values
- Shape:
  - Shapes are identified using shape codes
  - Some shapes have fill and col attributes
- Alpha:
  - Range from 0 to 1 to determine transparency
- Linetype:
  - Can be specified using strings or numbers from 0 to 6
  - Linetypes include "solid", "dashed", "dotted", etc.



http://www.win-vector.com/blog/wp-content/uploads/2016/04/shapeCodes.png

The Aesthetics Layer

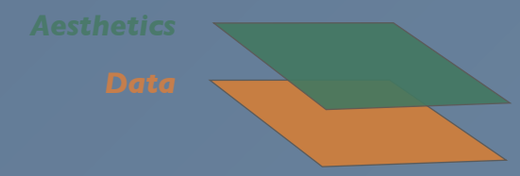


Attribute values

- Colour:
  - R has a huge range of character strings associated with colours
  - Can also use hexadecimal colour values
- Shape:
  - Shapes are identified using shape codes
  - Some shapes have fill and col attributes
- Alpha:
  - Range from 0 to 1 to determine transparency
- Linetype:
  - Can be specified using strings or numbers from 0 to 6
  - Linetypes include "solid", "dashed", "dotted", etc.

http://sape.inf.usi.ch/quick-reference/ggplot2/linetype

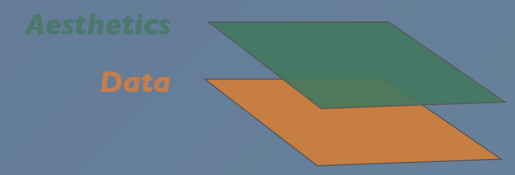
The Aesthetics Layer



Best practices

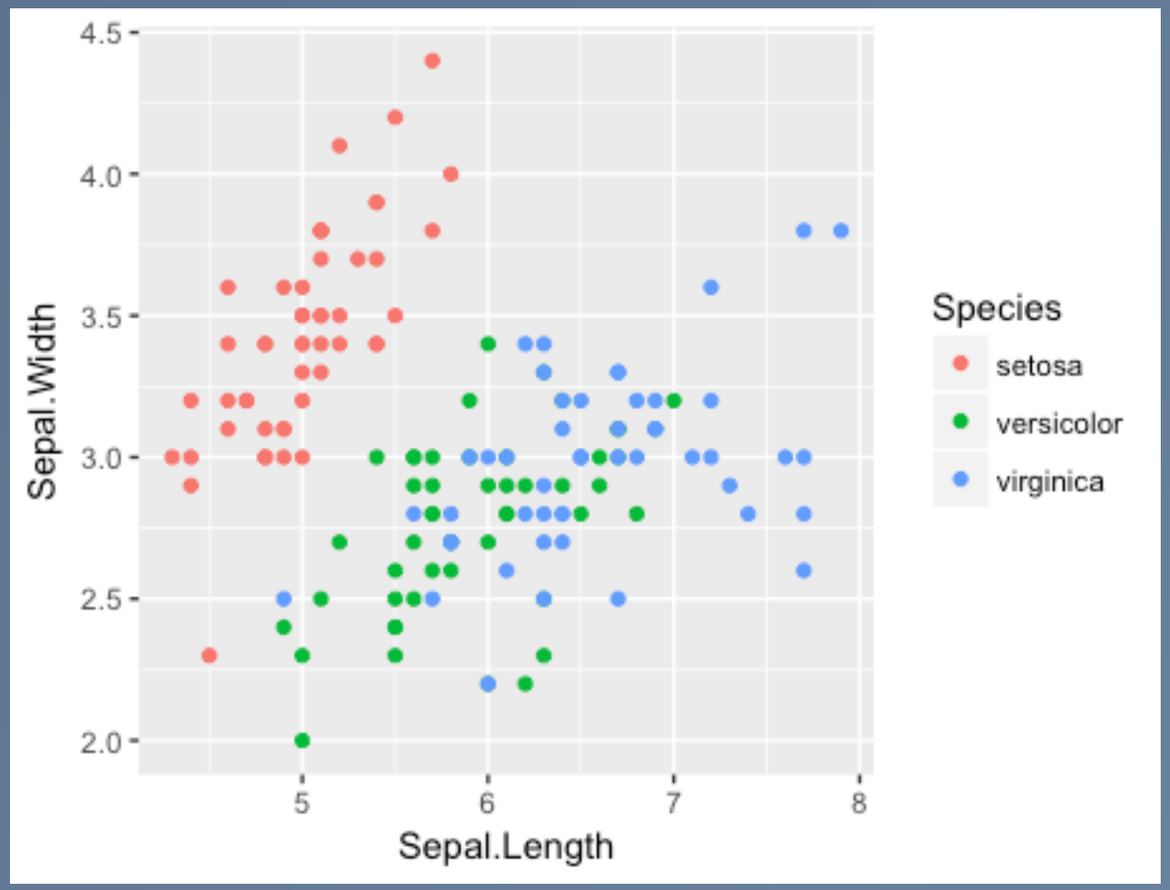
- Form follows function:
  - Choose your aesthetic mappings based on the information that you want to convey
- Different aesthetics work well for different variable types

The Aesthetics Layer

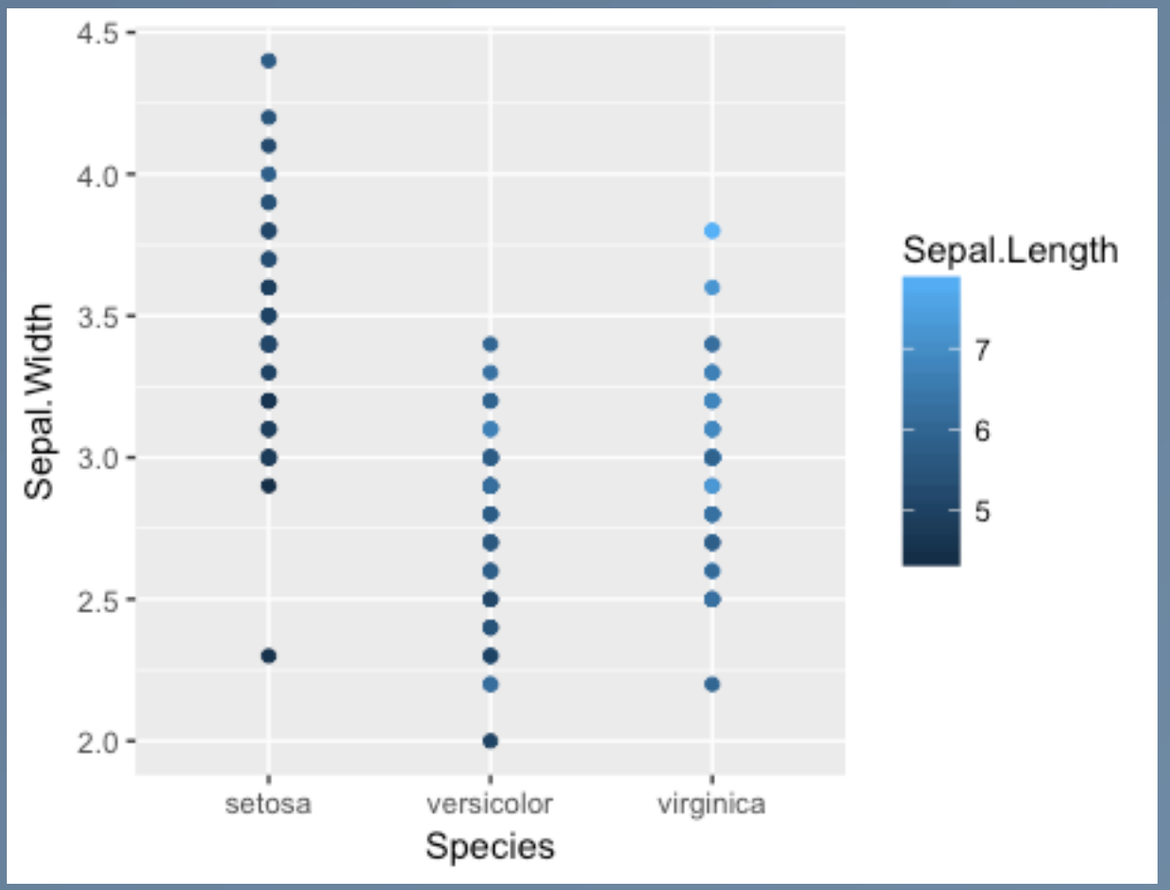


Best practices Different aesthetics work well for different variable types

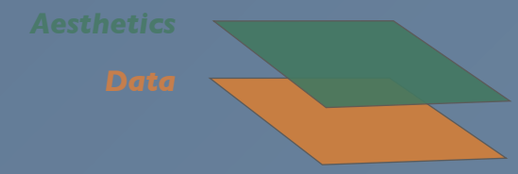
```
ggplot(iris, aes(x = Sepal.Length,
                 y = Sepal.Width,
                 col = Species)) +
  geom_point()
```




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

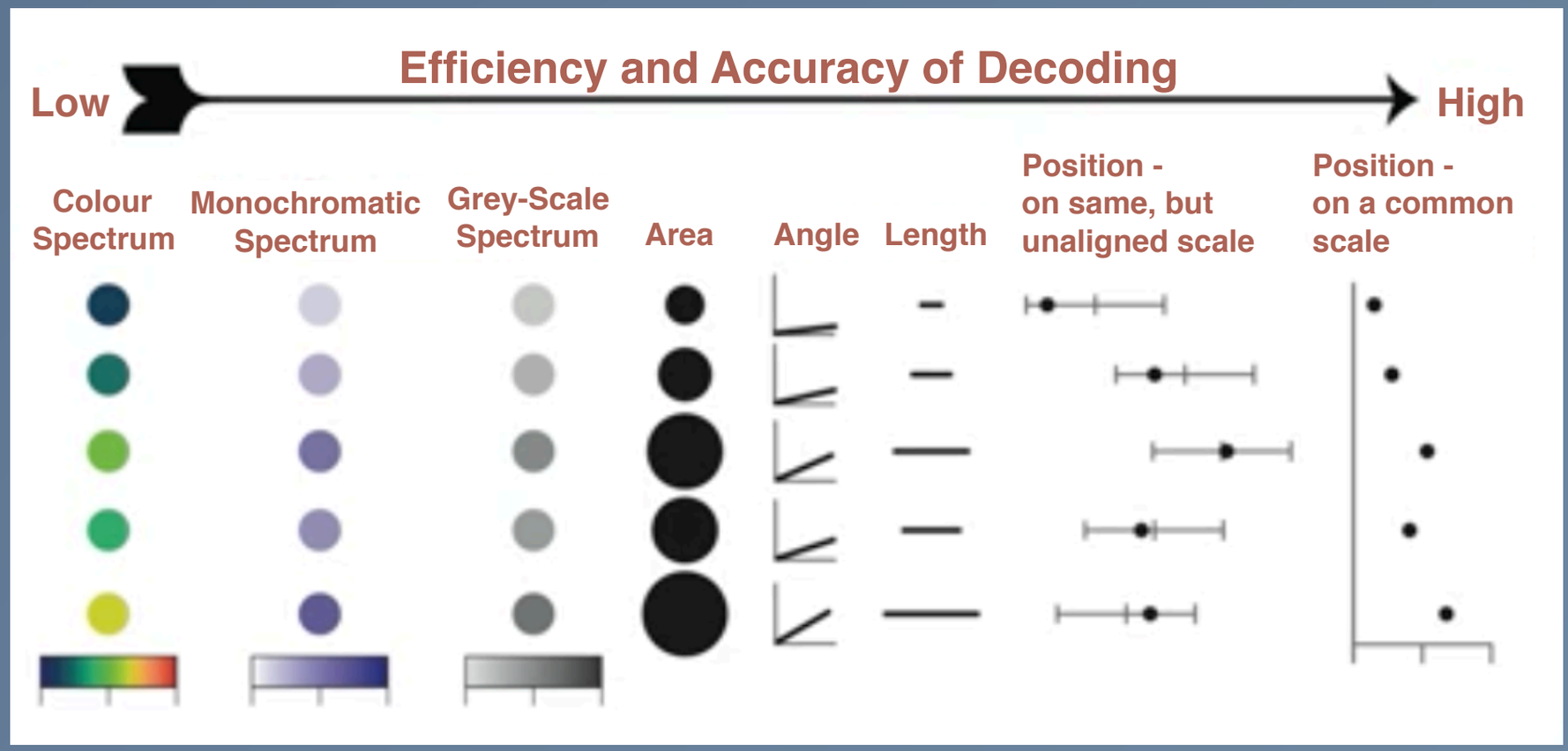


The Aesthetics Layer



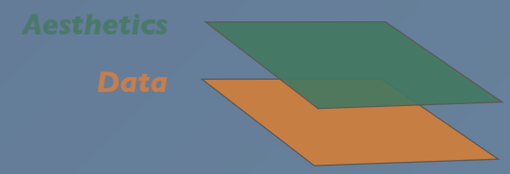
Best practices Different aesthetics work well for different variable types

Numerical variables



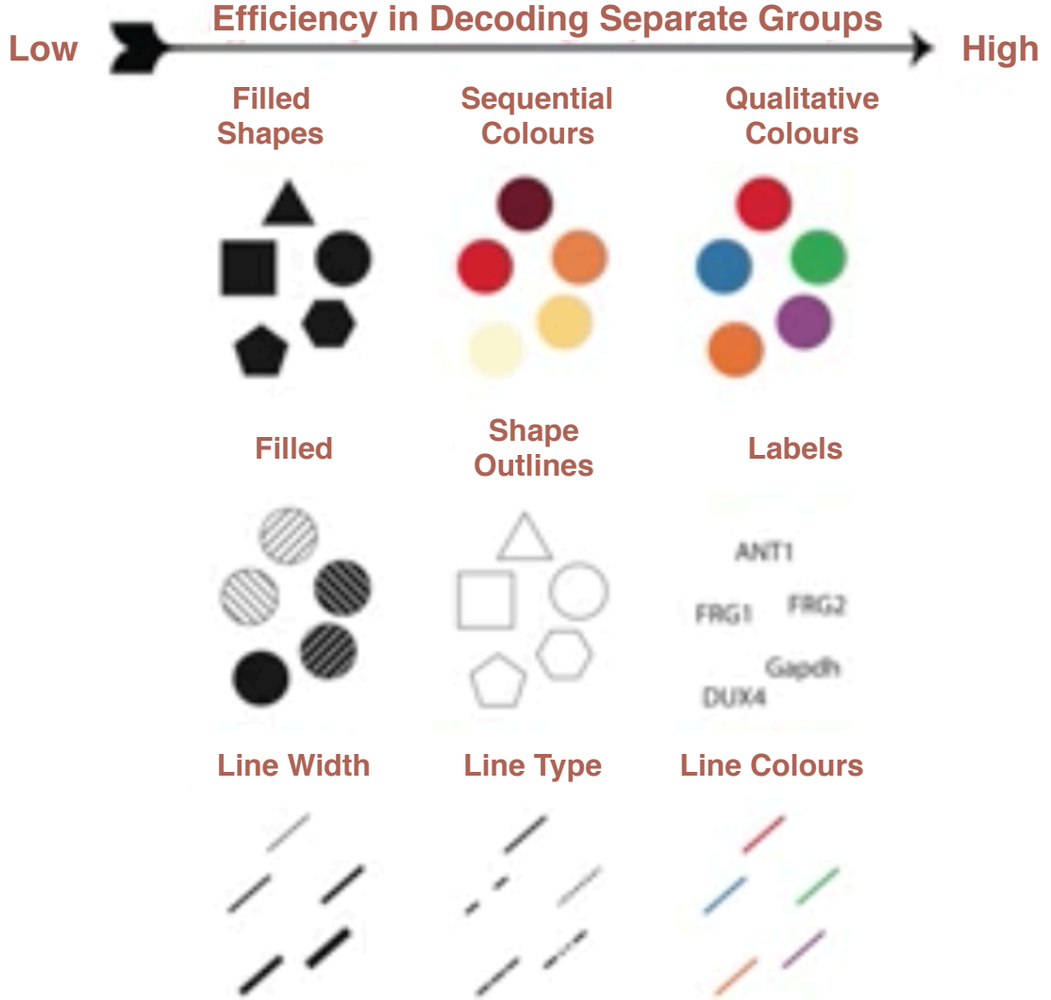
https://www.datacamp.com/courses/data-visualization-with-ggplot2-1

The Aesthetics Layer



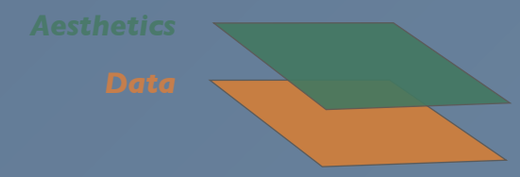
Best practices Different aesthetics work well for different variable types

Categorical variables



https://www.datacamp.com/courses/data-visualization-with-ggplot2-1

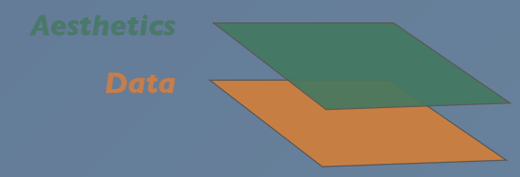
The Aesthetics Layer



Best practices

- Form follows function:
  - Choose your aesthetic mappings based on the information that you want to convey
- Different aesthetics work well for different variable types
- Avoid redundant aesthetic mappings

The Aesthetics Layer



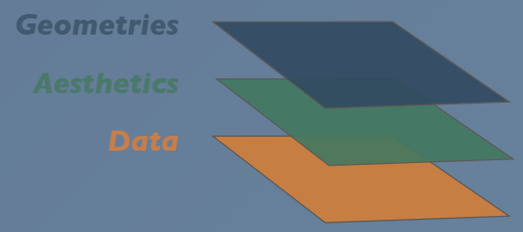
Best practices

- Form follows function:
  - Choose your aesthetic mappings based on the information that you want to convey
- Different aesthetics work well for different variable types
- Avoid redundant aesthetic mappings

Summary

- Aesthetics are the scales onto which variables are mapped
- Aesthetics are different from attributes
- Aesthetics are intimately tied to data structure
- Increasing the number of aesthetic mappings increases graphic complexity but decreases readability

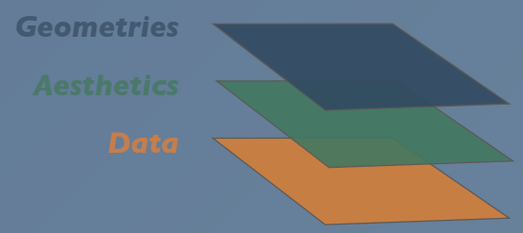
The Geometries Layer



Introduction

- The geometries layer describes the way in which the aesthetic mappings get converted to visual information
- This is where we get the standard plot types
    - Example: Scatter plots, histograms, box plots, etc.
- Each geometry is associated with a specific set of required and optional aesthetics
- There are 37 geometries in ggplot2
- This is the last of the essential layers needed to make a plot

The Geometries Layer



Adding a geometry layer

- Geometries are added to the base `ggplot()` object using the addition operator, +
- All geometry layers begin with `geom_`
- The geometry inherits the aesthetics from the `ggplot()` command

Examples:

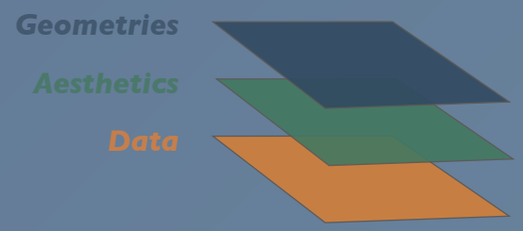
```
ggplot(iris, aes(x = Sepal.Length,
                 y = Sepal.Width)) +
  geom_point()

ggplot(iris, aes(x = Sepal.Length)) +
  geom_histogram()
```

ggplot2 geometries

abline	density2d	line	rect	vline
area	dotplot	linerange	ribbon	
bar	errorbar	map	rug	
bin2d	errorbarh	path	segment	
blank	freqpoly	point	smooth	
boxplot	hex	pointrange	step	
contour	histogram	polygon	text	
crossbar	hline	quantile	tile	
density	jitter	raster	violin	

https://www.datacamp.com/courses/data-visualization-with-ggplot2-1

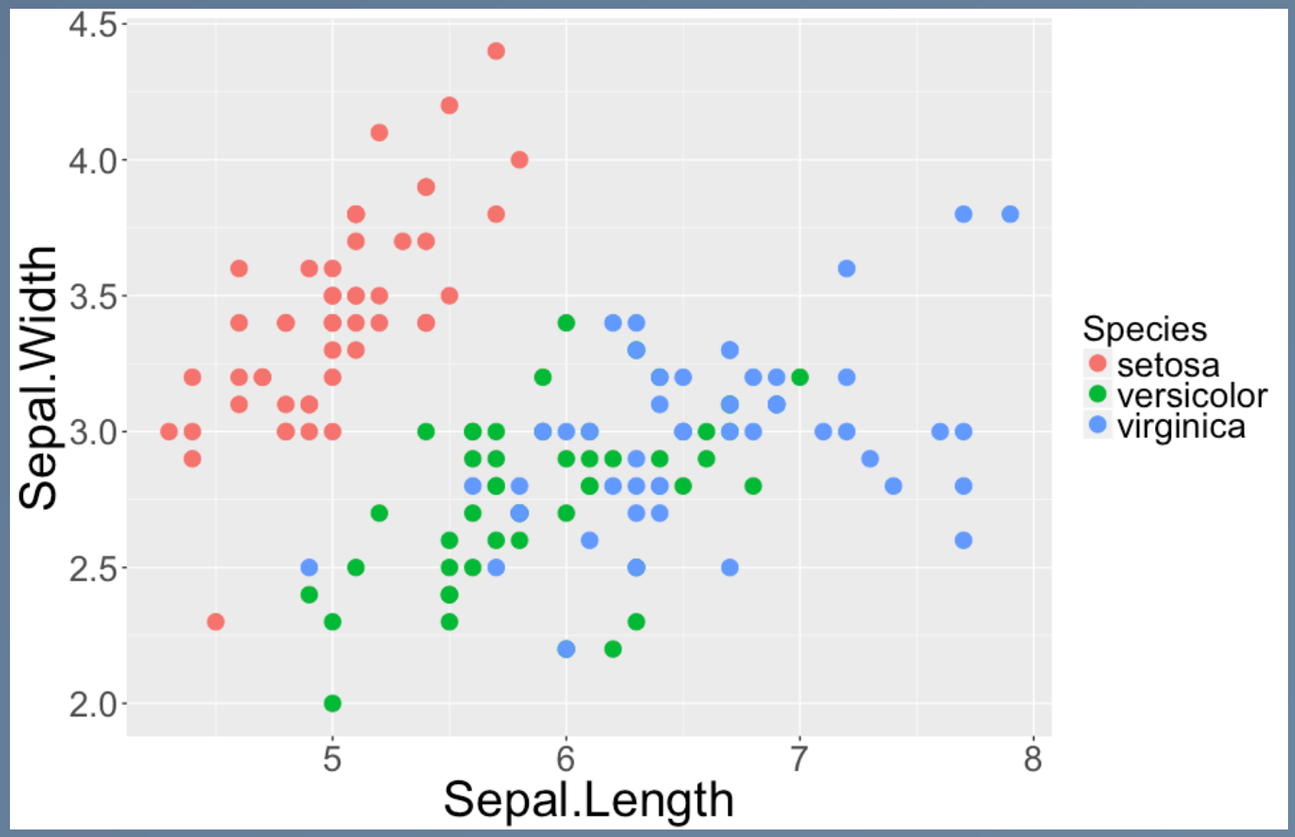
The Geometries Layer



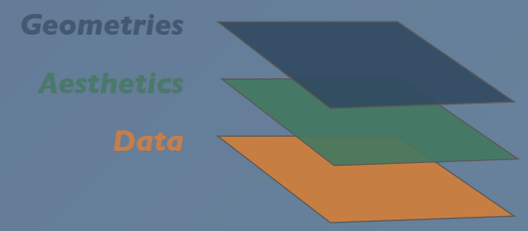
Standard plots: Scatter plots

- Scatter plots are created using `geom_point()`
- `geom_point()` requires aesthetics `x` and `y`

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_point(size = 4)
```



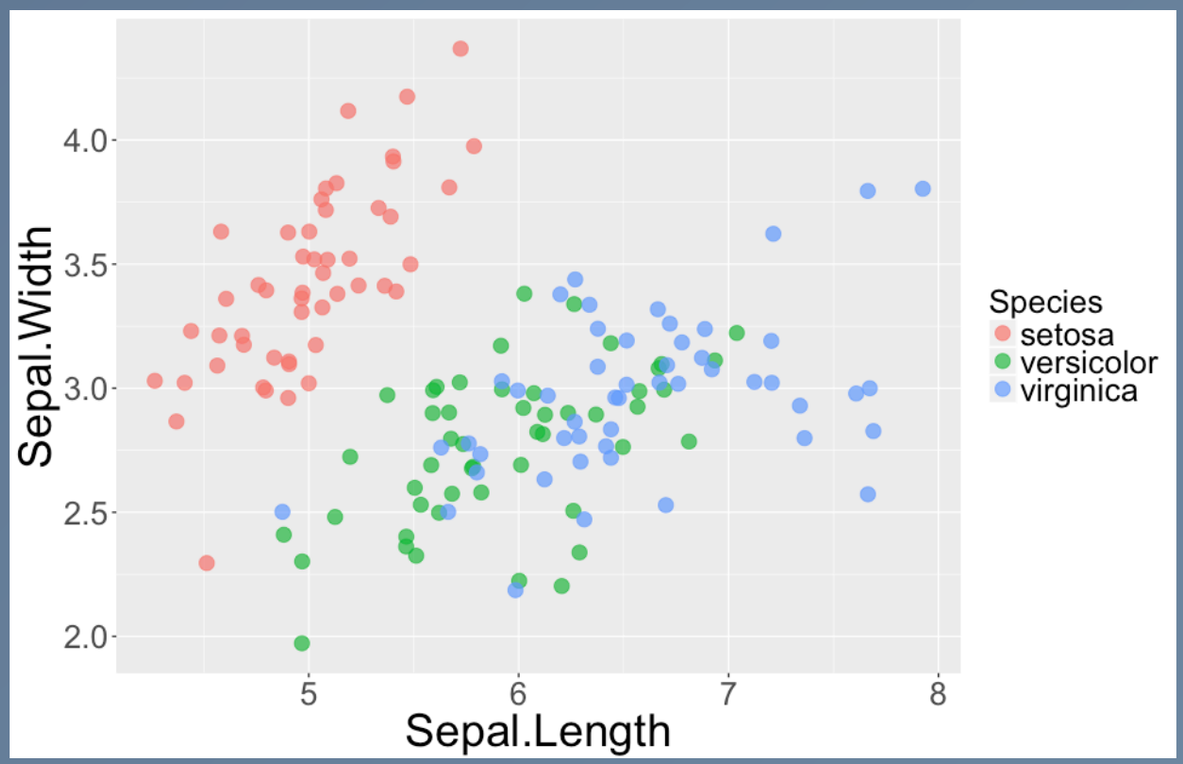
The Geometries Layer



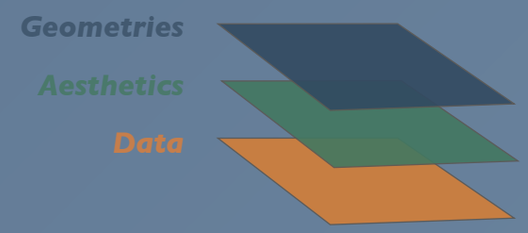
Standard plots: Scatter plots

- Notice that the iris data has multiple overlapping data points, due to the precision of the measurements
- We can add a "jitter", i.e. random noise, to the data points by using the `position` argument in the geometry layer.

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
geom_point(position = "jitter", size = 4, alpha = 0.7)
```



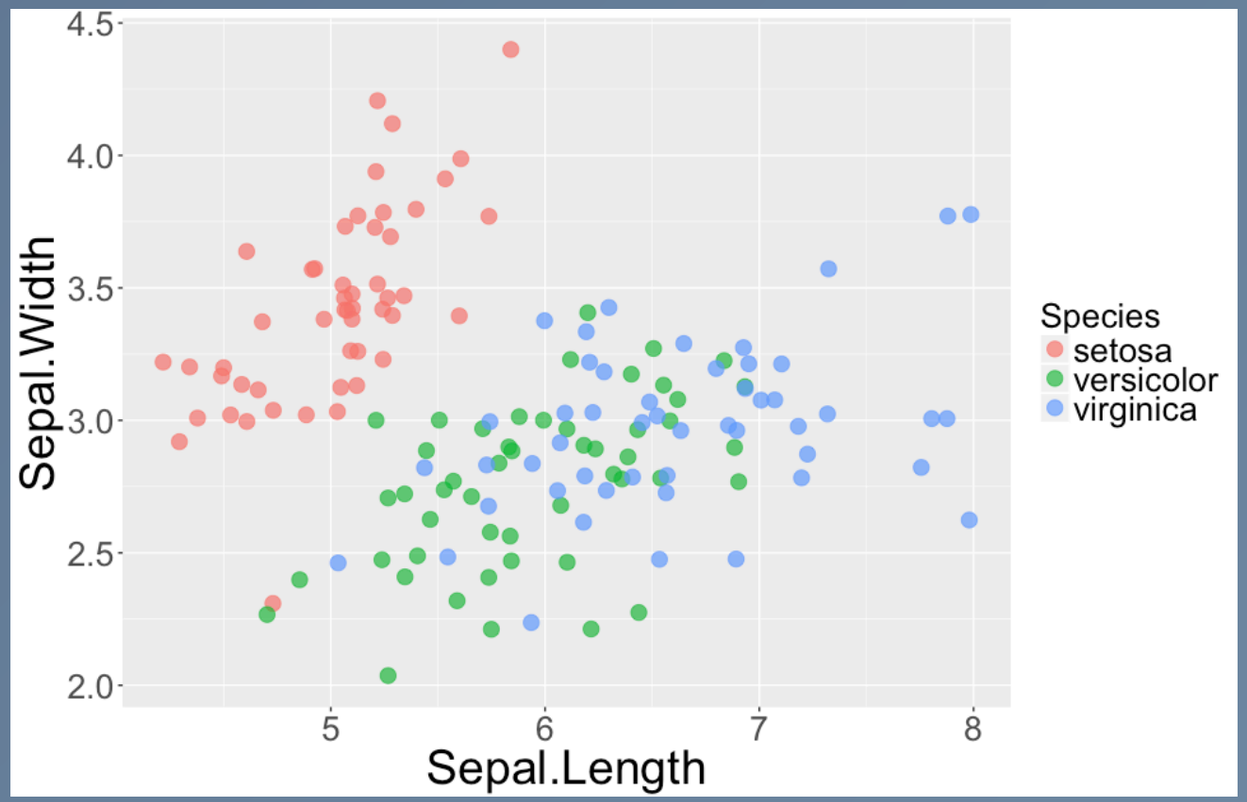
The Geometries Layer



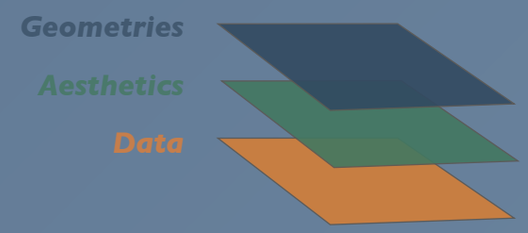
Standard plots: Scatter plots

- This can also be accomplished with `geom_jitter()`
- The amount of jitter is determined using the `width` argument

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_jitter(width = 0.3, size = 4, alpha = 0.7)
```



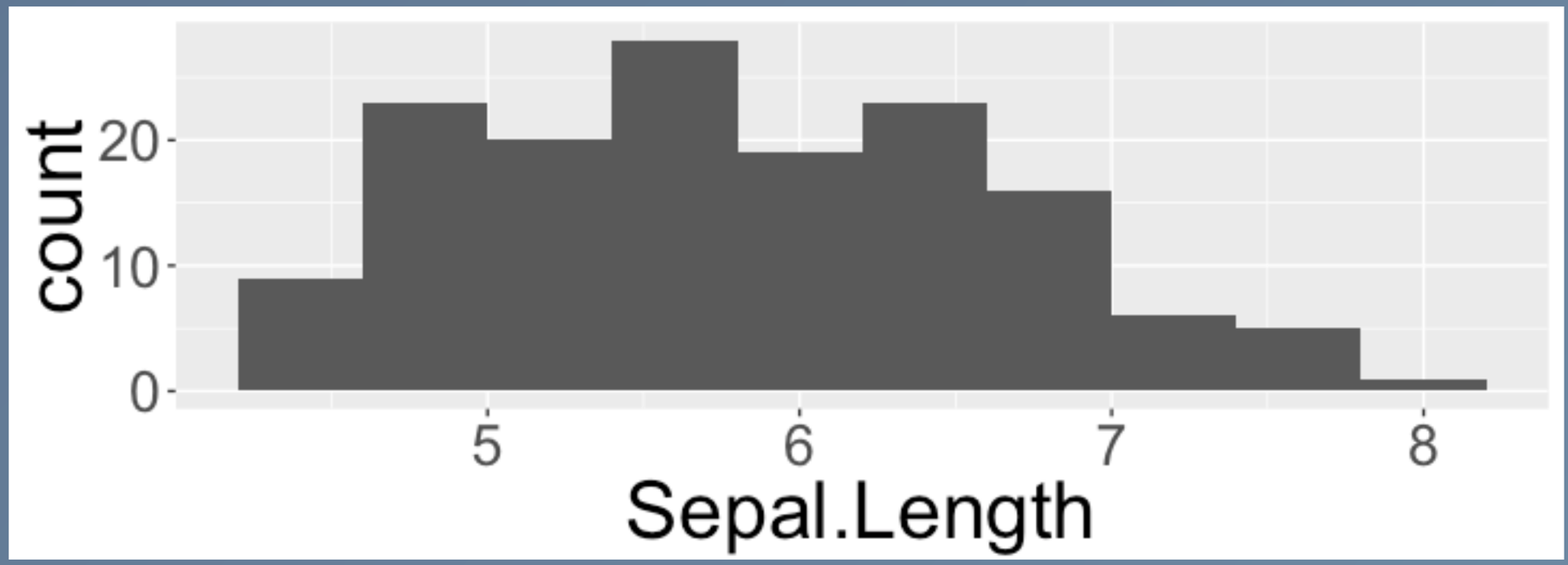
The Geometries Layer



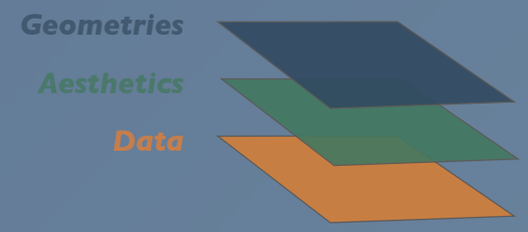
Standard plots: Histograms

- Histograms are created using `geom_histogram()`
- `geom_histogram()` requires only an `x` aesthetic
- By default, `geom_histogram()` uses 30 bins. This is controlled using the `bins` or `binwidth` arguments

```
ggplot(iris, aes(x = Sepal.Length)) + geom_histogram(bins = 10)
```



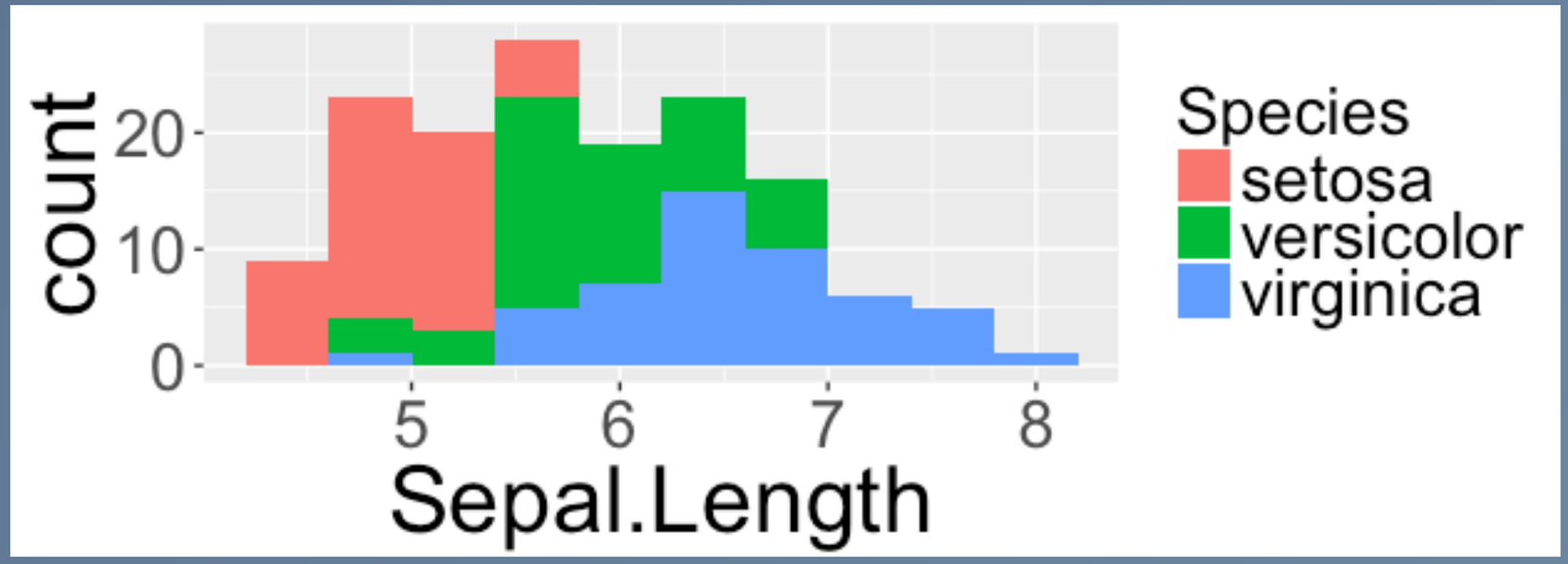
The Geometries Layer



Standard plots: Histograms

Using the fill aesthetic with `geom_histogram()` can be tricky:

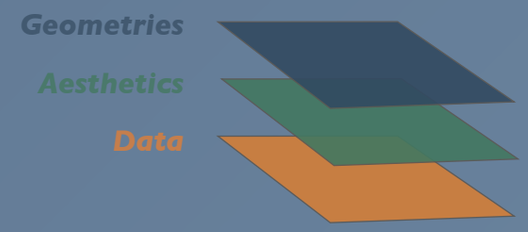
```
ggplot(iris, aes(x = Sepal.Length, fill = Species)) +
  geom_histogram(bins = 10)
```



It isn't clear whether there are three overlapping histograms or if the bars are stacked in each bin.

What do you think?

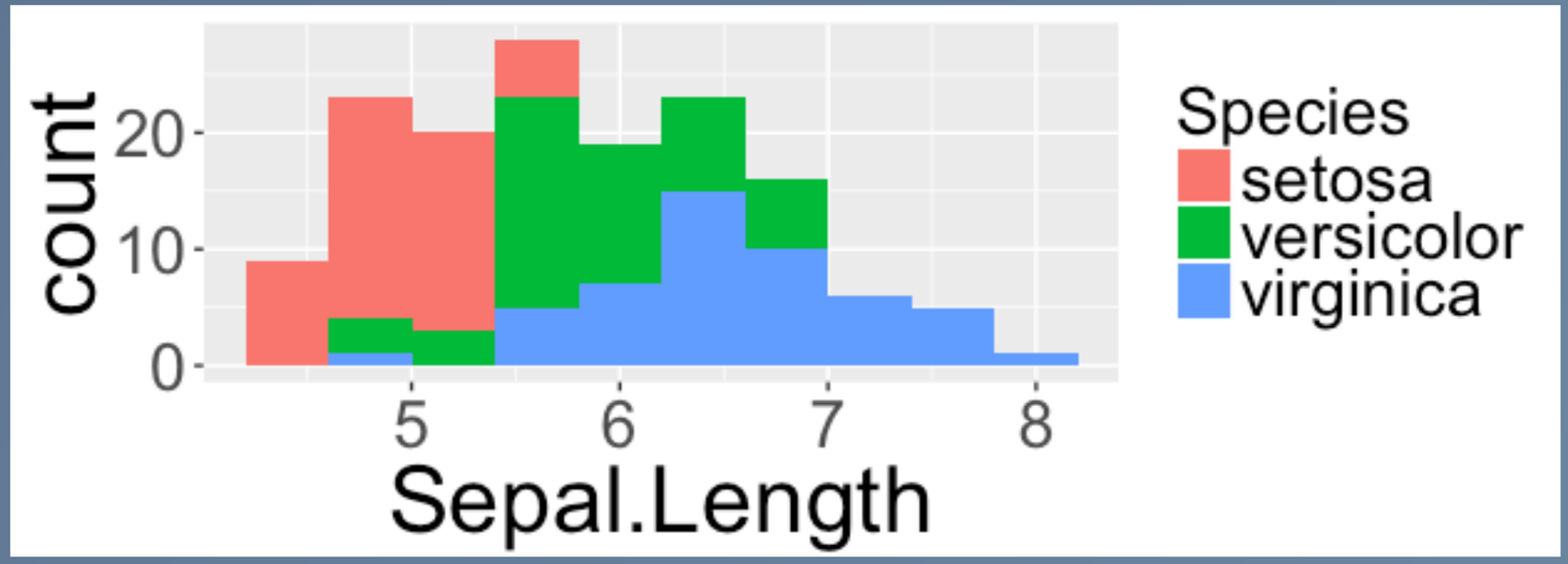
The Geometries Layer



Standard plots: Histograms

Using the fill aesthetic with `geom_histogram()` can be tricky:

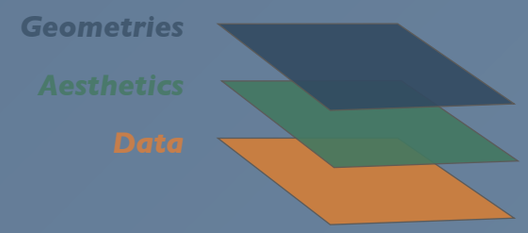
```
ggplot(iris, aes(x = Sepal.Length, fill = Species)) +
  geom_histogram(bins = 10)
```



It isn't clear whether there are three overlapping histograms or if the bars are stacked in each bin.

In fact the bars are stacked by default.

The Geometries Layer

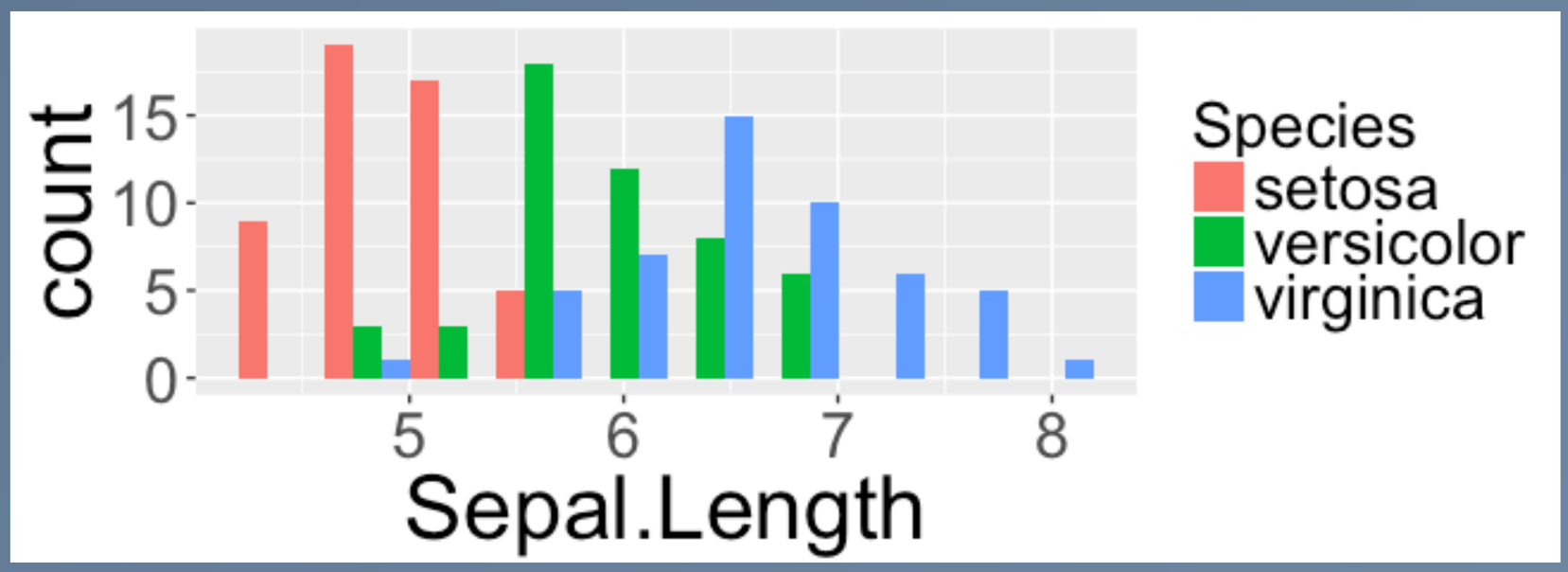


Standard plots: Histograms

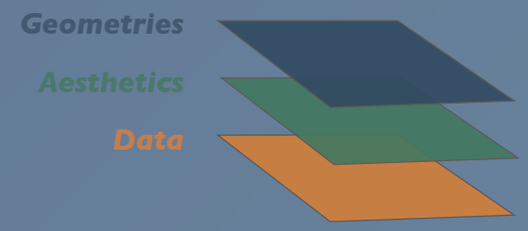
This is regulated with the `position` argument. The default is "stacked".

We can also "dodge" the bars in each bin:

```
ggplot(iris, aes(x = Sepal.Length, fill = Species)) +
geom_histogram(bins = 10, position = "dodge")
```



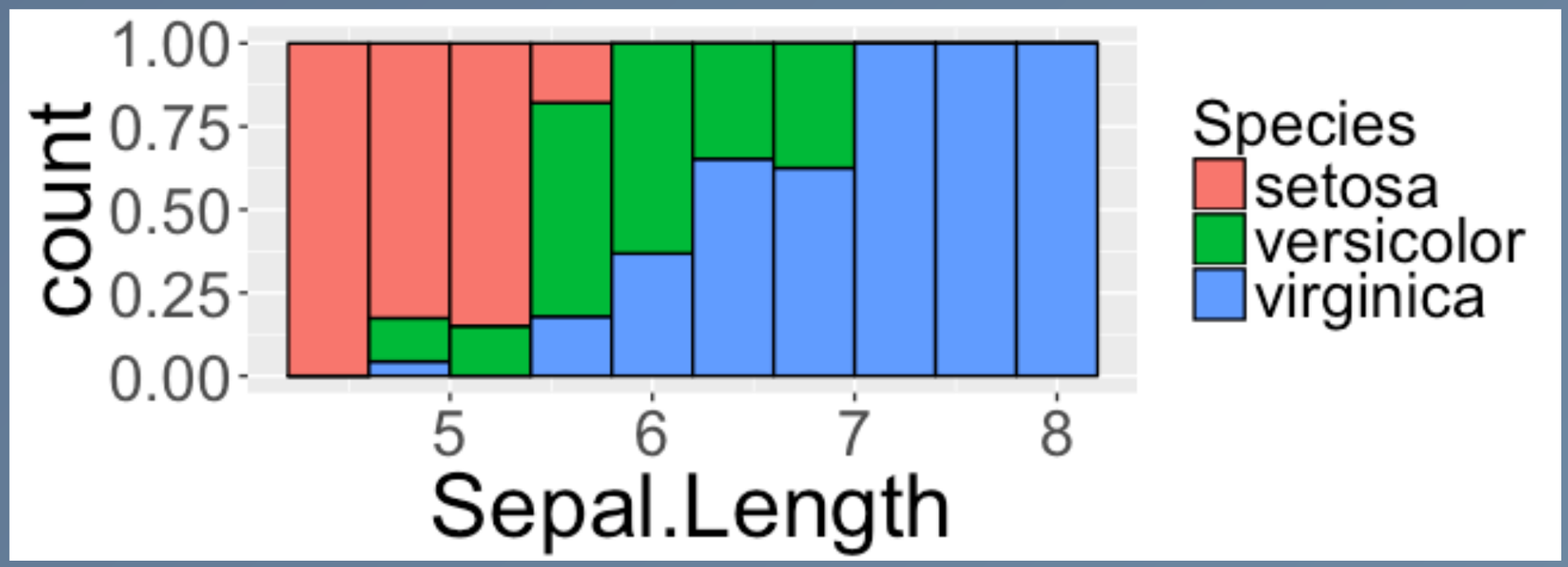
The Geometries Layer



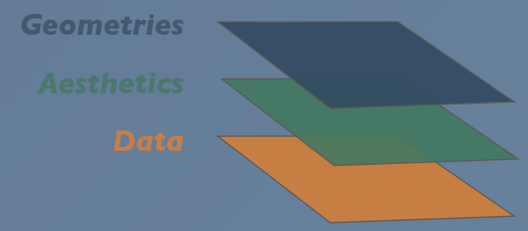
Standard plots: Histograms

Another approach is to represent the proportion of the different groups in each bin. This is done using "fill".

```
ggplot(iris, aes(x = Sepal.Length, fill = Species)) +
geom_histogram(bins = 10, position = "fill", col = "black")
```



The Geometries Layer

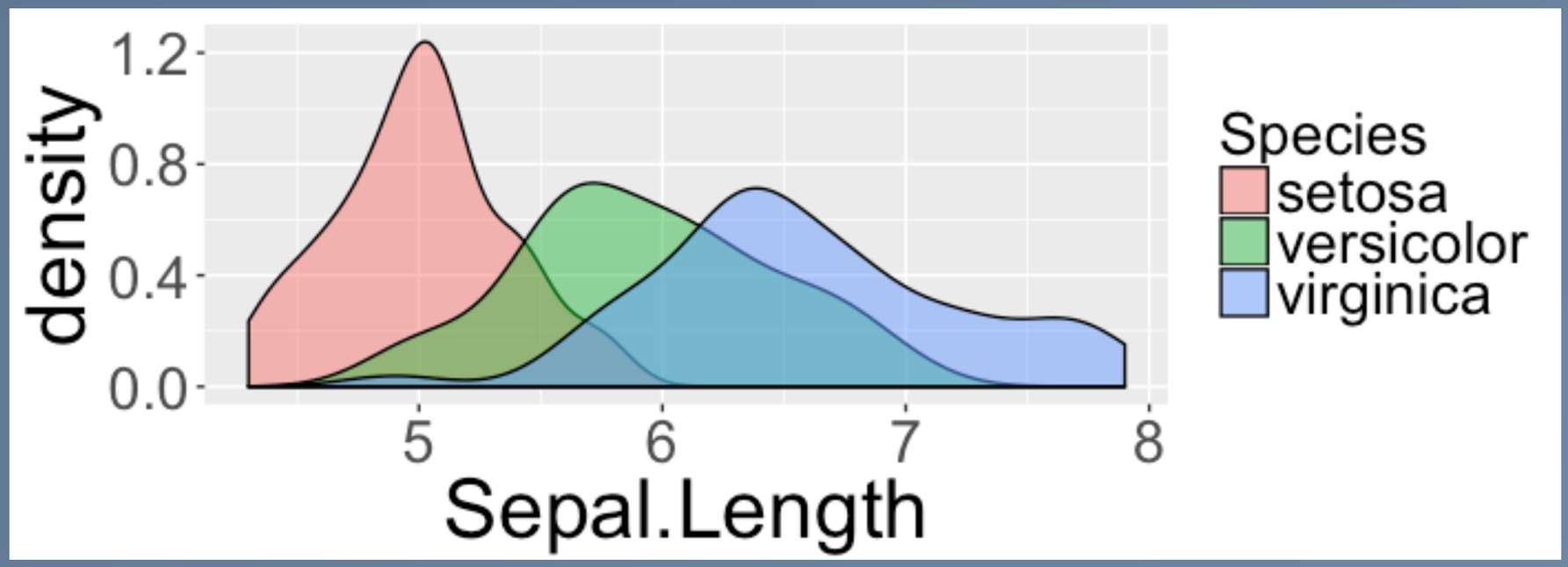


Standard plots: Histograms

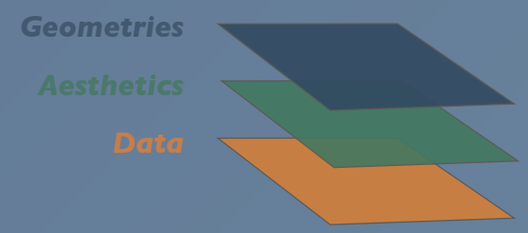
A better way to represent overlapping distributions is to plot their estimated probability density functions.

This can be done with `geom_density()`.

```
ggplot(iris, aes(x = Sepal.Length, fill = Species)) +
geom_density(alpha = 0.5)
```



The Geometries Layer

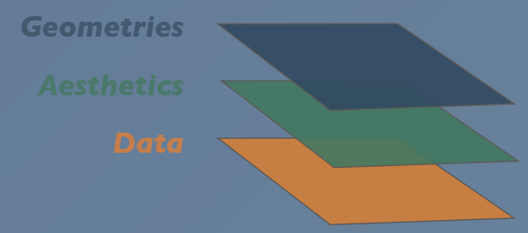


Standard plots: Bar plots

- Bar plots are created using `geom_bar()`
- `geom_bar()` requires only an `x` aesthetic
- The variable mapped to `x` must be categorical
- By default, `geom_bar()` counts the number of observations in each group

```
ggplot(iris, aes(x = Species)) + geom_bar()
```

The Geometries Layer



Standard plots: Bar plots

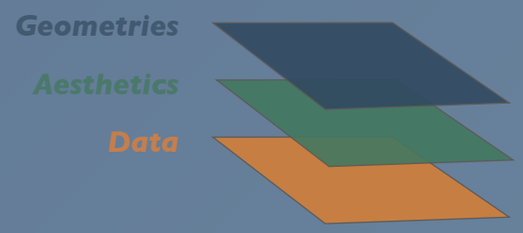
Suppose we have summary statistics that we want to display using a dynamite plot:

```
iris_summary <- iris %>%
  group_by(Species) %>%
summarise(avg = mean(Sepal.Length),
          stdev = sd(Sepal.Length))
```

	Species	avg	stdev
1	setosa	5.006	0.3524897
2	versicolor	5.936	0.5161711
3	virginica	6.588	0.6358796

How do we make this plot?

The Geometries Layer



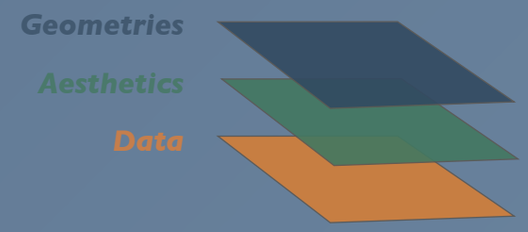
Standard plots: Bar plots

First we have to change the counting statistics underlying `geom_bar()`.This is done using the `stat` argument.

We also now need to specify a y aesthetic to use instead of counts.

```
ggplot(iris_summary, aes(x = Species, y = avg)) +
  geom_bar(stat = "identity")
```

The Geometries Layer

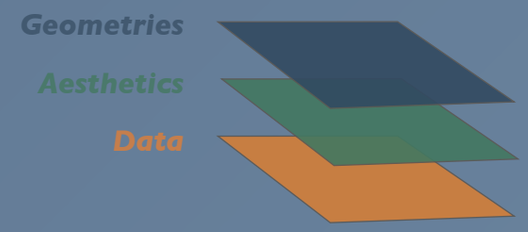


Standard plots: Bar plots

We can display the error bars by adding a new geometry with `geom_errorbar()`:

```
ggplot(iris_summary, aes(x = Species, y = avg)) +
geom_bar(stat = “identity") +
geom_errorbar()
```

The Geometries Layer



Standard plots: Bar plots

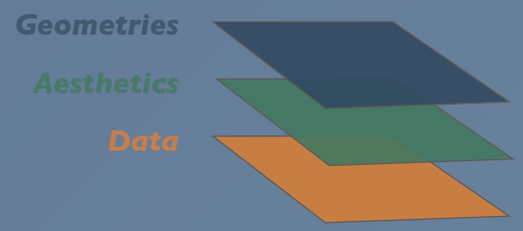
We can display the error bars by adding a new geometry with `geom_errorbar()`:

```
ggplot(iris_summary, aes(x = Species, y = avg)) +
  geom_bar(stat = “identity") +
  geom_errorbar()
```

This will throw an error.

Why?

The Geometries Layer



Standard plots: Bar plots

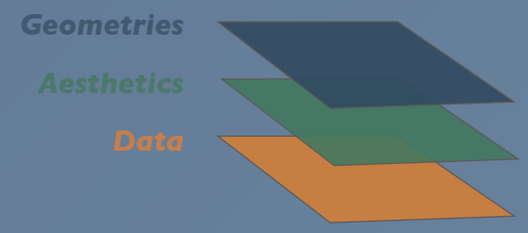
Geometries inherit the aesthetics from the aesthetic layer, but certain geometries require specific aesthetics.

`geom_errorbar()` requires aesthetics ymin and ymax.

We can overwrite the basic aesthetic mapping by calling `aes()` inside the geometry layer.

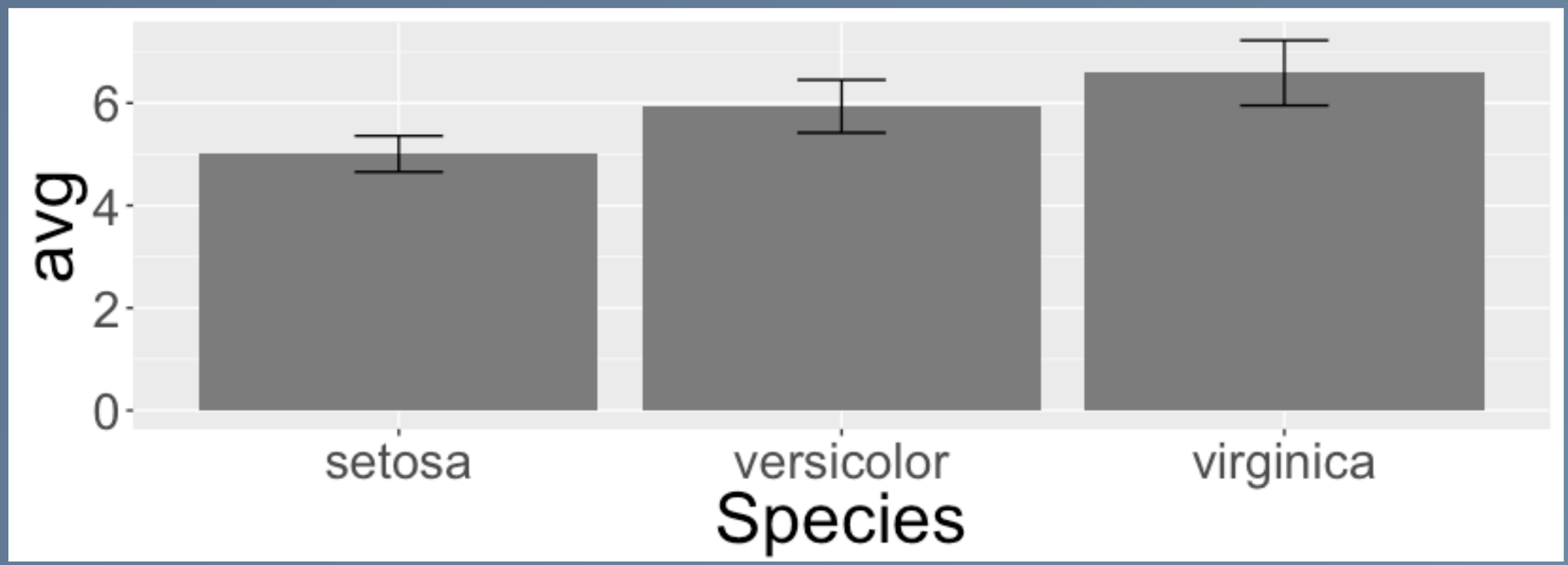
```
ggplot(iris_summary, aes(x = Species, y = avg)) +
  geom_bar(stat = "identity") +
  geom_errorbar(aes(ymin = avg-stdev, ymax = avg+stdev))
```

The Geometries Layer

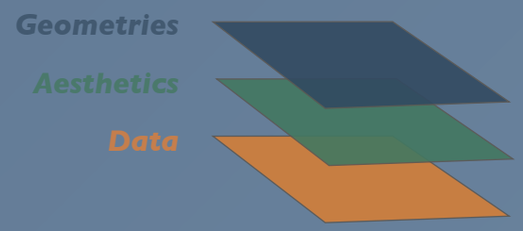


Standard plots: Bar plots

```
ggplot(iris_summary, aes(x = Species, y = avg)) +
  geom_bar(stat = "identity",
           fill = "grey50") +
  geom_errorbar(aes(ymin = avg-stdev, ymax = avg+stdev),
                width = 0.2)
```



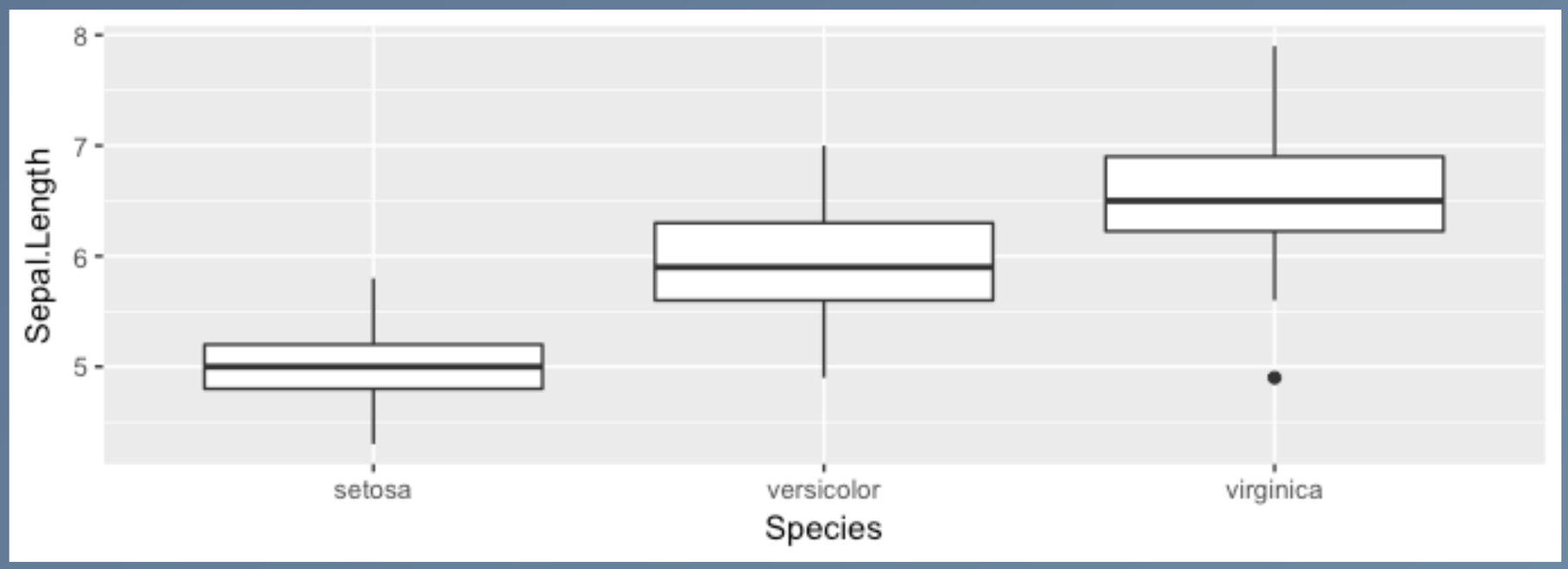
The Geometries Layer



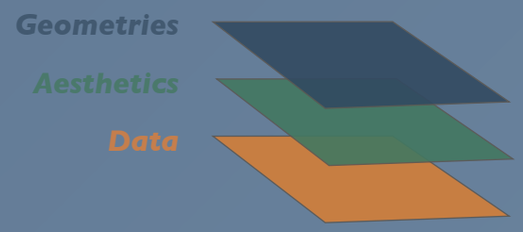
Standard plots: Box plots

- Box plots are created using `geom_boxplot()`
- Required aesthetics included x and y
- The variable mapped to x must be categorical

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
geom_boxplot()
```



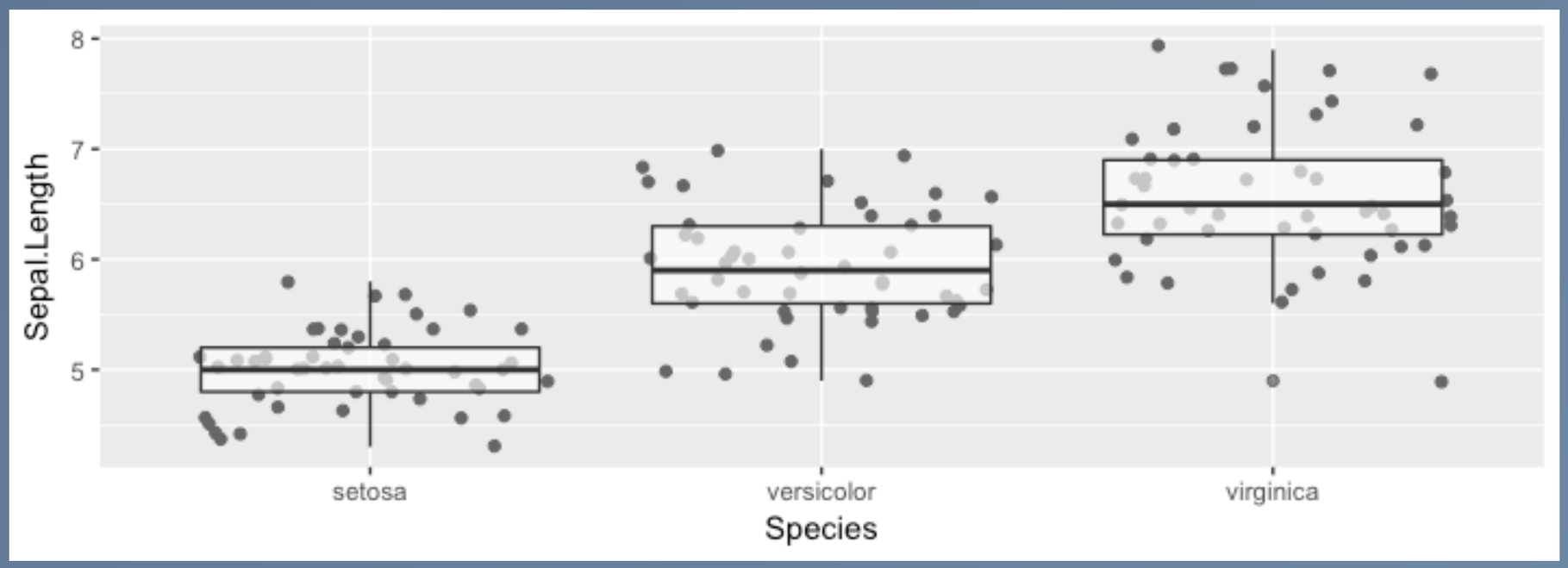
The Geometries Layer



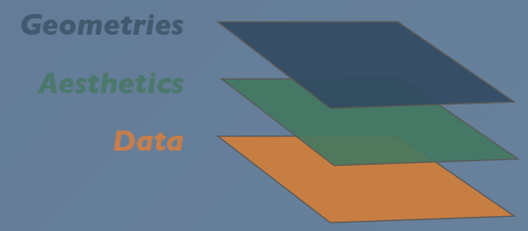
Standard plots: Box plots

It is good practice to display the data points in a box plot:

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_jitter(fill = “grey40”) +
  geom_boxplot(alpha = 0.6)
```



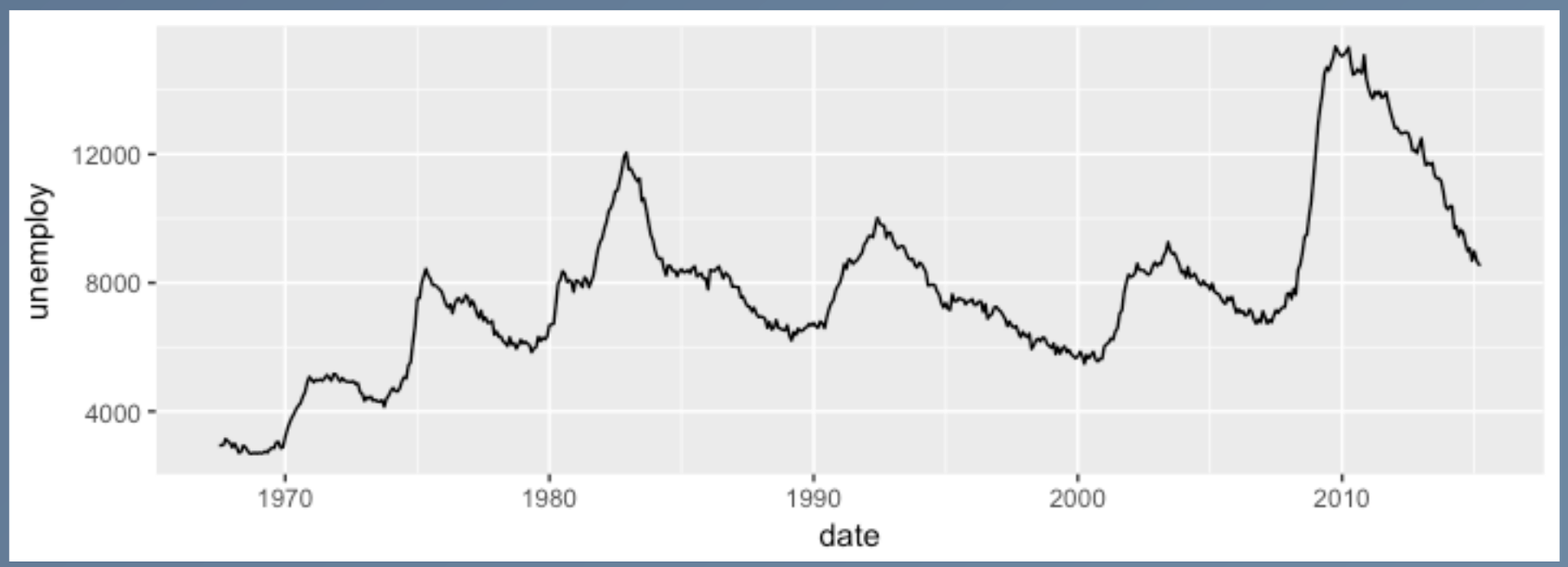
The Geometries Layer



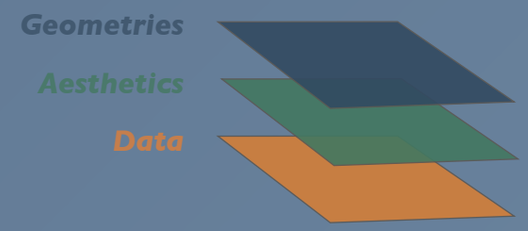
Standard plots: Line plots

- Line plots are useful when working with time series data
- The associated geometry is geom_line()
- The required aesthetics are x and y

```
ggplot(economics, aes(x = date, y = unemploy)) +
  geom_line()
```



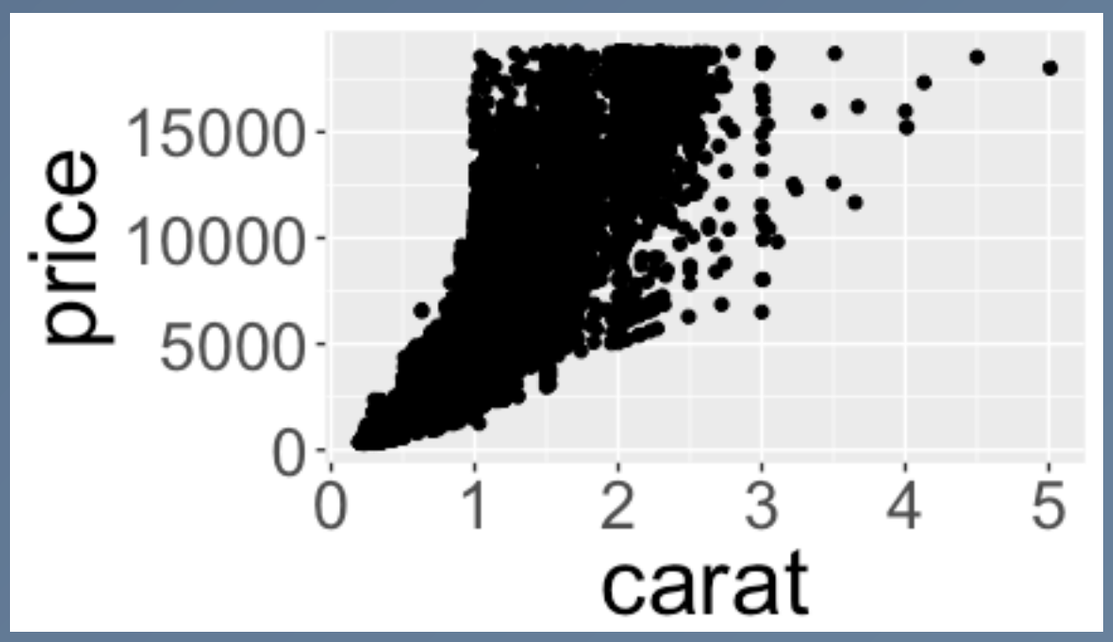
The Geometries Layer



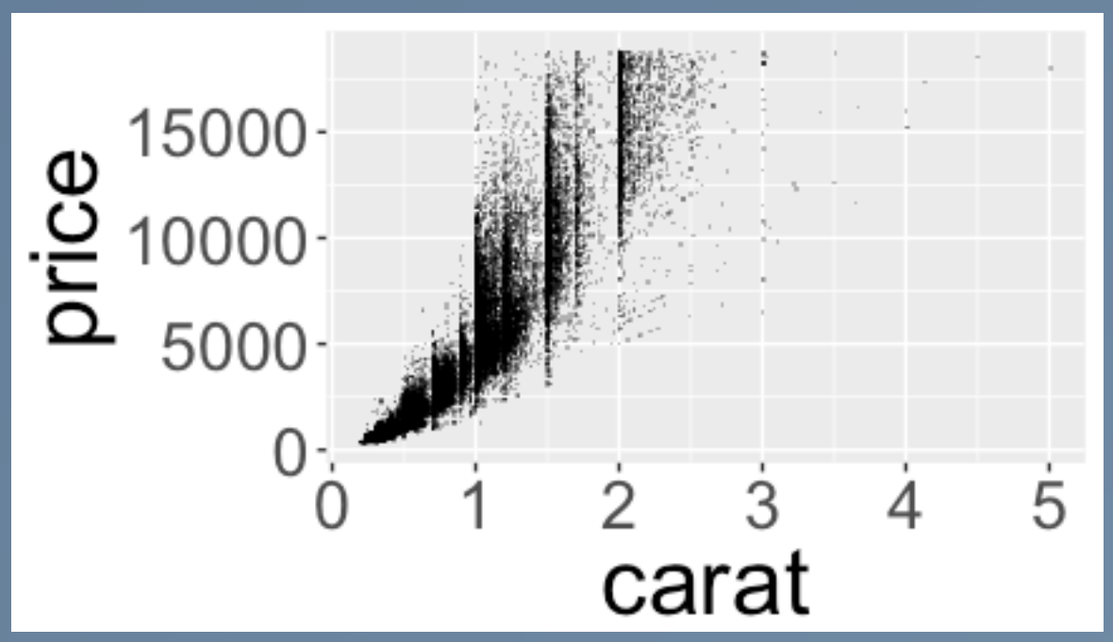
Additional remarks

It's always a good idea to optimize the attributes of a geometry layer for the data that you are plotting

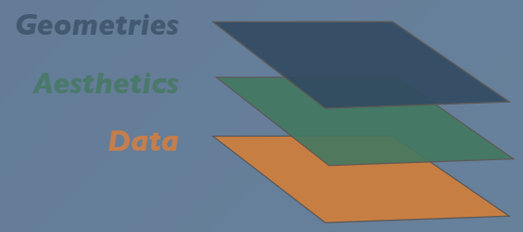
```
ggplot(diamonds, aes(x = carat,
                     y = price)) +
  geom_point()
```




```
ggplot(diamonds, aes(x = carat,
                     y = price)) +
  geom_point(alpha = 0.3,
             shape = ".")
```



The Geometries Layer



Summary

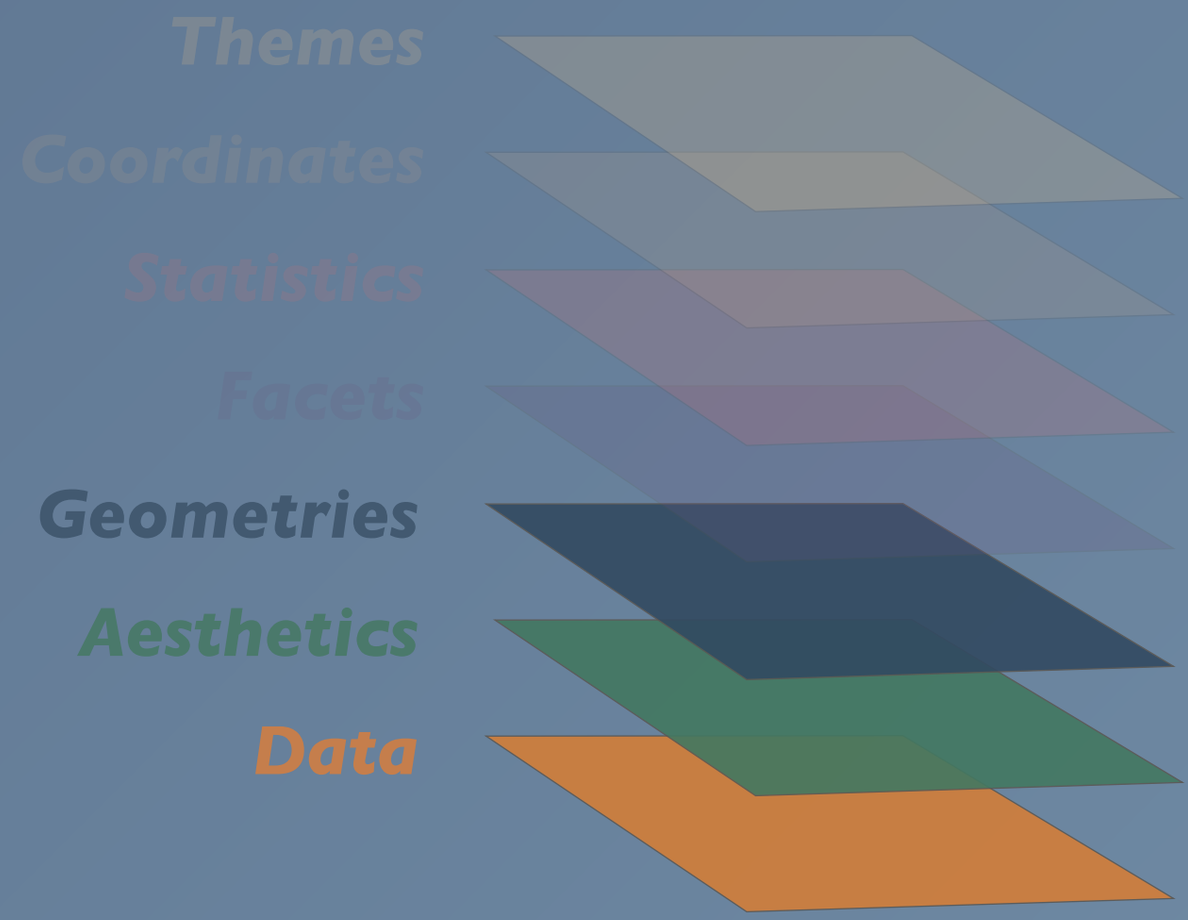
- Geometries display the information defined by the aesthetic mappings
- There are 37 geometries in ggplot2
    - `geom_point()`, `geom_jitter()`
    - `geom_histogram()`, `geom_density()`
    - `geom_bar()`, `geom_errorbar()`
    - `geom_boxplot()`
- Each geometry requires specific aesthetics
- Additional aesthetic mappings can be defined within the geometries layer
- Attributes are used to refine the appearance of geometric visual elements

The Grammar of Graphics — Revisited

The three grammatical elements required to generate a plot are:

- Data
- Aesthetics
- Geometries

The four remaining layers are optional but can be used to enhance the information content.



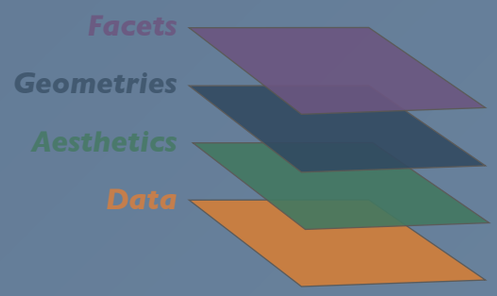
The Grammar of Graphics — Revisited

The three grammatical elements required to generate a plot are:

- Data
- Aesthetics
- Geometries

The four remaining layers are optional but can be used to enhance the information content.

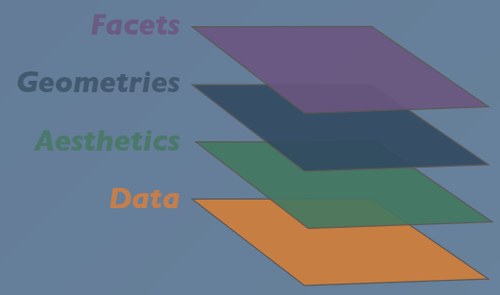
The Facets Layer



Introduction

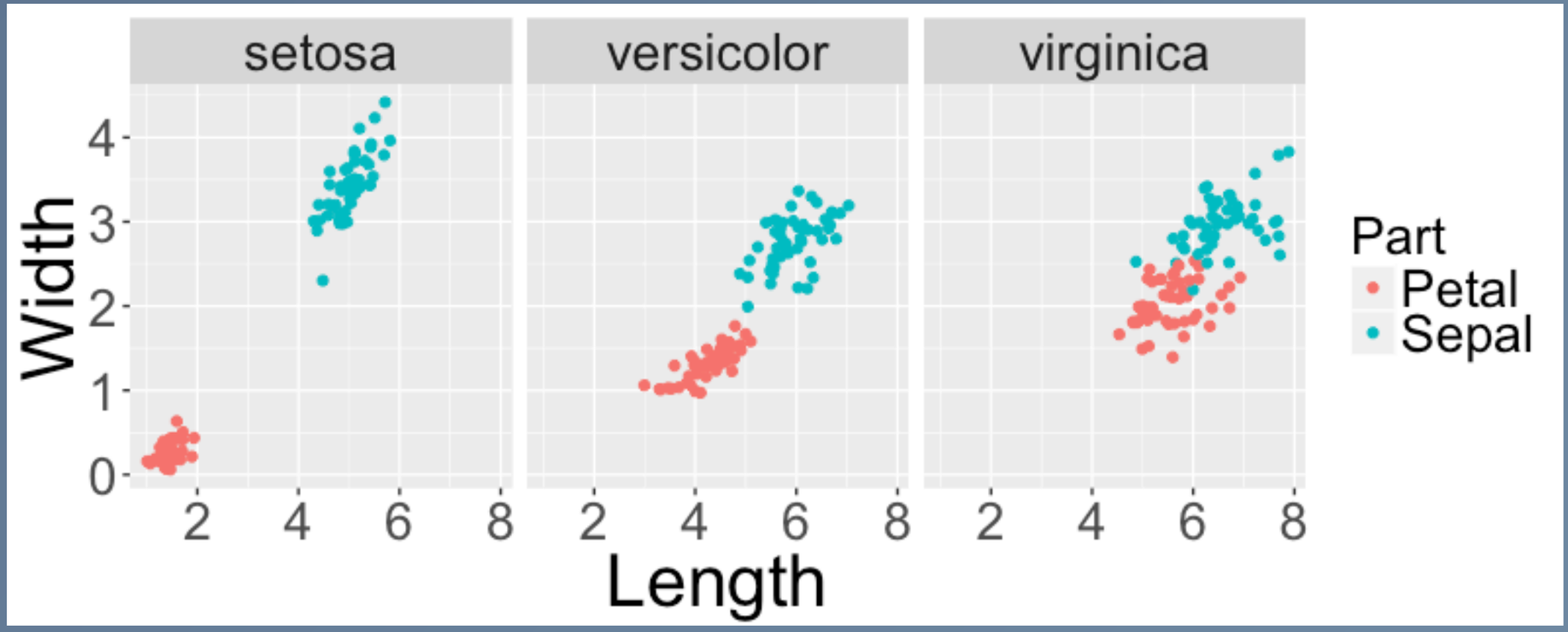
- The facets layer is used to split a plot into subplots, i.e. facets, according to the levels of a factor
- Facets can be organized into rows, columns or both
- The commands to create a facet layer are `facet_grid()` and `facet_wrap()`
- The main purpose of the facets layer is to add variables to your graphic

The Facets Layer

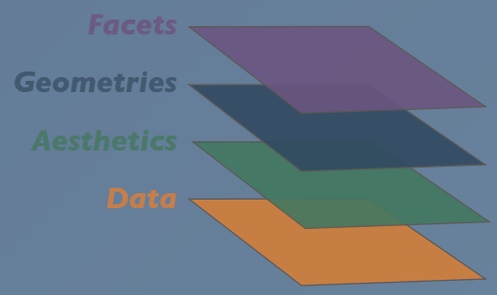


Organizing facets into columns

```
ggplot(iris.wide, aes(x = Length, y = Width, col = Part)) +
  geom_jitter() +
  facet_grid(.~Species)
```

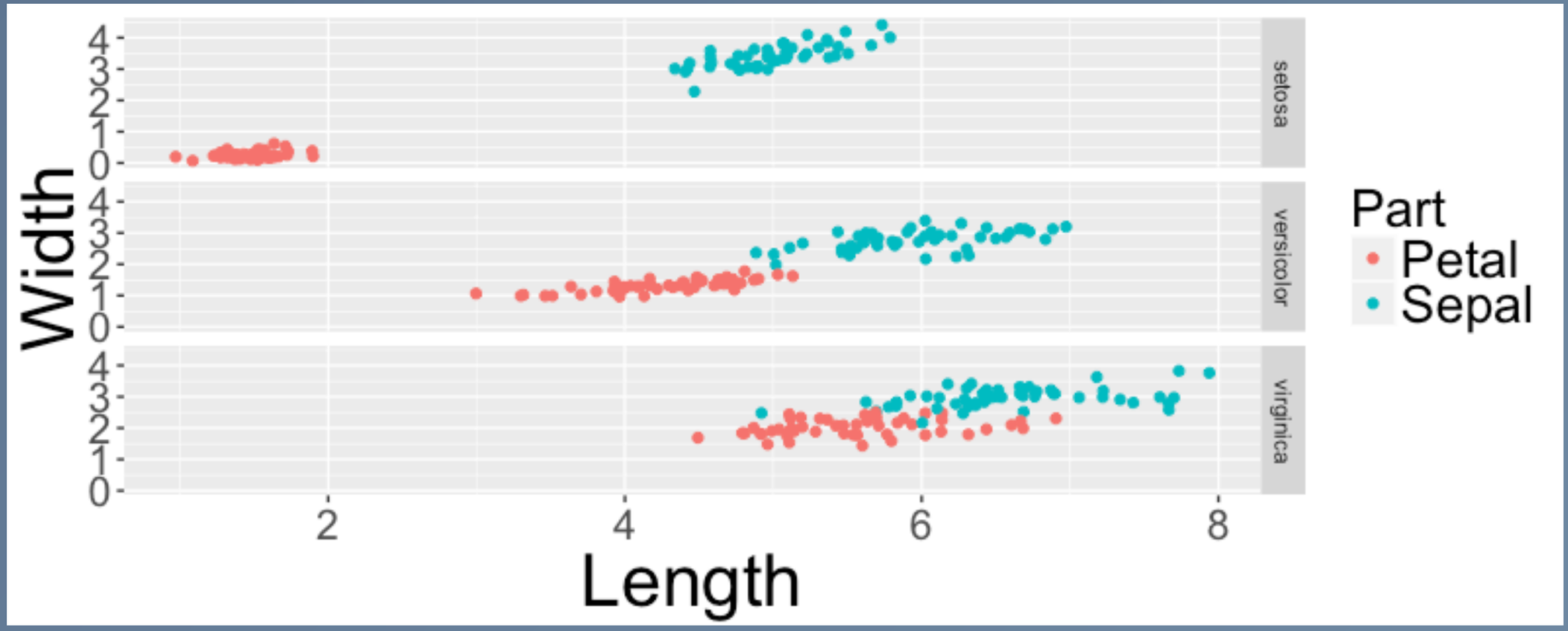


The Facets Layer

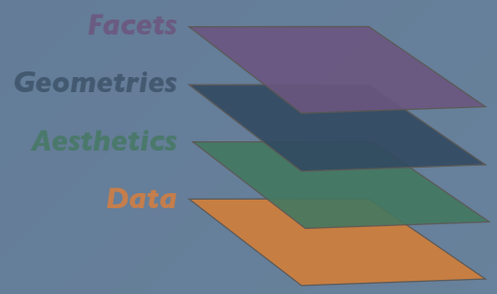


Organizing facets into rows

```
ggplot(iris.wide, aes(x = Length, y = Width, col = Part)) +
  geom_jitter() +
  facet_grid(Species~.)
```

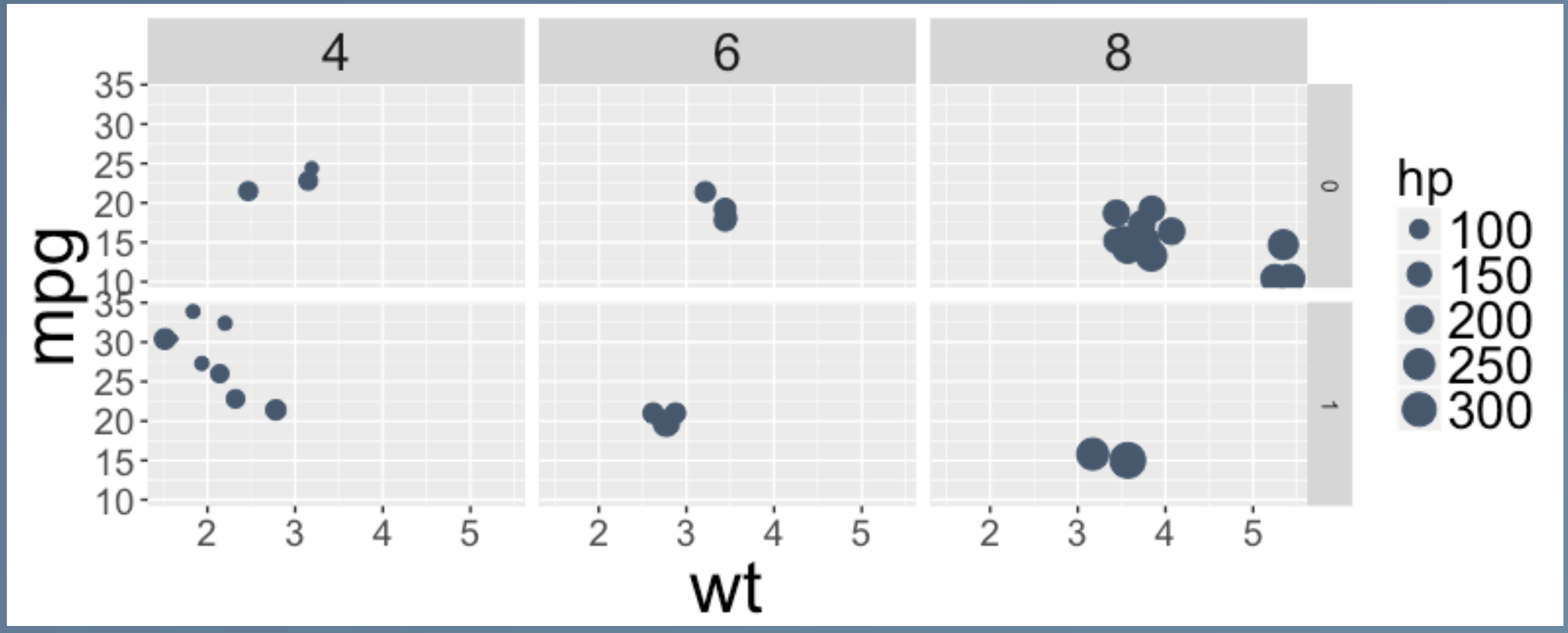


The Facets Layer

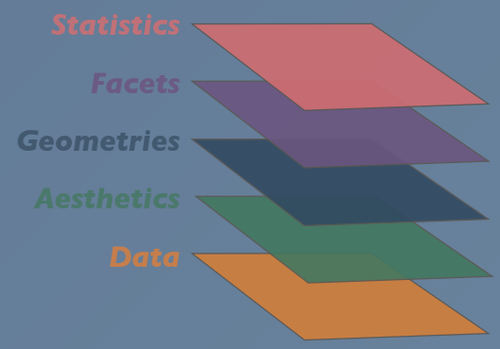


Faceting according to two variables

```
ggplot(mtcars, aes(x = wt, y = mpg, size = hp)) +
  geom_point(col = “#475970") +
  facet_grid(am ~ cyl)
```



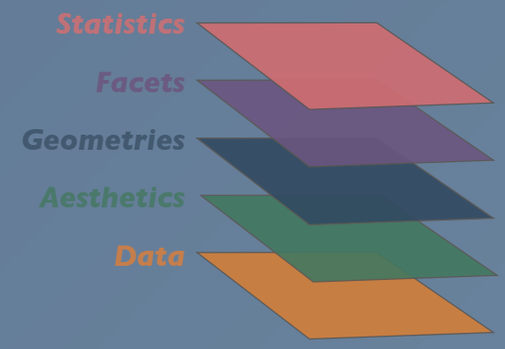
The Statistics Layer



Introduction

- The statistics layer is used to compute and display summary statistics or statistical models on the fly
- Statistics can be accessed with certain geometries or on their own using `stat_` functions
- Many geometries use implicit statistics
    - Example: `geom_histogram()` uses `stat_bin()` under the hood to bin the data

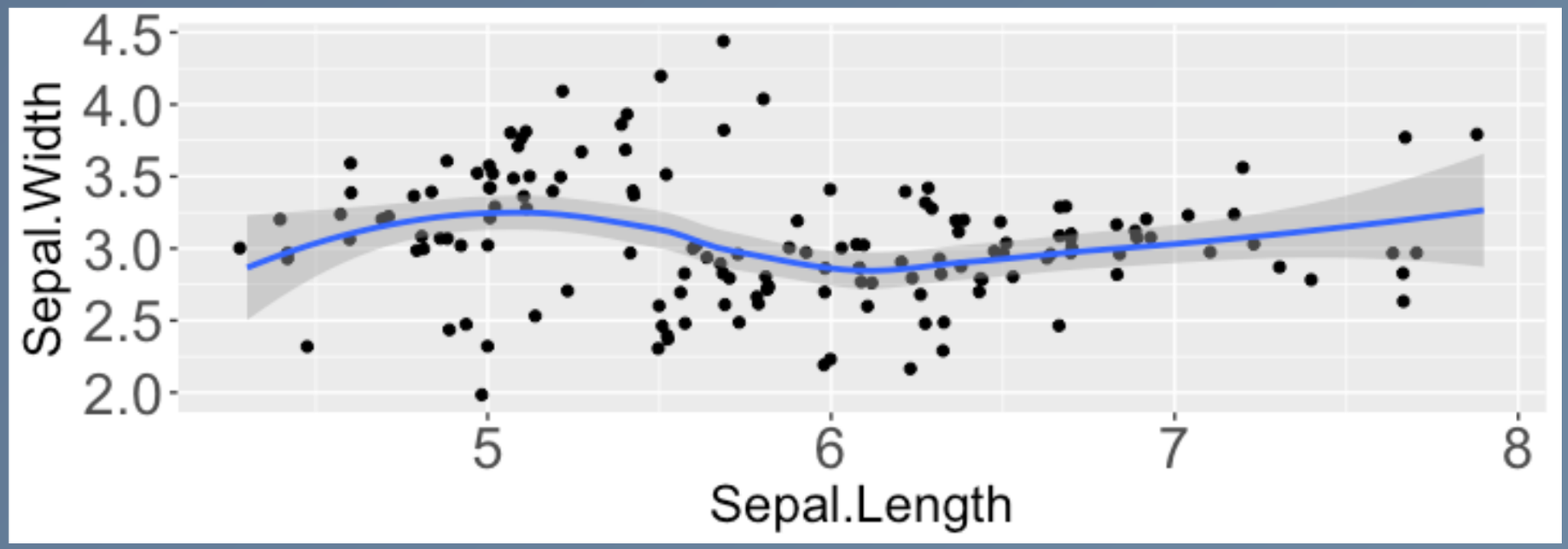
The Statistics Layer



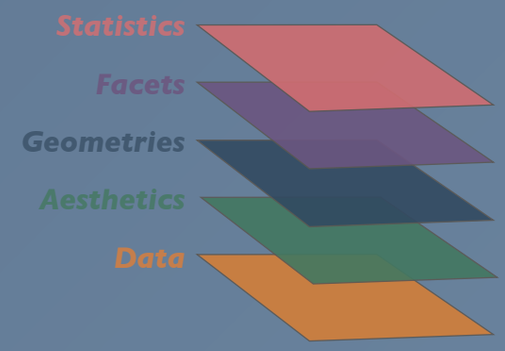
Fitting statistical models on the fly

Statistical models can be fit to the data using geom_smooth() or stat_smooth()

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_jitter() +
  geom_smooth()
```



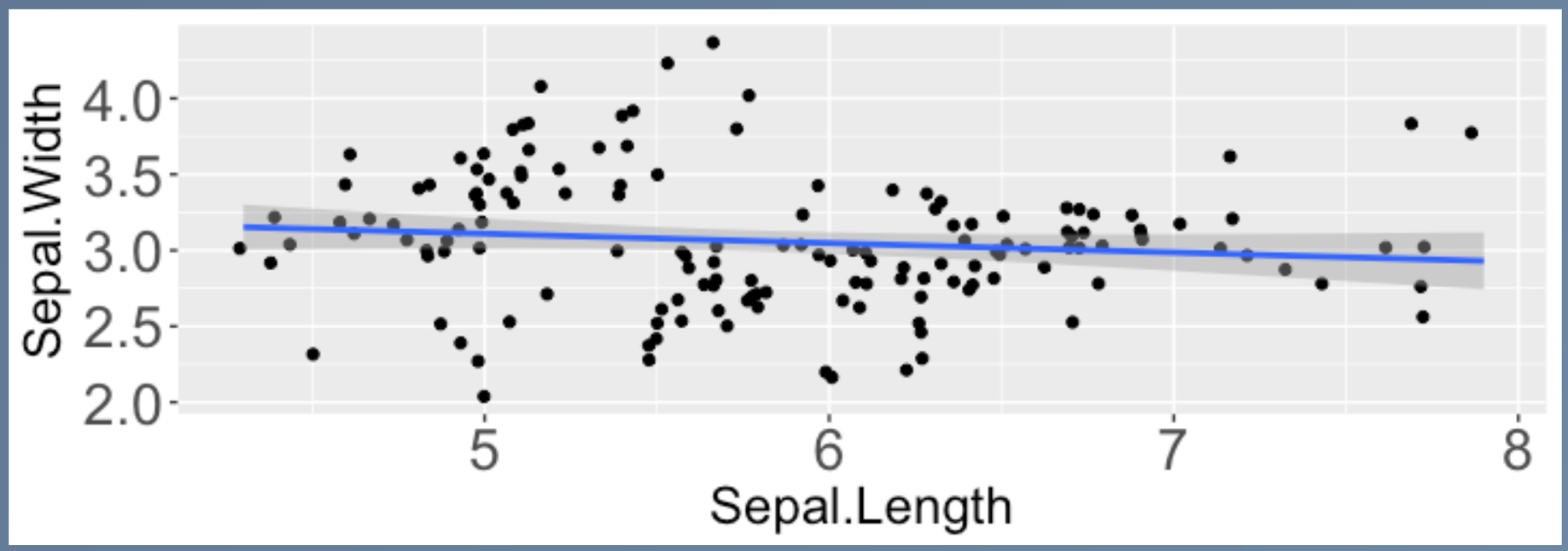
The Statistics Layer



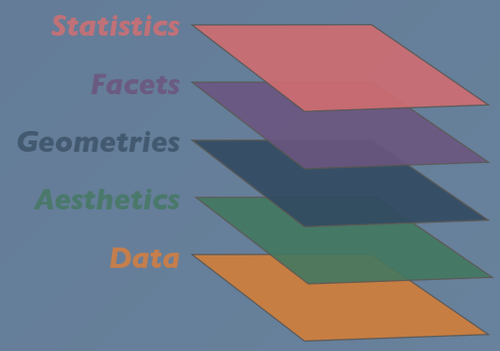
Fitting statistical models on the fly

The default method uses LOESS (locally estimated scatterplot smoothing) curve fitting.
Additional methods can be specified using the `method` argument:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
geom_jitter() +
geom_smooth(method = “lm”)
```



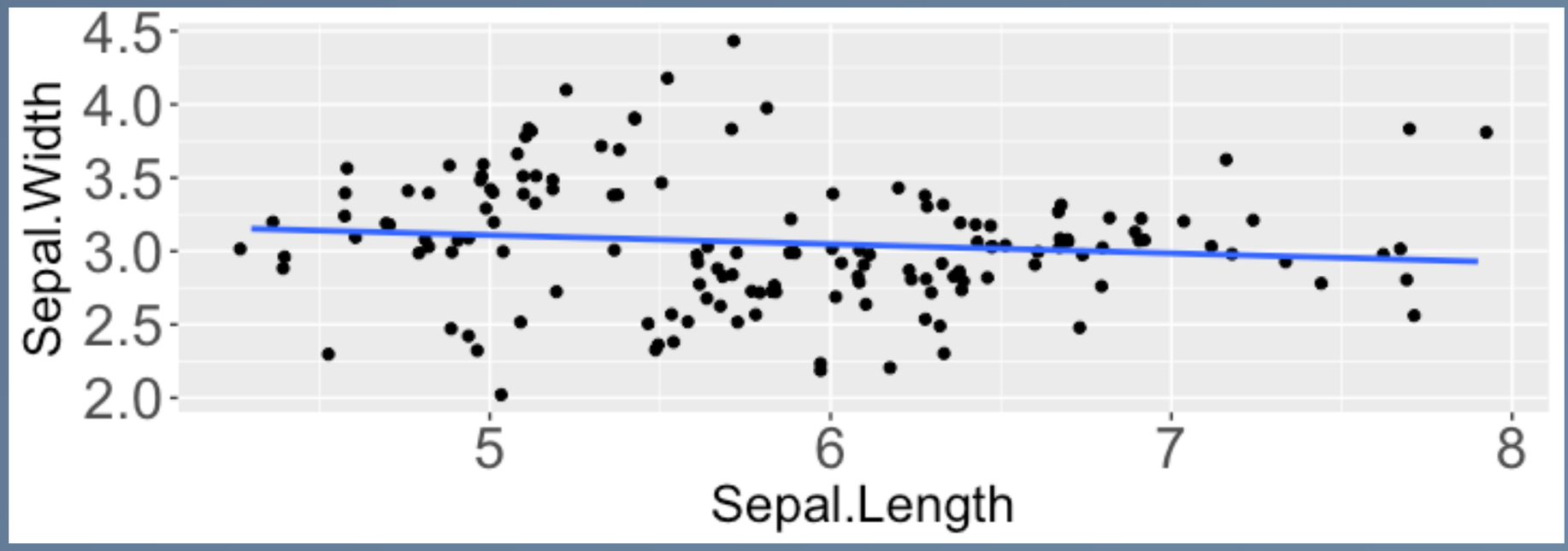
The Statistics Layer



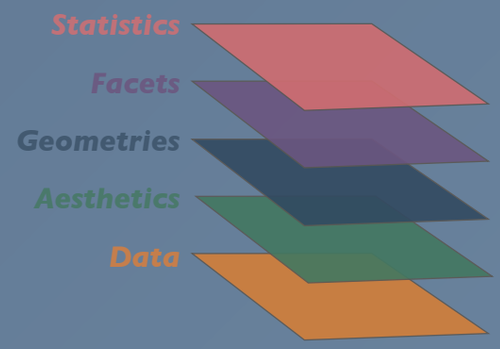
Fitting statistical models on the fly

The shaded bands represent 95% confidence intervals. These can be turned off by setting `se = F`

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_jitter() +
  geom_smooth(method = “lm”, se = F)
```



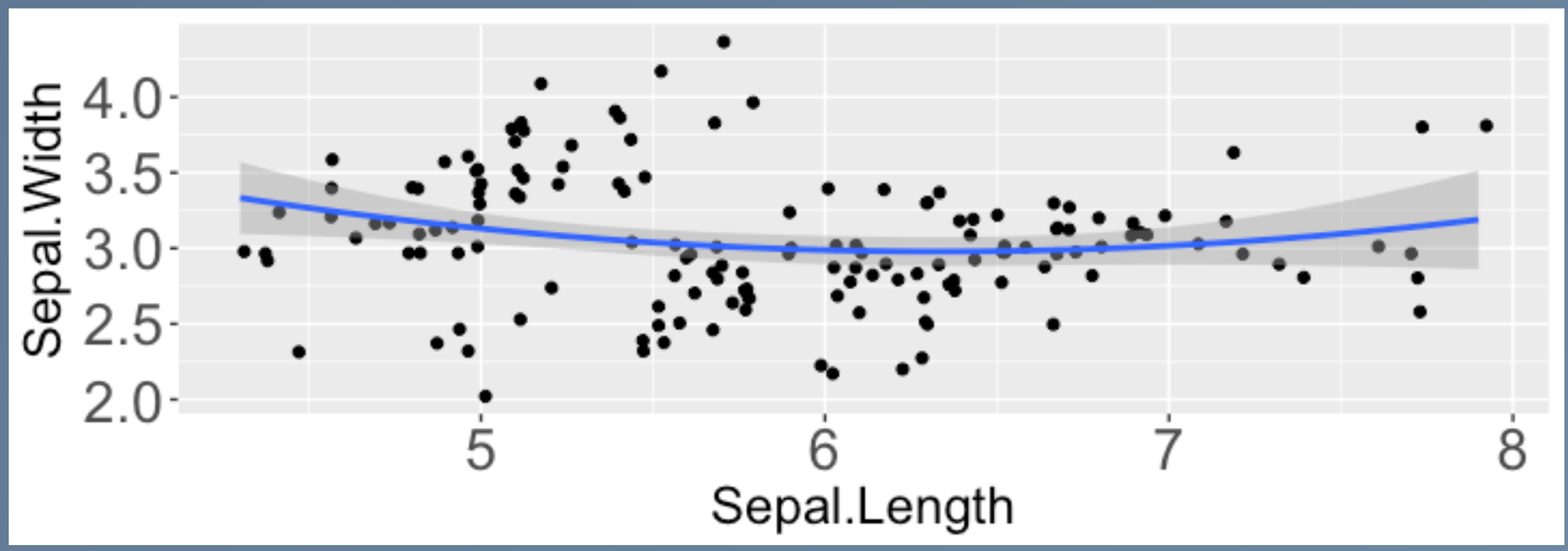
The Statistics Layer



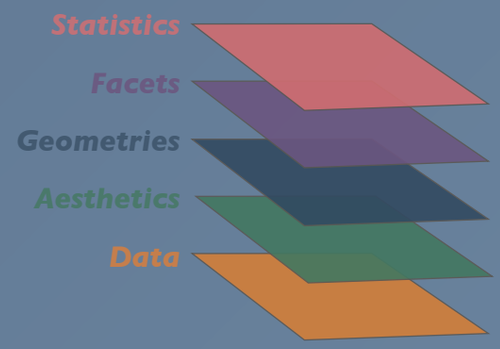
Fitting statistical models on the fly

More complicated models can be specified using the `formula` argument

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_jitter() +
  geom_smooth(method = “lm”, formula = y ~ x + I(x^2))
```



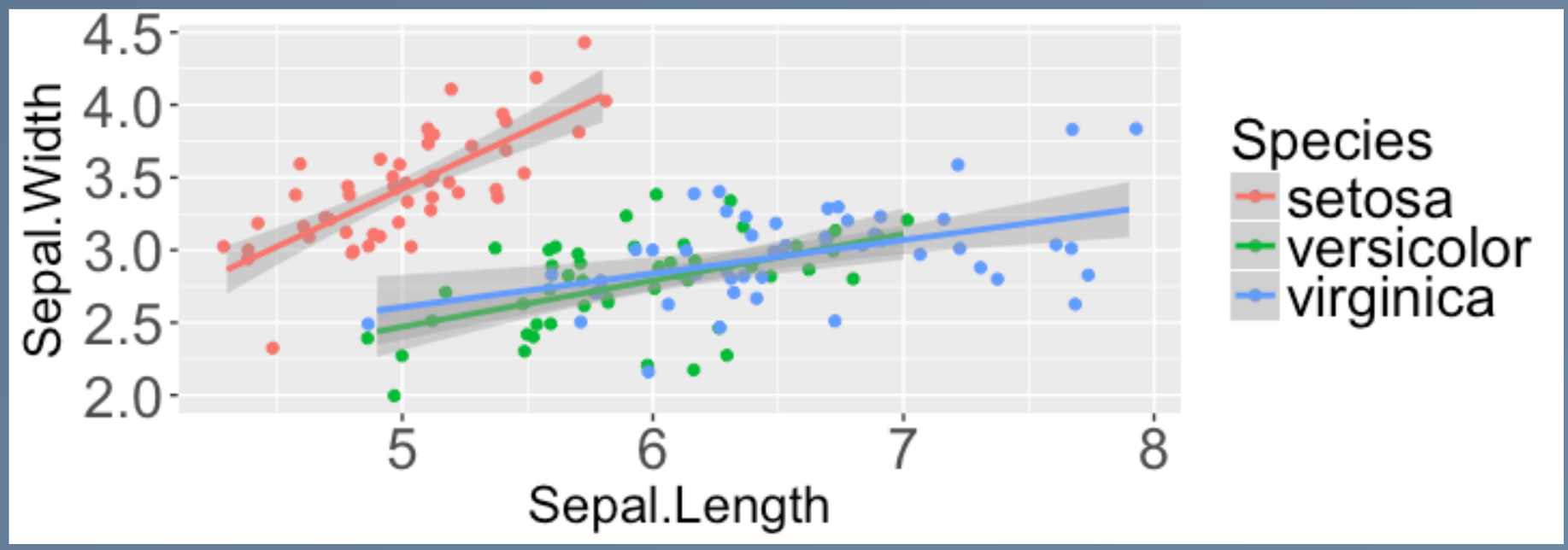
The Statistics Layer



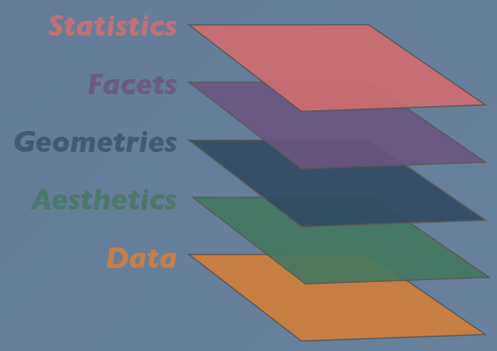
Fitting statistical models on the fly

Models are applied to data subsets when aesthetic mappings are defined using categorical variables:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_jitter() +
  geom_smooth(method = “lm”)
```



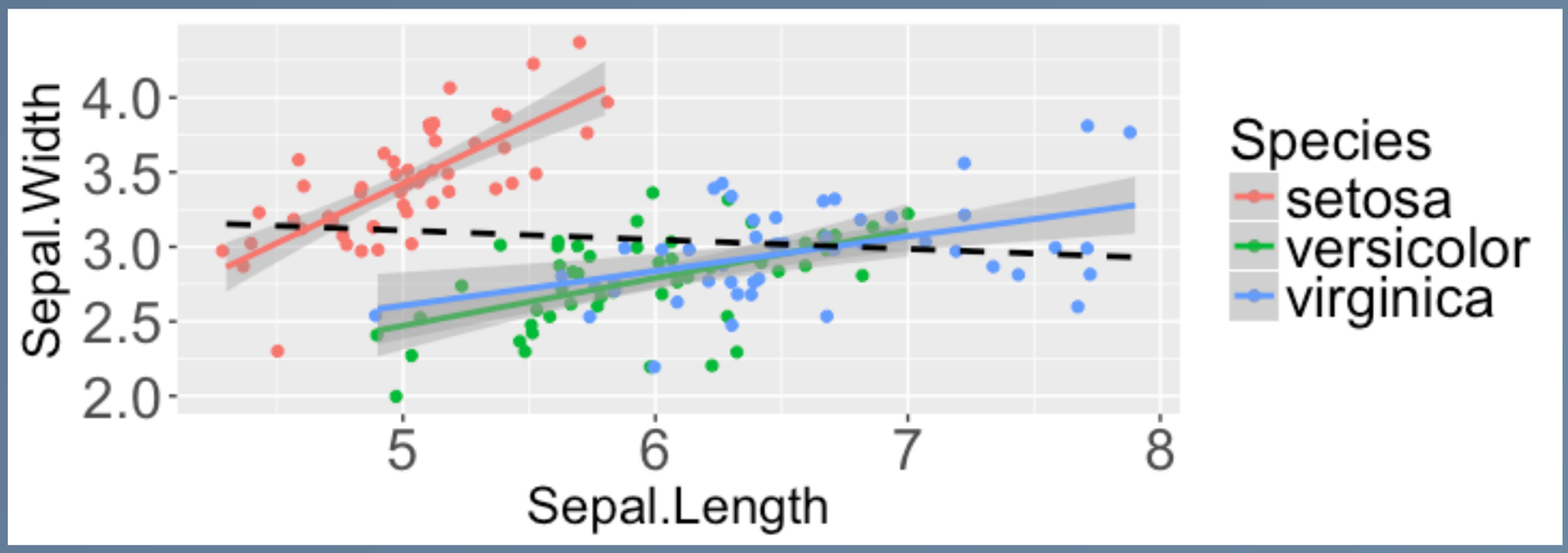
The Statistics Layer



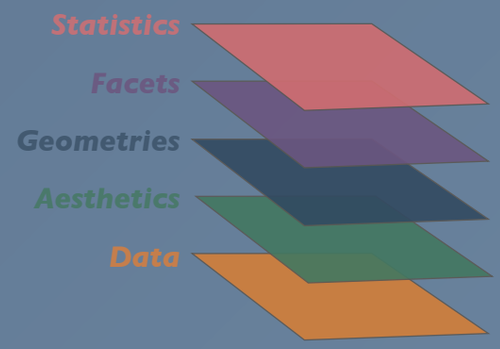
Fitting statistical models on the fly

Subsetting behaviour can be overwritten by using `aes(group = 1)` inside the statistics layer:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_jitter() +
  geom_smooth(method = “lm”) +
  geom_smooth(method = “lm”, aes(group = 1),
              se = F, linetype = “dashed”, col = “black”)
```



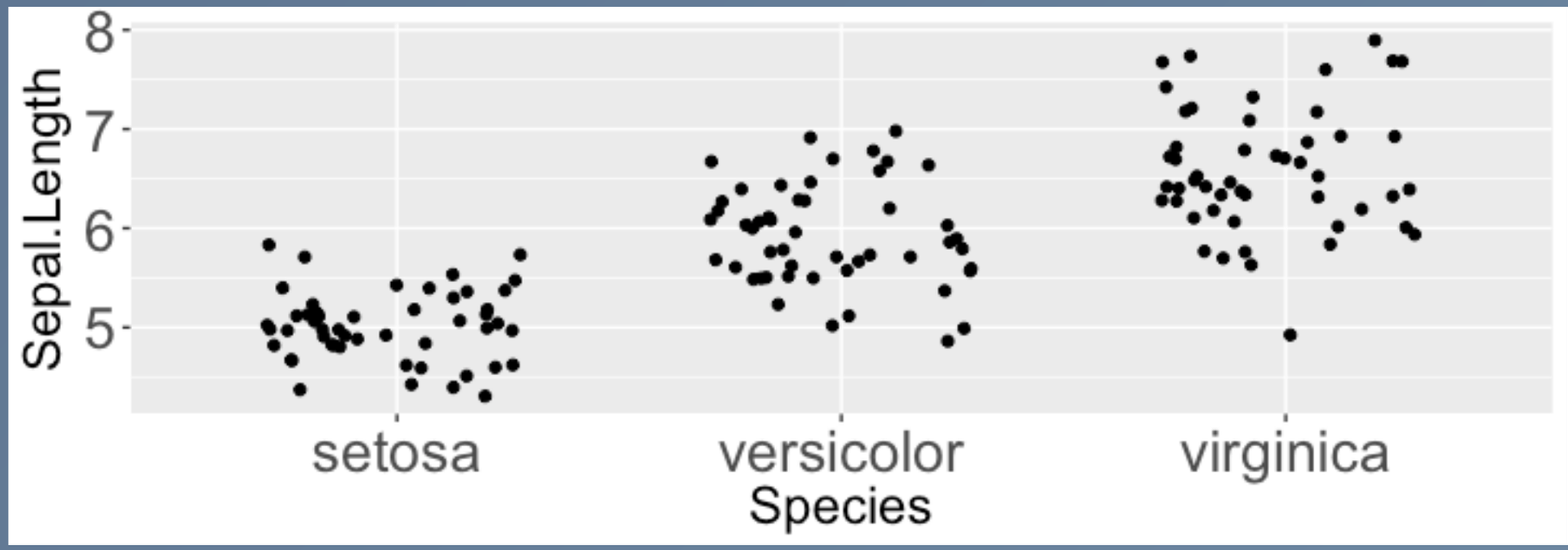
The Statistics Layer



Displaying summary statistics

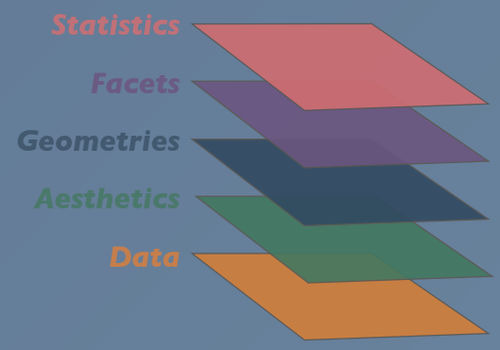
Consider the following plot:

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_jitter(width = 0.3)
```



We want to compute and display group means and standard deviations

The Statistics Layer

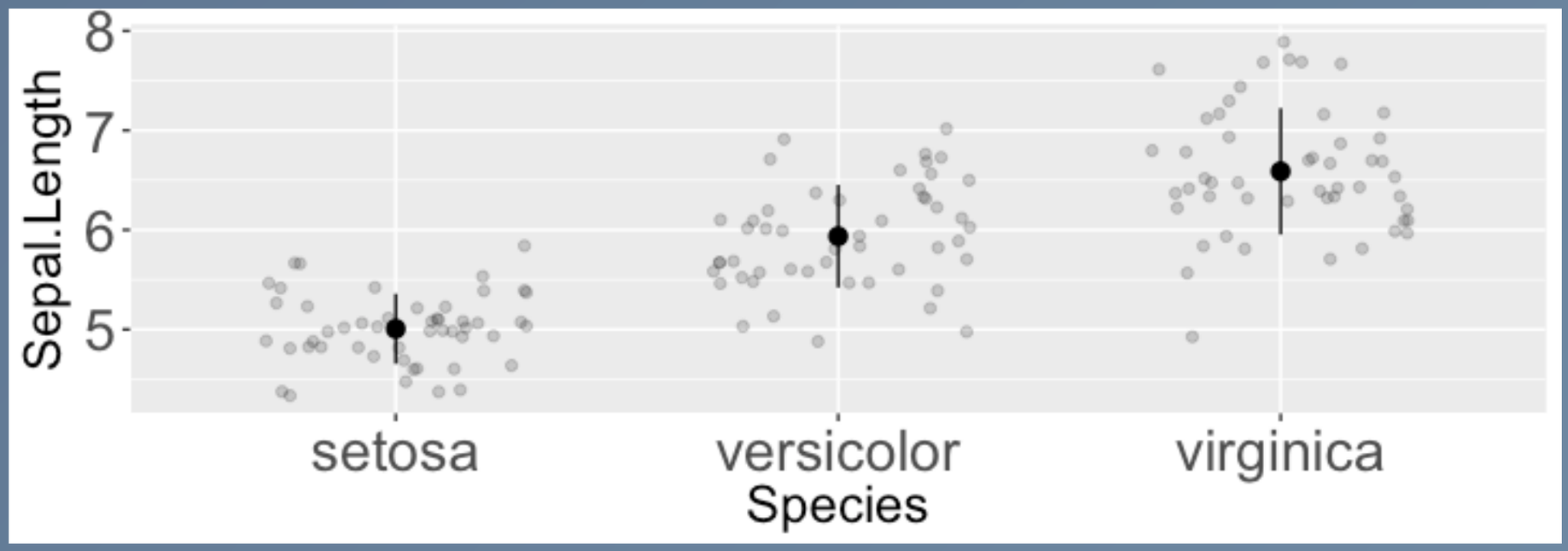


Displaying summary statistics

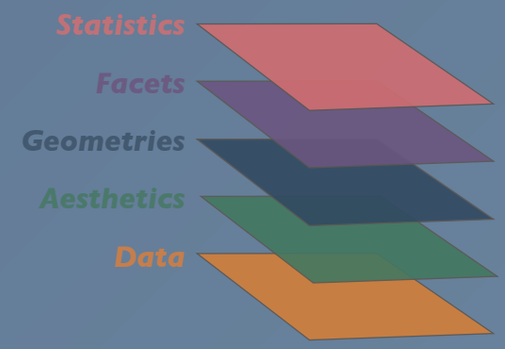
One way would be to create a new data frame with these values and add secondary data and aesthetic layers to the plot.

Instead we can add a statistics layer with `stat_summary()`:

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
geom_jitter(width = 0.3, alpha = 0.2) +
stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1))
```



The Statistics Layer



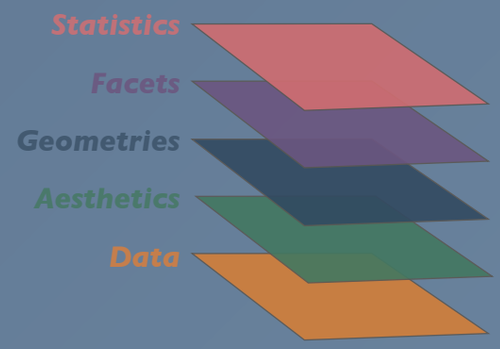
Displaying summary statistics

The `fun.data` argument specifies the function used to generate the summary statistics.

`stat_summary()` uses `geom_pointrange` by default. The output of the function used must match the aesthetics required by the geometry.

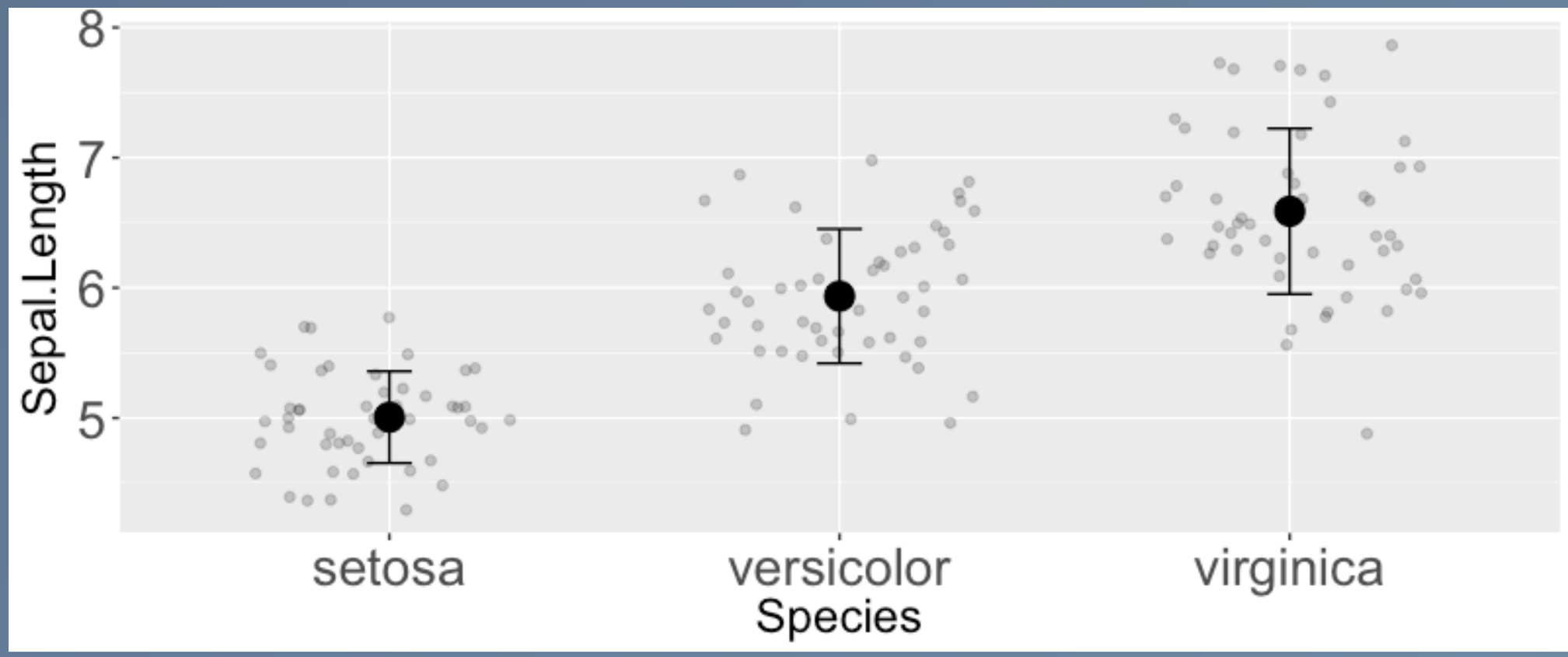
Different geometries can be specified using the `geom` argument.

The Statistics Layer

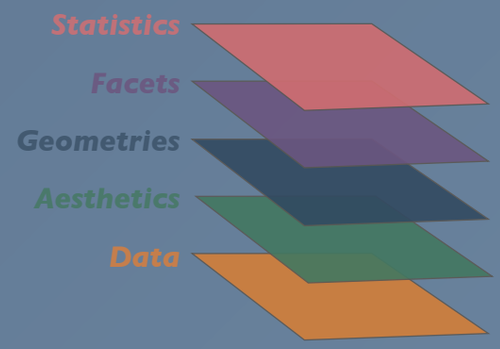


Displaying summary statistics

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  geom_jitter(width = 0.3, alpha = 0.2) +
  stat_summary(fun.y = mean, geom = “point”, size = 5) +
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1),
               geom = "errorbar", width = 0.1)
```

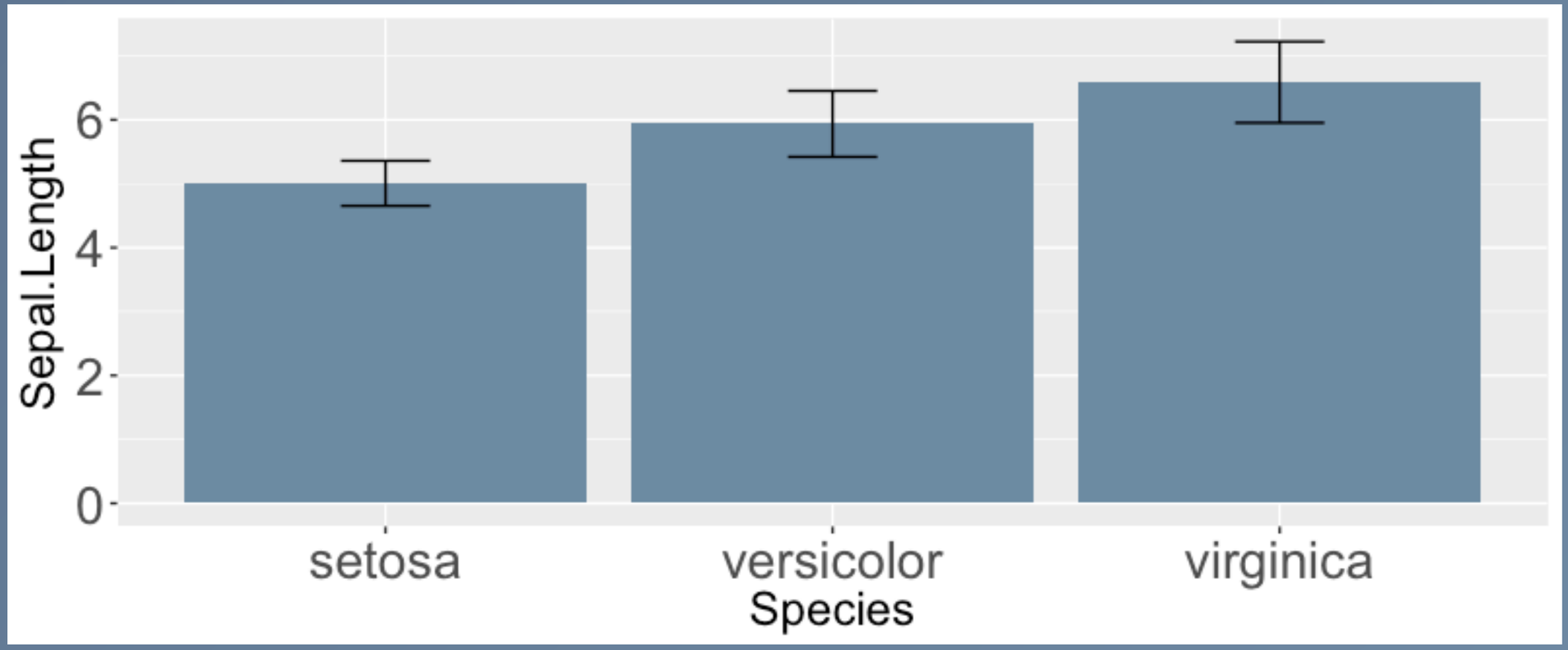


The Statistics Layer

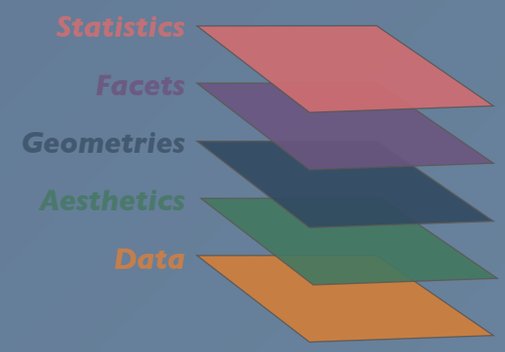


Displaying summary statistics

```
ggplot(iris, aes(x = Species, y = Sepal.Length)) +
  stat_summary(fun.y = mean, geom = “bar”, fill = "#708CA5") +
  stat_summary(fun.data = mean_sdl, fun.args = list(mult = 1),
                    geom = "errorbar", width = 0.2)
```



The Statistics Layer



Summary

- The statistics layer is used to display statistics on the graphic
- Statistical models can be generated using `geom_smooth()` or `stat_smooth()`
- Summary statistics are created using `stat_summary()`
- Functions used to create statistical summaries may require different geometries
- Adding a statistics layer is a quick and easy way to visualize trends in your data

The Coordinates Layer

Introduction

- The coordinates layer controls the dimensions of the plot
- The dimensions of the plot are modified using `coord_` or `scale_` functions
- Scales control the range of aesthetic mappings
- Coordinates control the extent of plot displayed

The Coordinates Layer

Scales vs. coordinates

Each aesthetic has an associated scale.

Scales are accessed using the `scale_` functions:

- `scale_x_continuous()`, `scale_x_discrete()`, …
- `scale_y_continuous()`, `scale_y_discrete()`, …
- `scale_color_continuous()`, `scale_color_discrete()`, …
- `scale_fill_…`
- `scale_alpha_…`

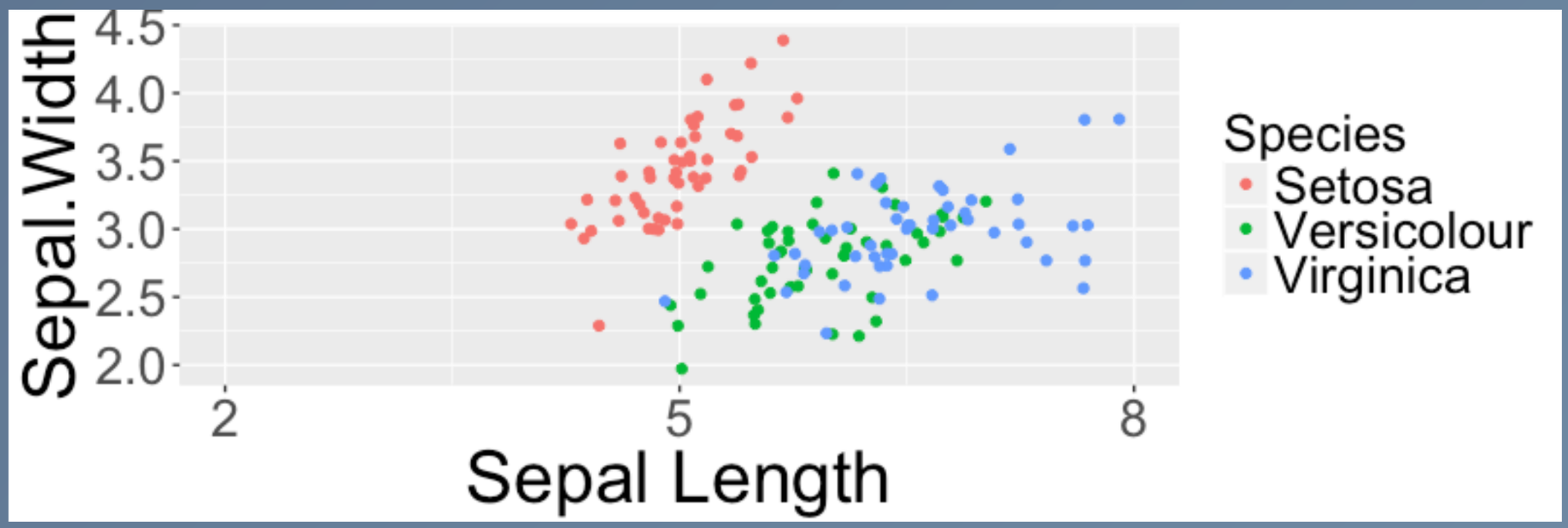
Scales modify the range of the aesthetic mappings.

We can also use scales to modify the associated labels.

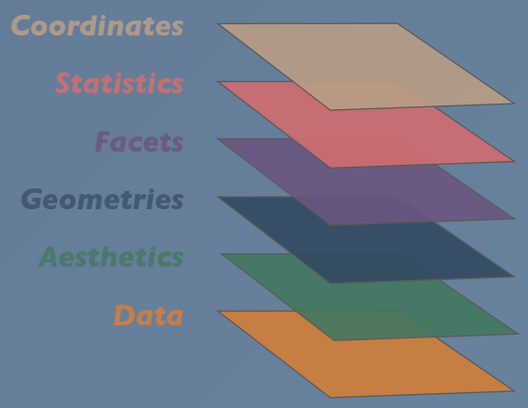
The Coordinates Layer

Scales vs. coordinates

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_point(position = "jitter") +
  scale_x_continuous("Sepal Length",
                     limits = c(2,8),
                     breaks = seq(2,8,3)) +
  scale_color_discrete("Species",
                       labels = c("Setosa","Versicolour","Virginica"))
```



The Coordinates Layer

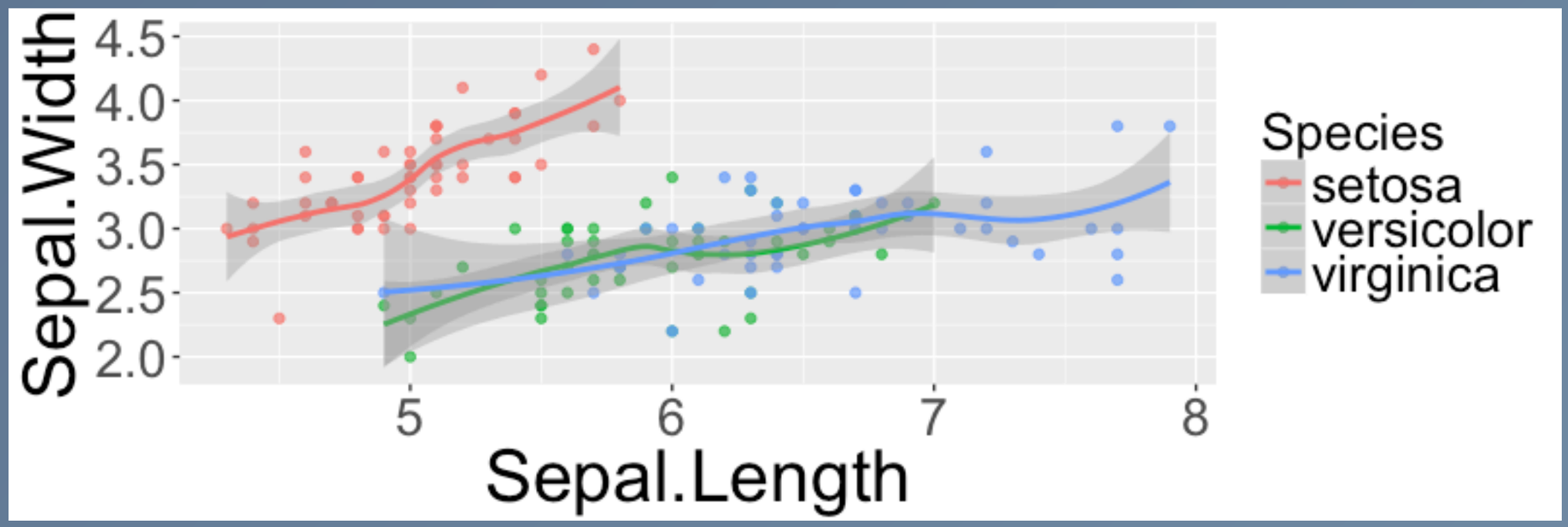


Scales vs. coordinates

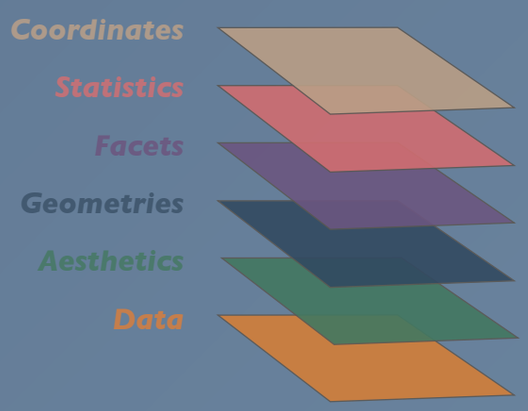
Scales can be used to zoom in on a plot.

Consider the following plot:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_point(alpha = 0.7) +
  geom_smooth()
```



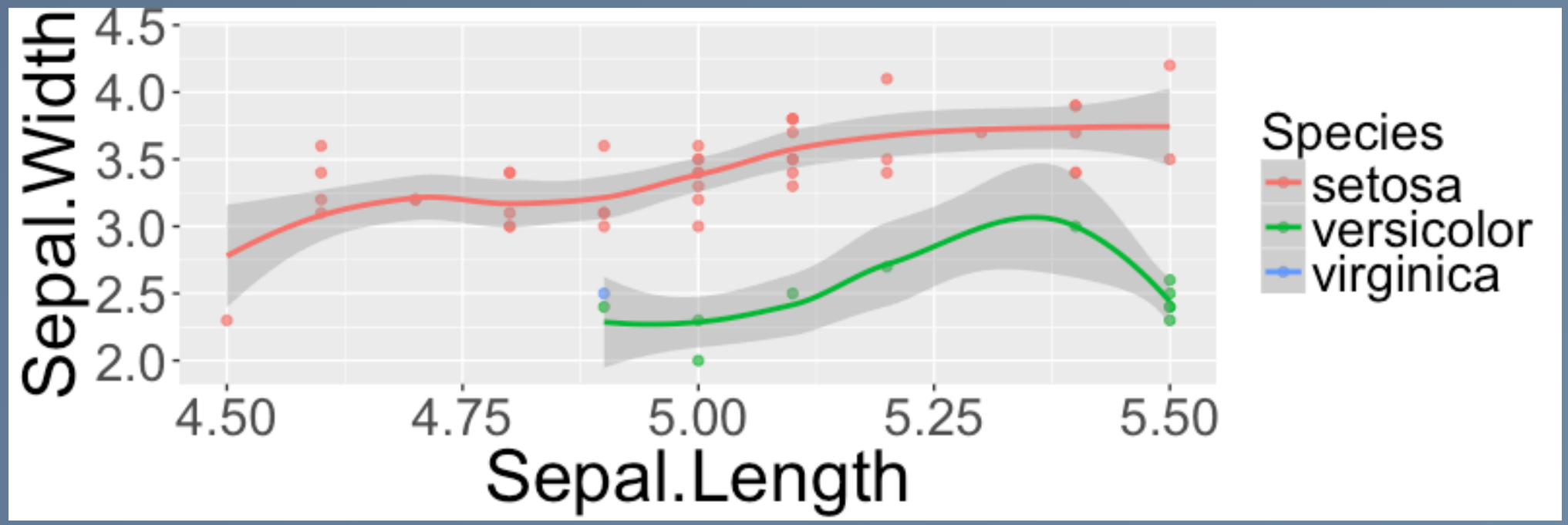
The Coordinates Layer



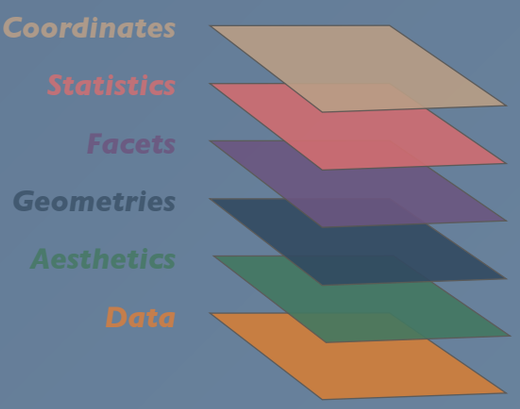
Scales vs. coordinates

Change the x-axis scale to zoom:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_point(alpha = 0.7) +
  geom_smooth() +
  scale_x_continuous(limits = c(4.5, 5.5))
```



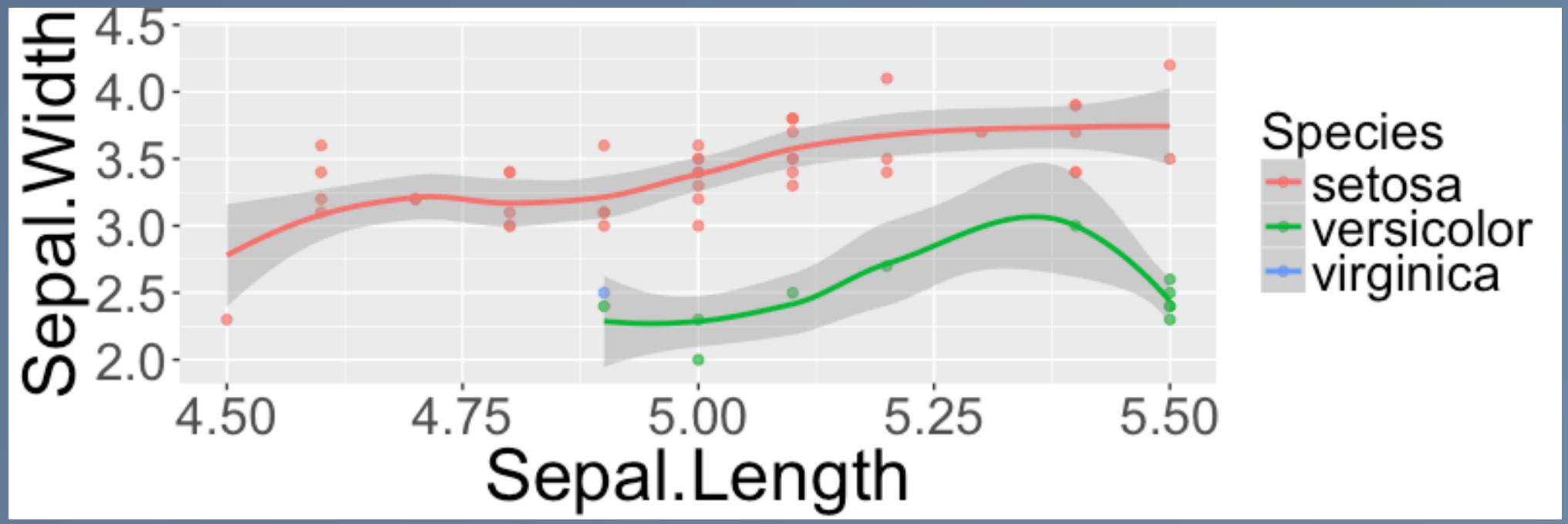
The Coordinates Layer



Scales vs. coordinates

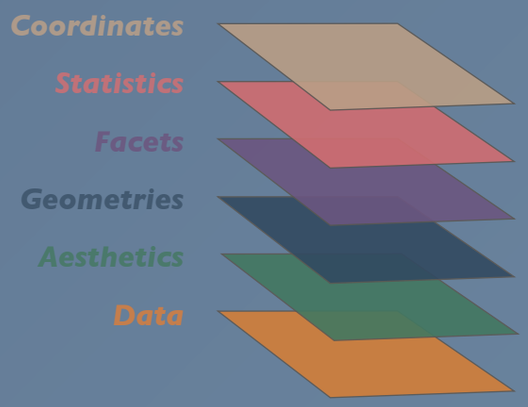
Change the x-axis scale to zoom:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_point(alpha = 0.7) +
  geom_smooth() +
  scale_x_continuous(limits = c(4.5, 5.5))
```



Modifying a scale filters the aesthetic mapping!

The Coordinates Layer



Scales vs. coordinates

Scales act at the level of the aesthetic mappings.

This is will have an impact on how statistics are computed in a statistics layer.

Coordinates allow us to modify the dimensions without modifying the mappings.

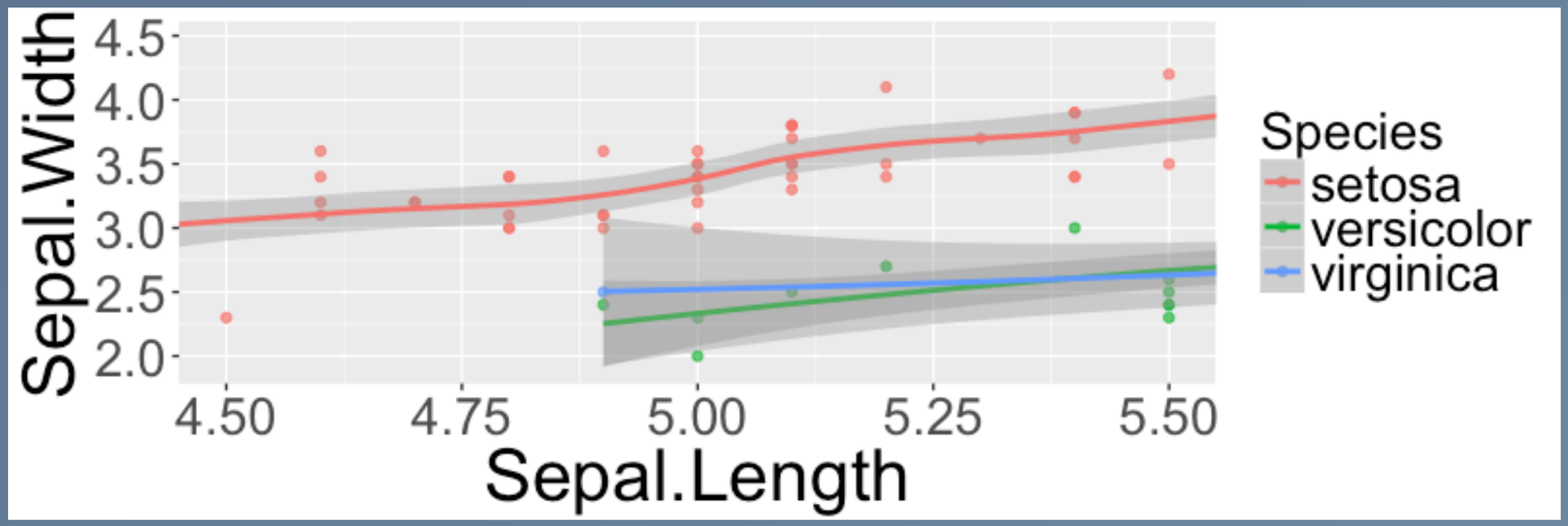
The most common coordinates function is `coord_cartesian()`.

The Coordinates Layer

Scales vs. coordinates

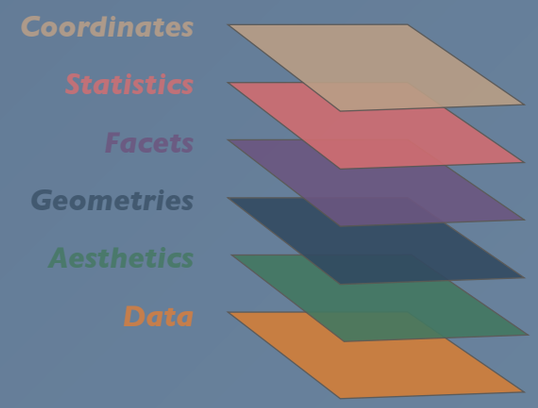
Zooming in with `coord_cartesian()`:

```
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width, col = Species)) +
  geom_point(alpha = 0.7) +
  geom_smooth() +
  coord_cartesian(xlim = c(4.5, 5.5))
```



Notice the original statistical models are preserved.

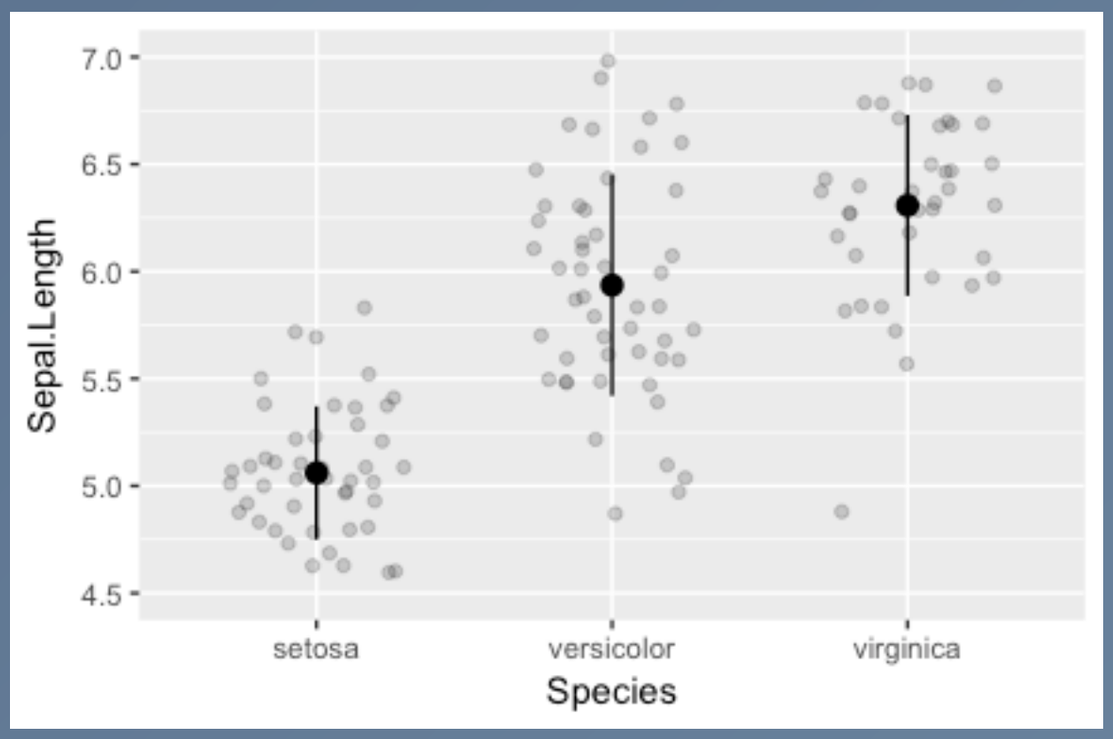
The Coordinates Layer



Another example:

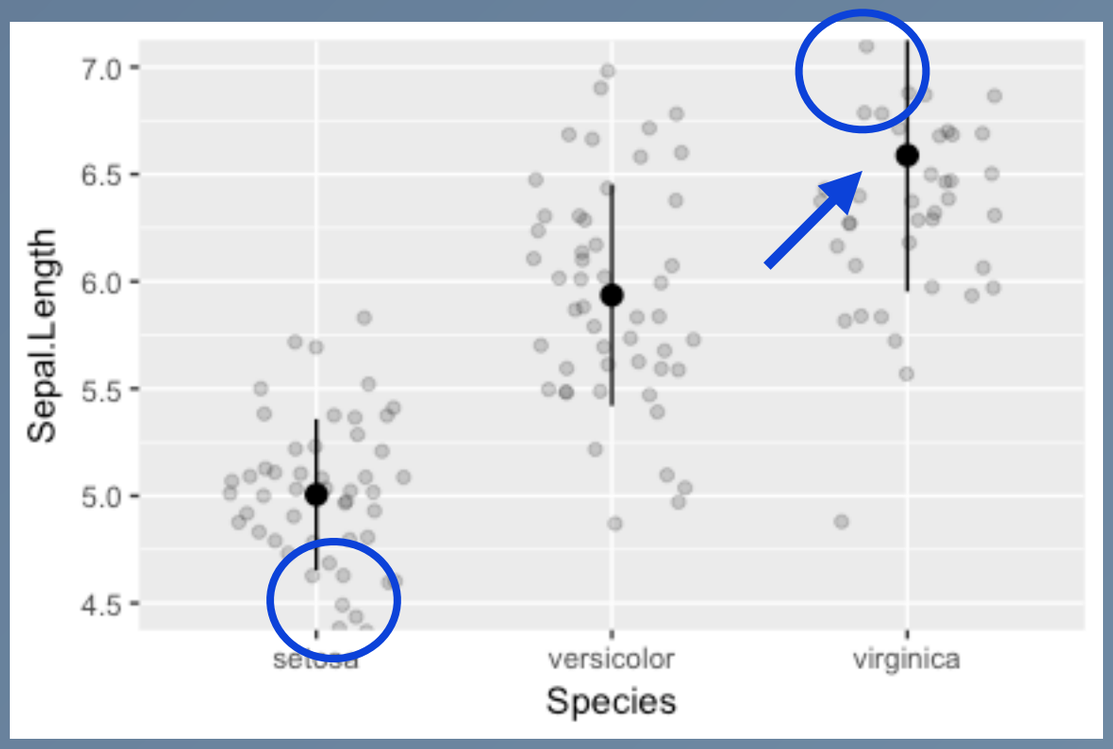
Scales

```
ggplot(iris, aes(x = Species,
                 y = Sepal.Length)) +
  geom_jitter(width = 0.3,
              alpha = 0.2) +
  stat_summary(fun.data = mean_sdl,
               fun.args = list(mult = 1)) +
  scale_y_continuous(limits = c(4.5, 7.0))
```

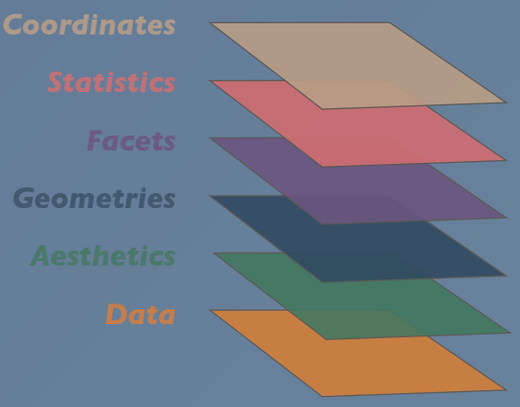


Coordinates

```
ggplot(iris, aes(x = Species,
                 y = Sepal.Length)) +
  geom_jitter(width = 0.3,
              alpha = 0.2) +
  stat_summary(fun.data = mean_sdl,
               fun.args = list(mult = 1)) +
  coord_cartesian(ylim = c(4.5, 7.0))
```



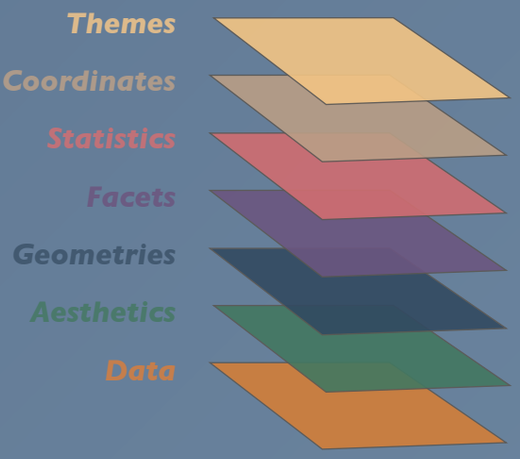
The Coordinates Layer



Summary

- The coordinates layer controls the dimensions of the plot
- `scale_` functions filter the aesthetic mappings
- `coord_` functions change the plot dimensions in the proper sense

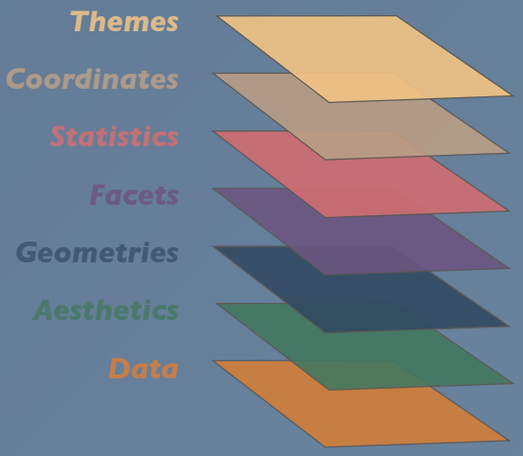
The Themes Layer



Introduction

- The themes layer controls all non-data ink on your visualization
- The themes layer is where you beautify your plots
- Themes are controlled using the `theme()` function
- There are three main thematic elements: text, lines and rectangles
- Thematic elements are controlled using `element_` functions within the `theme()` function:
    - `element_text()`
    - `element_line()`
    - `element_rect()`

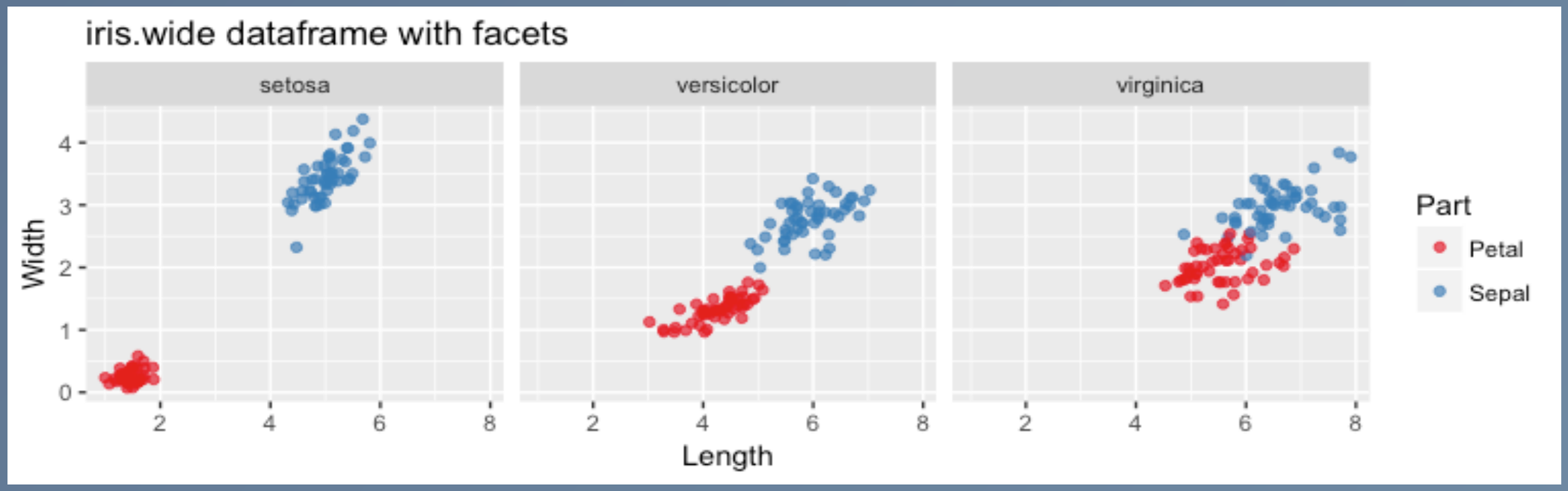
The Themes Layer



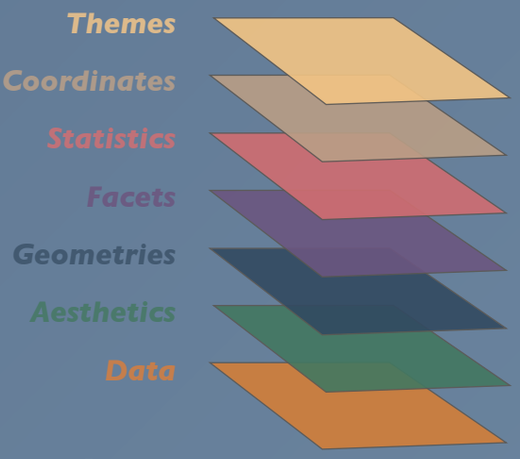
Thematic elements

Consider the following plot:

```
ggplot(iris.wide, aes(x = Length, y = Width, col = Part)) +
  geom_jitter(alpha = 0.7) +
  scale_color_brewer(palette = "Set1") +
  facet_grid(.~Species) +
  ggtitle("iris.wide dataframe with facets")
```



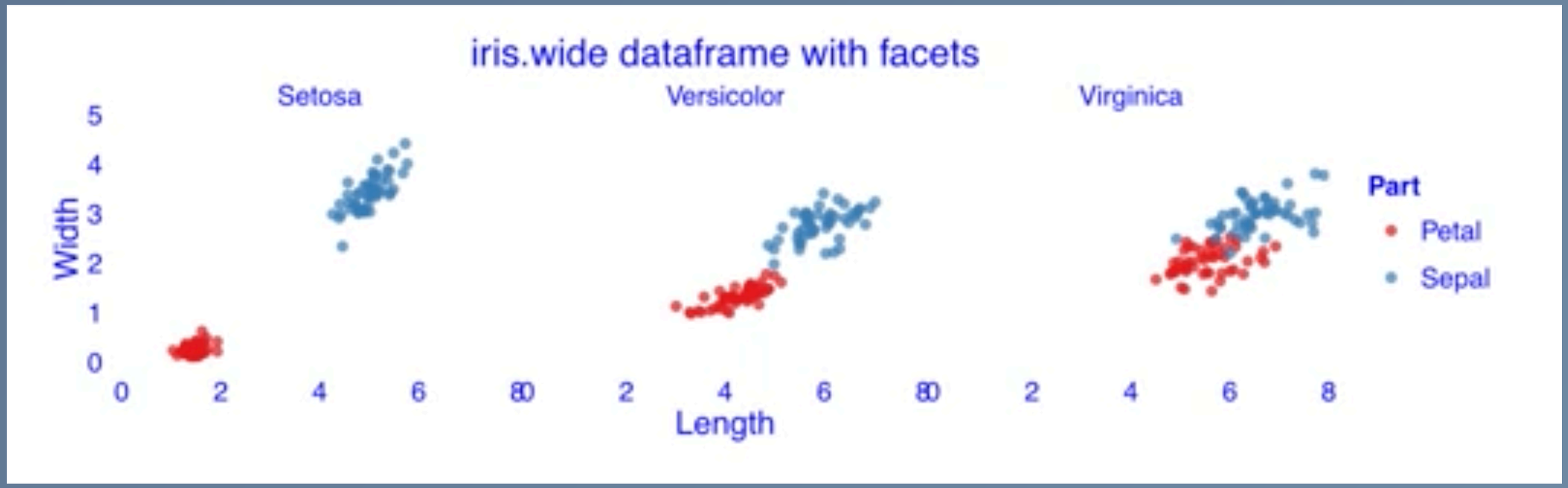
The Themes Layer



Thematic elements: Text

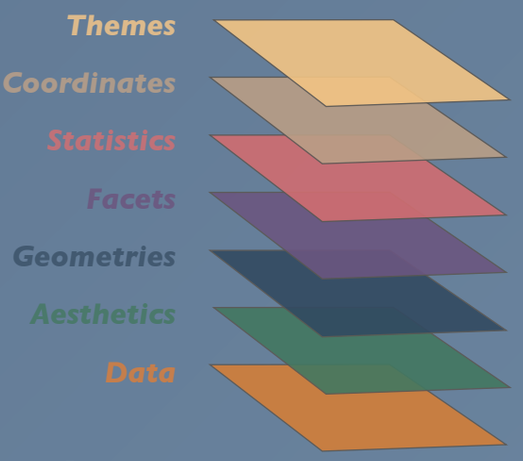
`element_text()` controls all of the text elements in the plot.

This includes the title, axis titles, axis labels, facet labels, and the legend.



https://www.datacamp.com/courses/data-visualization-with-ggplot2-2

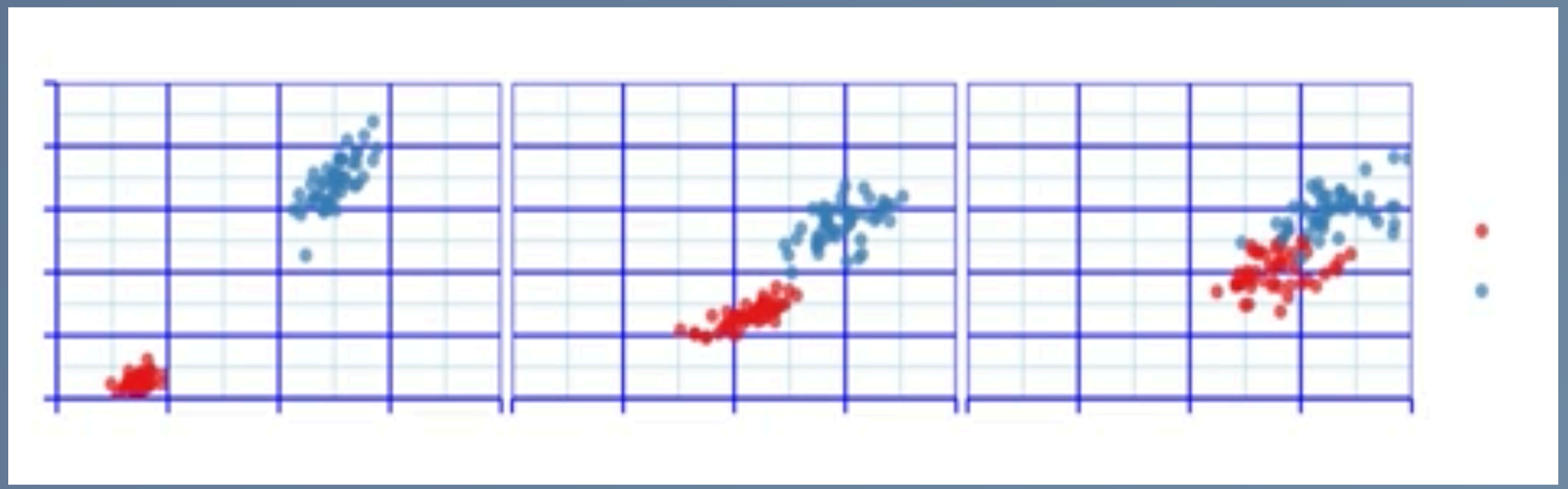
The Themes Layer



Thematic elements: Lines

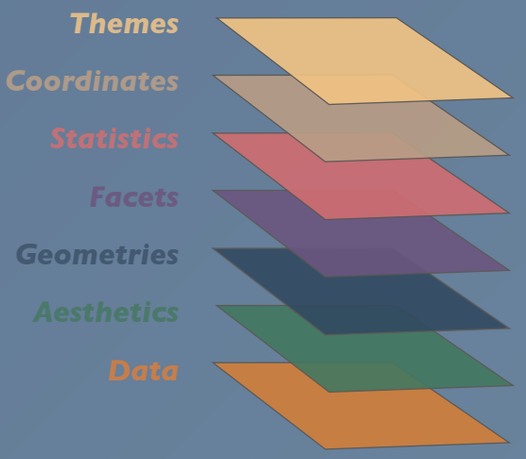
`element_line()` controls all of the line elements in the plot.

This includes the major and minor panel grid lines, the axis lines, the axis ticks, and the background border.



https://www.datacamp.com/courses/data-visualization-with-ggplot2-2

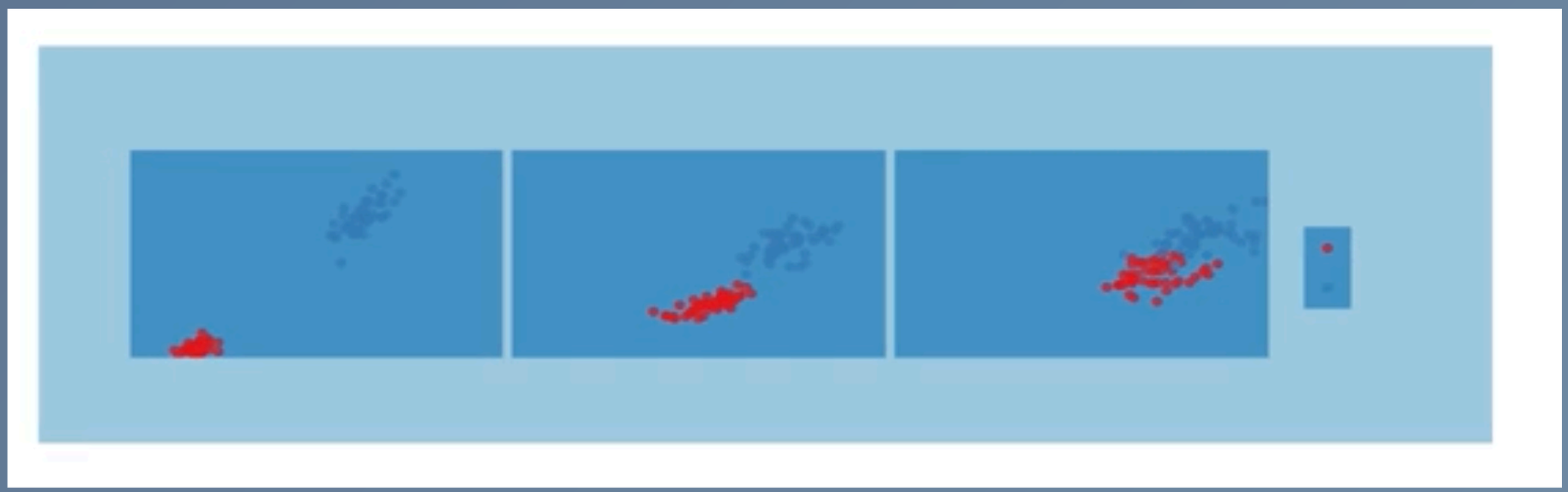
The Themes Layer



Thematic elements: Rectangles

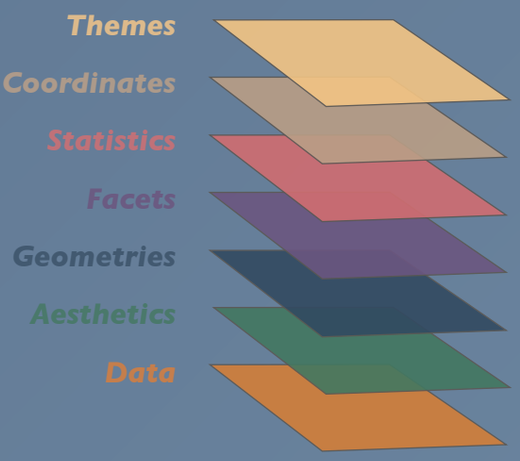
`element_rect()` controls all of the rectangle elements in the plot.

This includes the plot background, the panel backgrounds, the legend box.



https://www.datacamp.com/courses/data-visualization-with-ggplot2-2

The Themes Layer



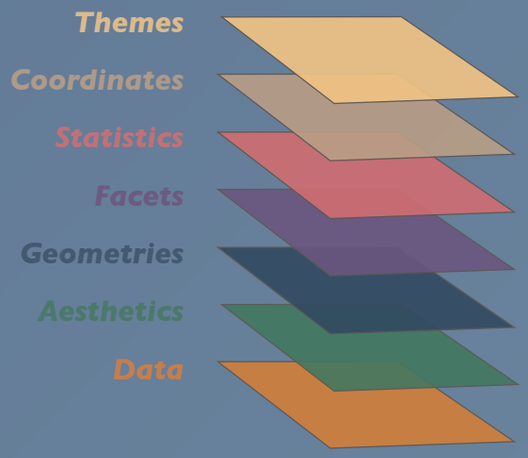
Arguments of `theme()`

Visual elements are modified via the arguments of the `theme()` function using the appropriate `element_` function.

Text elements include:

- `text`
- `title`
- `plot.title`
- `legend.text`
- `legend.title`
- `axis.title`
- `axis.title.x`
- `axis.title.y`
- `axis.text`
- `axis.text.x`
- `axis.text.y`
- `strip.text`
- `strip.text.x`
- `strip.text.y`

The Themes Layer



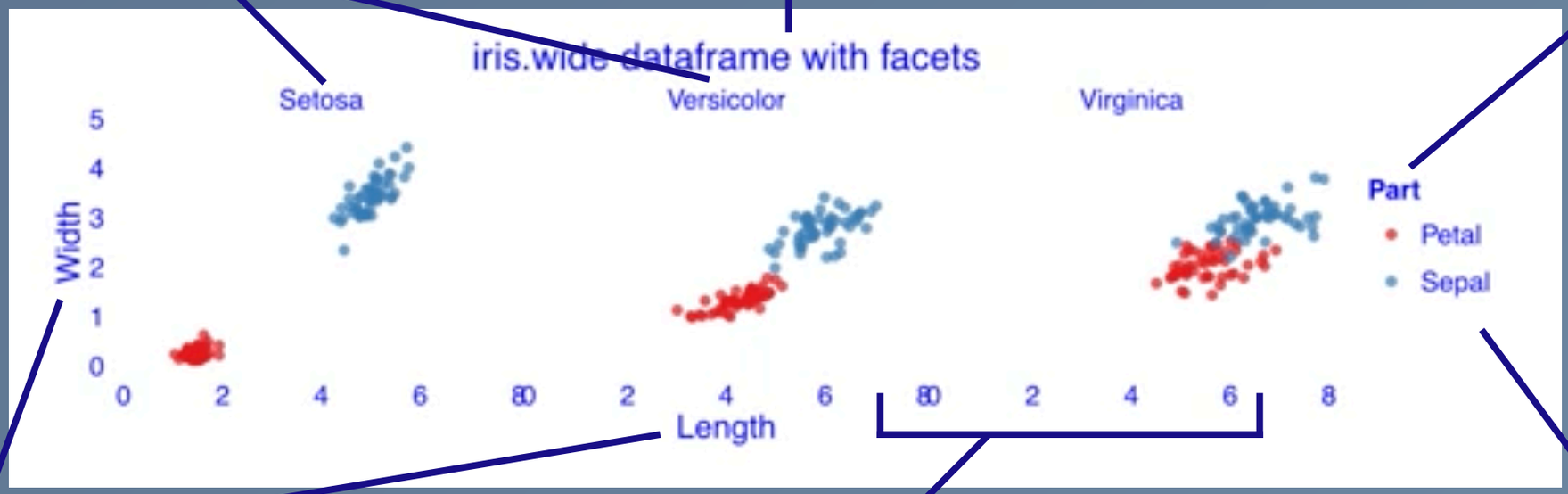
Arguments of `theme()`

Text elements include:

`plot.title`

`strip.text`

`legend.title`

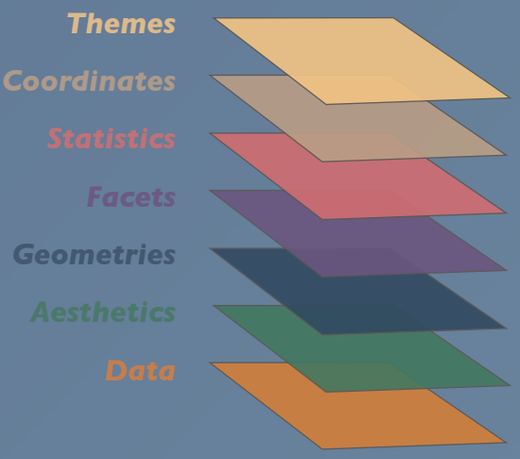


`axis.title`

`axis.text`

`legend.text`

The Themes Layer



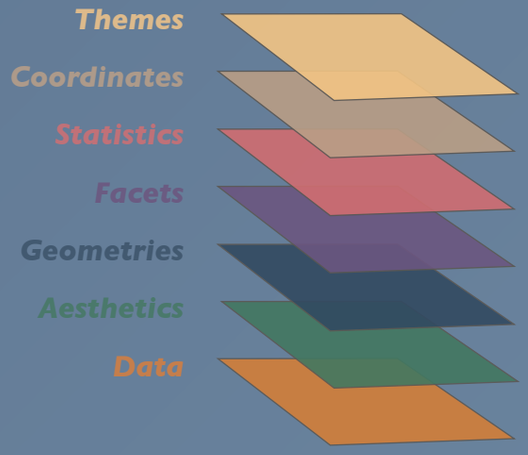
Arguments of `theme()`

Visual elements are modified via the arguments of the `theme()` function using the appropriate `element_` function.

Line elements include:

- `line`
- `axis.ticks`
- `axis.ticks.x`
- `axis.ticks.y`
- `axis.line`
- `axis.line.x`
- `axis.line.y`
- `panel.grid`
- `panel.grid.major`
- `panel.grid.major.x`
- `panel.grid.major.y`
- `panel.grid.minor`
- `panel.grid.minor.x`
- `panel.grid.minor.y`

The Themes Layer

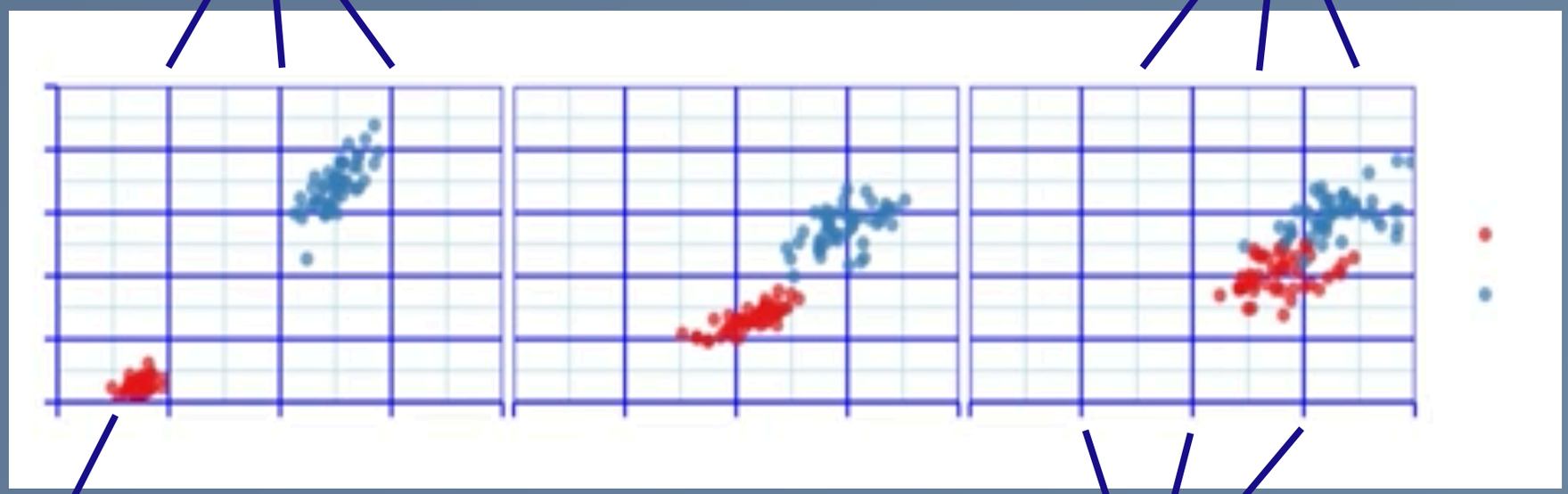


Thematic elements: Lines

Line elements include:

`panel.grid.major`

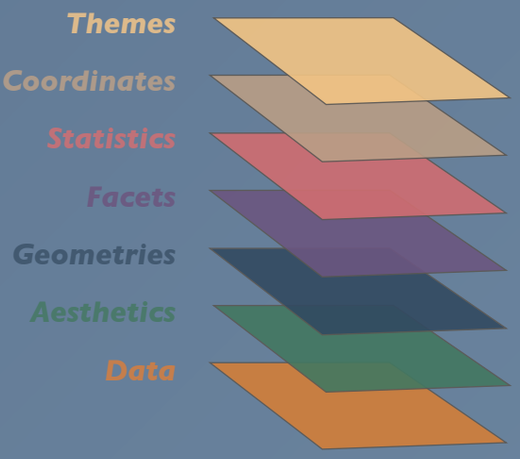
`panel.grid.minor`



`axis.line`

`axis.ticks`

The Themes Layer



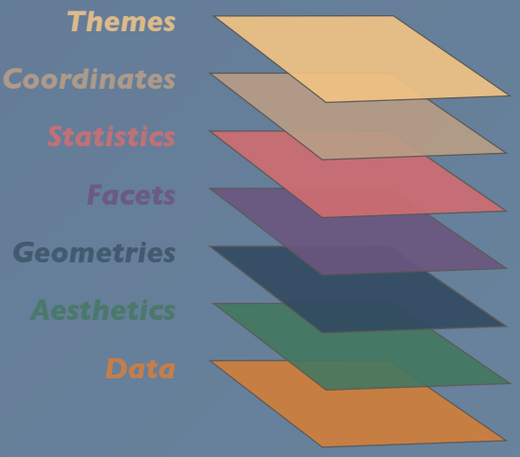
Arguments of `theme()`

Visual elements are modified via the arguments of the `theme()` function using the appropriate `element_` function.

Rectangle elements include:

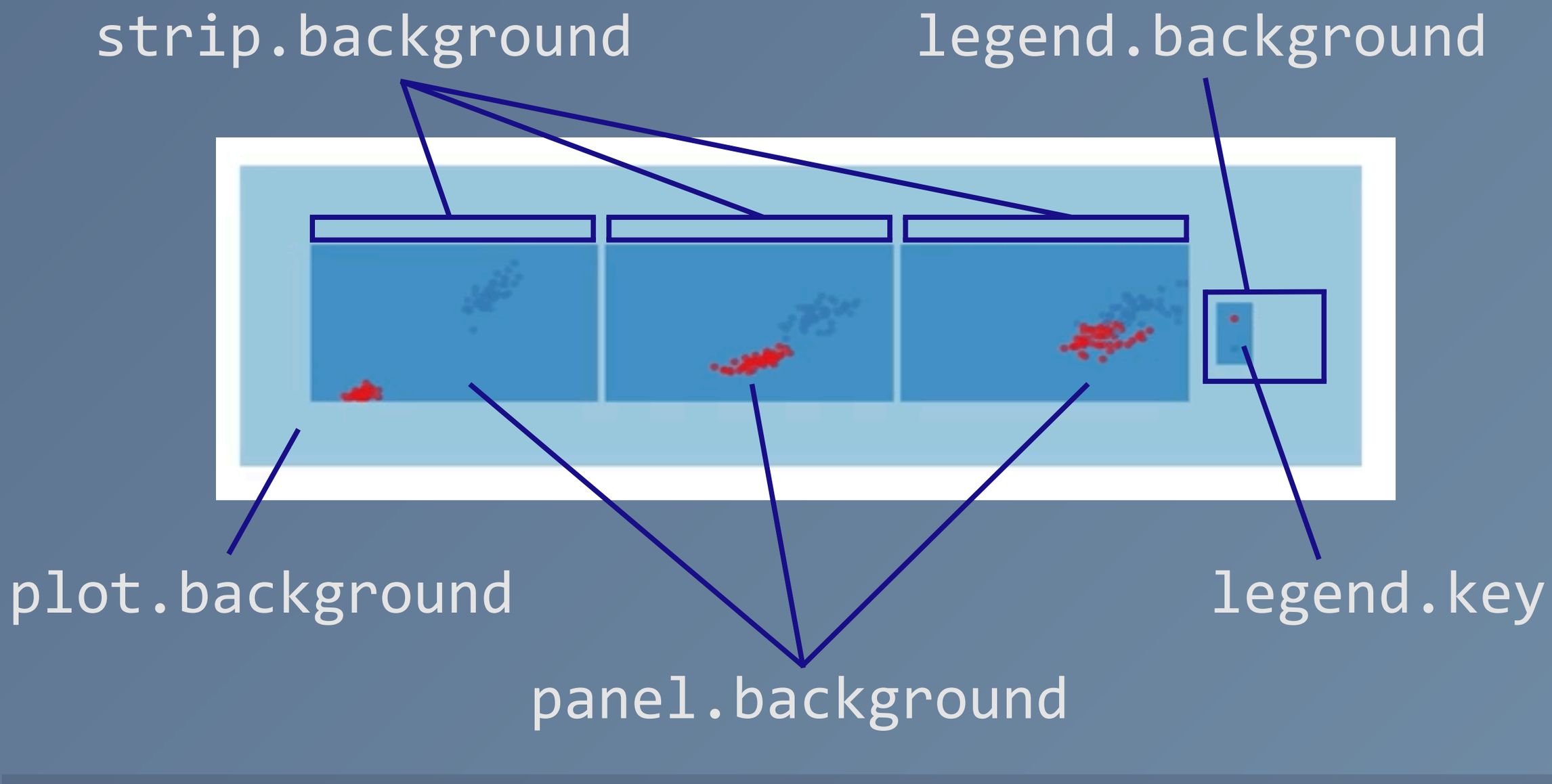
- `rect`
- `legend.background`
- `legend.key`
- `strip.background`
- `panel.background`
- `panel.border`
- `plot.background`

The Themes Layer

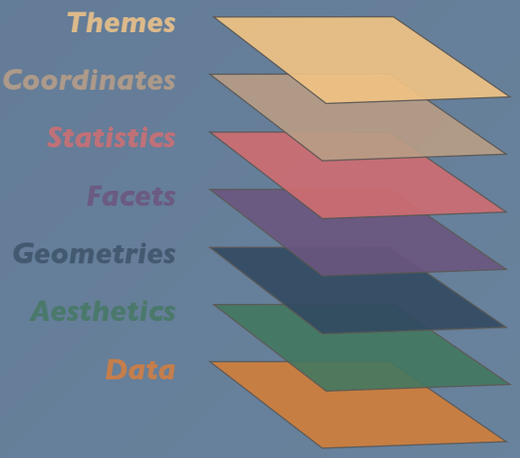


Arguments of `theme()`

Rectangle elements include:



The Themes Layer



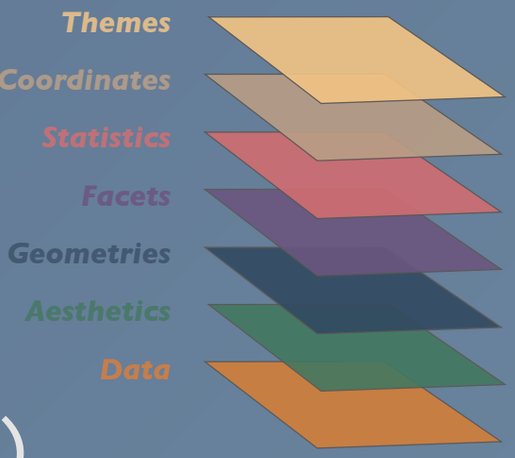
Arguments of `theme()`

The arguments of themes follow a hierarchical inheritance:

- text
  - title
    - plot.title
    - legend.title
  - axis.title
    - axis.title.x
    - axis.title.y
  - legend.text
  - axis.text
    - axis.text.x
    - axis.text.y
  - strip.text
    - strip.text.x
    - strip.text.y
- line
  - axis.ticks
    - axis.ticks.x
    - axis.ticks.y
  - axis.line
    - axis.line.x
    - axis.line.y
  - panel.grid
    - panel.grid.major
      - panel.grid.major.x
      - panel.grid.major.y
    - panel.grid.minor
      - panel.grid.minor.x
      - panel.grid.minor.y
- rect
  - legend.background
  - legend.key
  - panel.background
  - panel.border
  - plot.background
  - strip.background

https://www.datacamp.com/courses/data-visualization-with-ggplot2-2

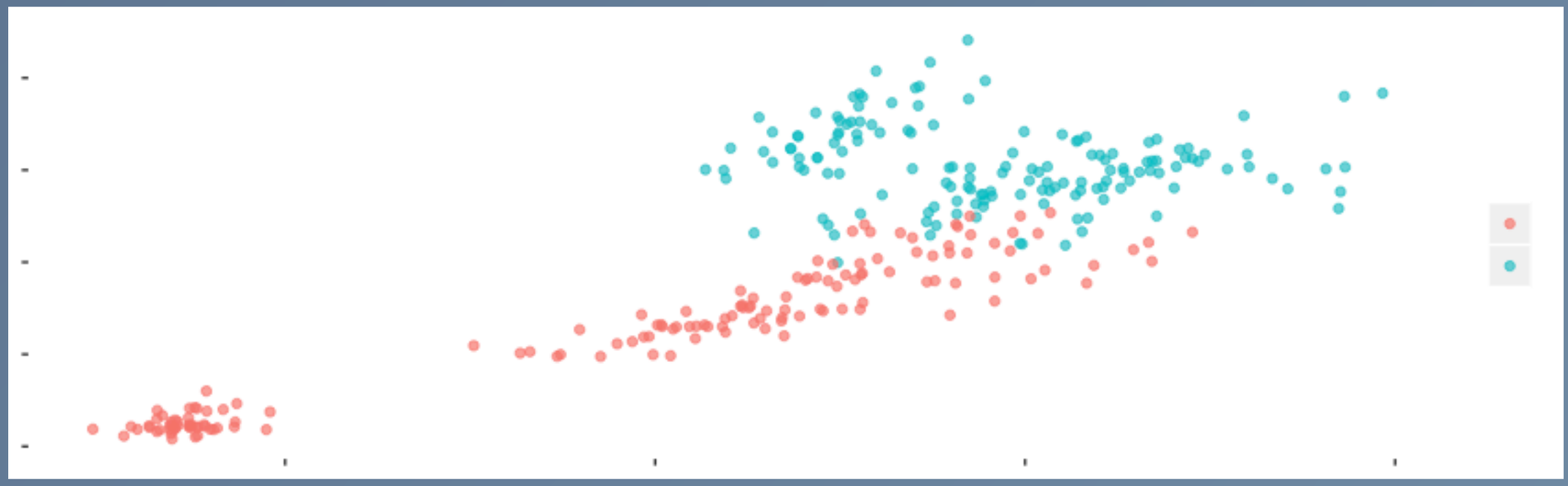
The Themes Layer



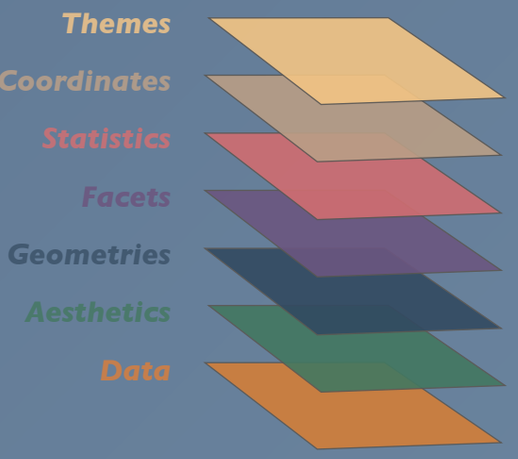
Removing thematic elements

Visual elements can be removed using `element_blank()`

```
ggplot(iris.wide, aes(x = Length, y = Width, col = Part)) +
  geom_jitter(alpha = 0.7) +
  theme(panel.background = element_blank(),
strip.background = element_blank(),
text = element_blank())
```



Aside: The `ggplot` Object



Graphics created using `ggplot()` are stored as an object, rather than an image.

We can assign these objects to a variable:

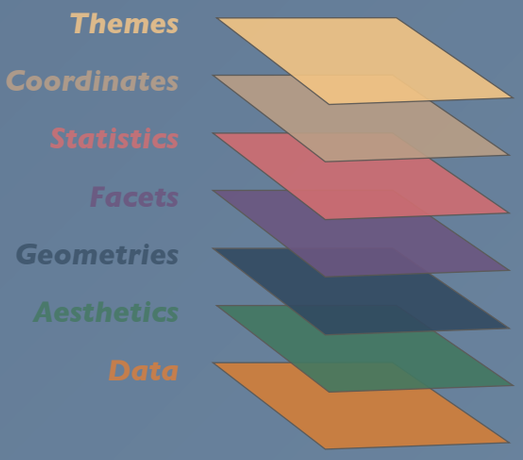
```
p <- ggplot(iris.wide, aes(x = Length, y = Width, col = Part))
```

Grammatical elements can then be added and stored sequentially:

```
p_geom <- p + geom_jitter(alpha = 0.7)

p_stat <- p_geom + geom_smooth()
```

The plot can be displayed by calling the variable.

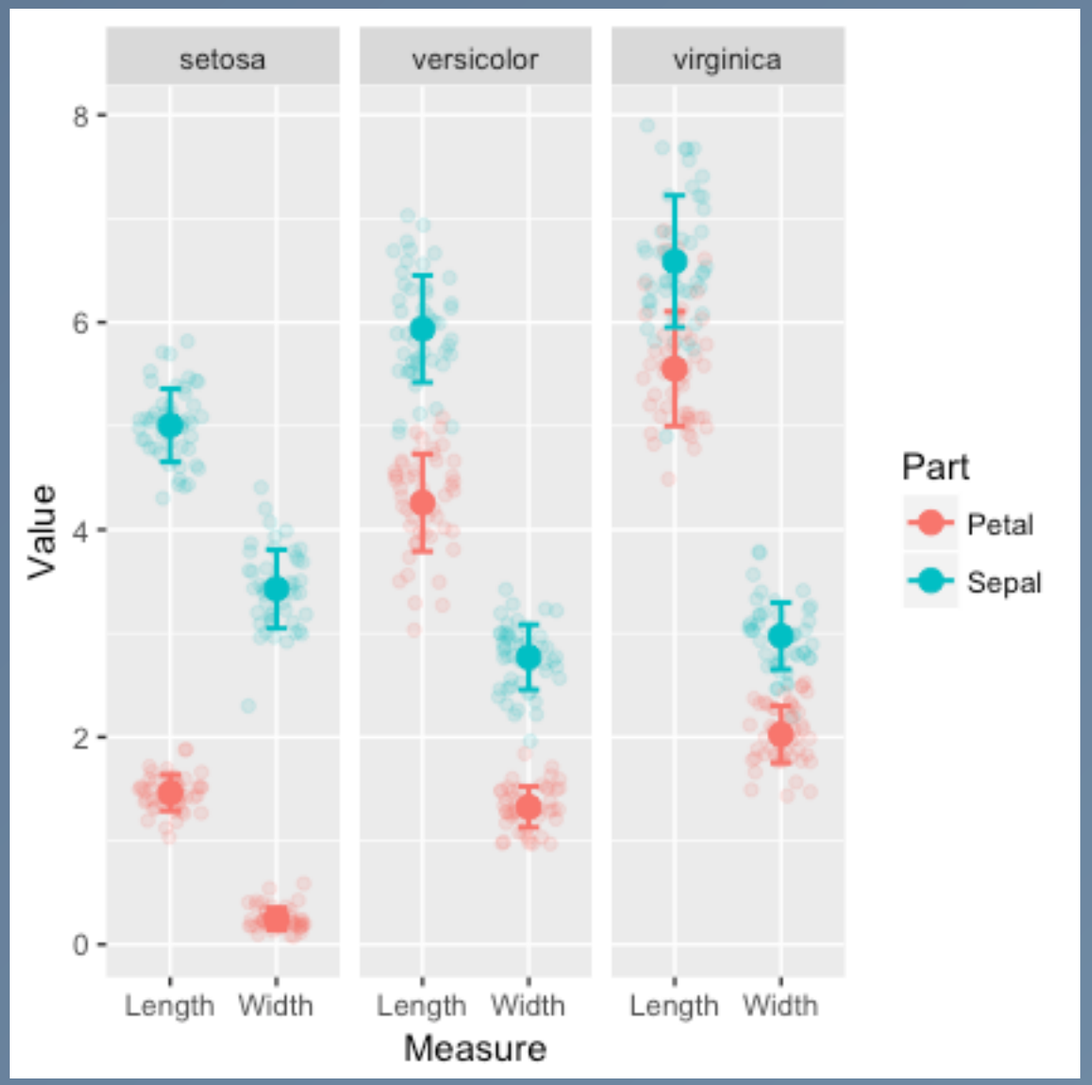
The Themes Layer



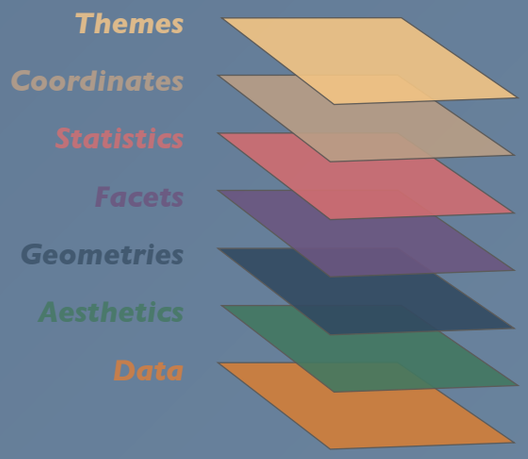
Creating a custom theme

Begin by defining a plot object:

```
myPlot <- ggplot(iris.tidy,
                 aes(x = Measure,
                     y = Value,
                     col = Part)) +
  geom_jitter(alpha = 0.15,
              width = 0.3) +
stat_summary(fun.y = mean,
             geom = "point",
             size = 3) +
stat_summary(fun.data = mean_sdl,
             fun.args = list(mult = 1),
             geom = "errorbar",
             width = 0.2,
             size = 0.75) +
facet_grid(.~Species)
```



The Themes Layer

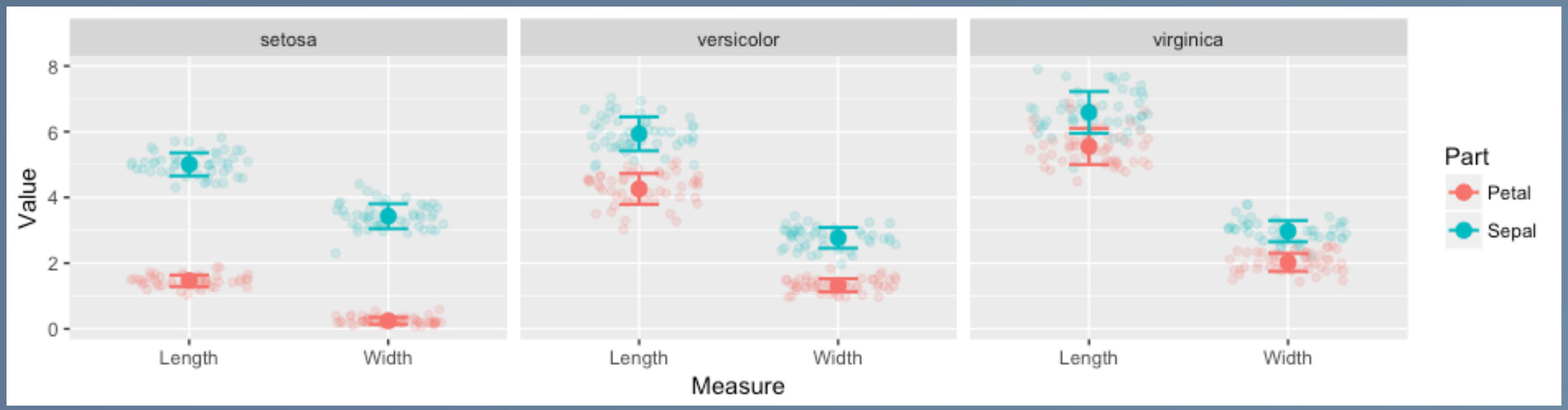


Creating a custom theme

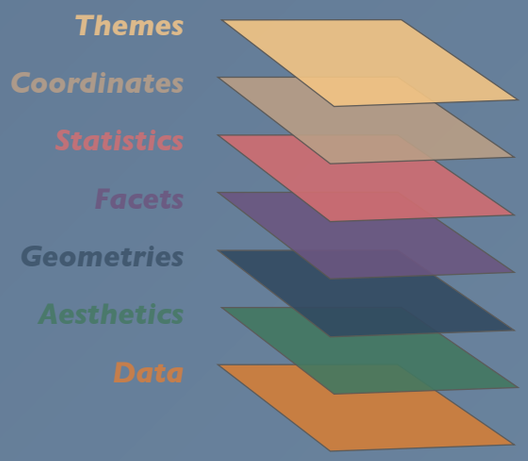
Define some custom colours:

```
backgroundBlue <- "#586D8A"
textGrey <- "#E4E4E4"
headerOrange <- "#E1BC81"
textGreen <- "#ACCDC3"
```

Here is the base plot:

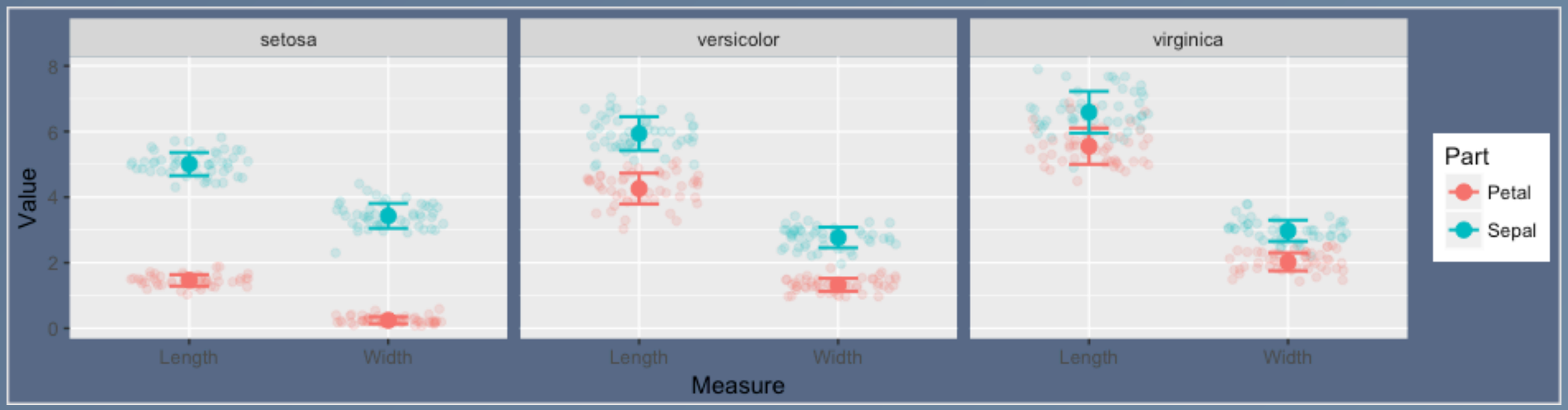


The Themes Layer

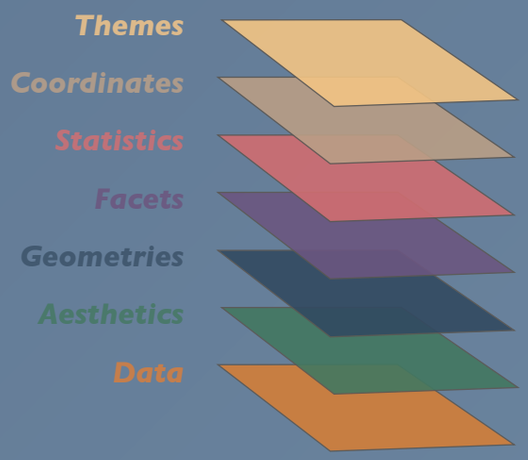


Creating a custom theme: Plot background

```
myPlot +
  theme(plot.background = element_rect(fill = backgroundBlue, color = textGrey, size = 1))
```

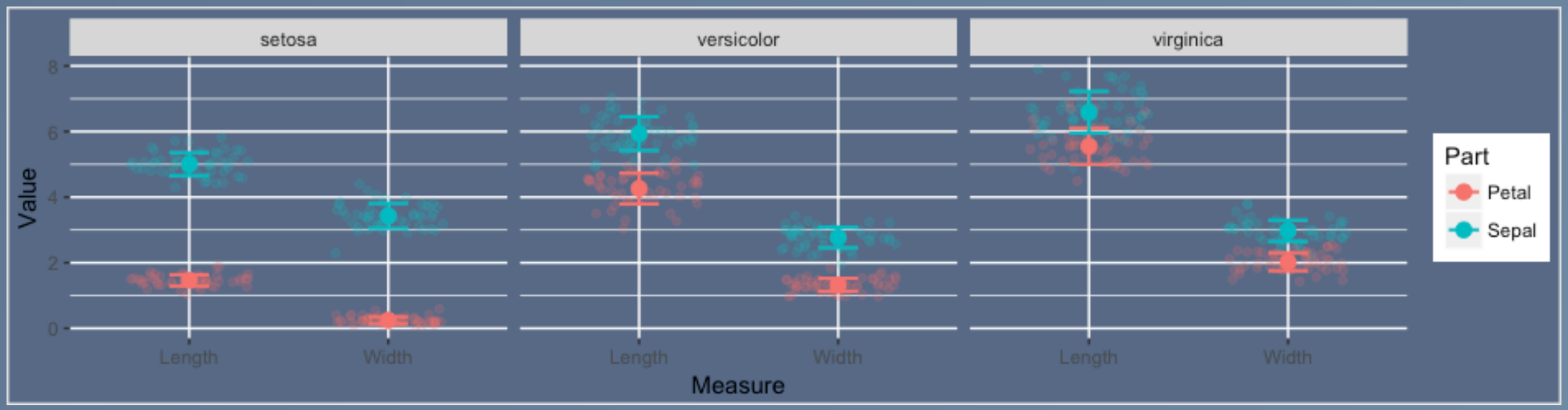


The Themes Layer

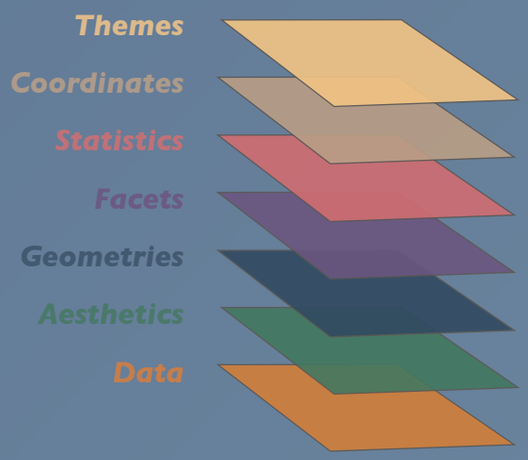


Creating a custom theme: Panel backgrounds

```
myPlot +
  theme(plot.background = element_rect(fill = backgroundBlue, color = textGrey, size = 1),
        panel.background = element_rect(fill = backgroundBlue))
```

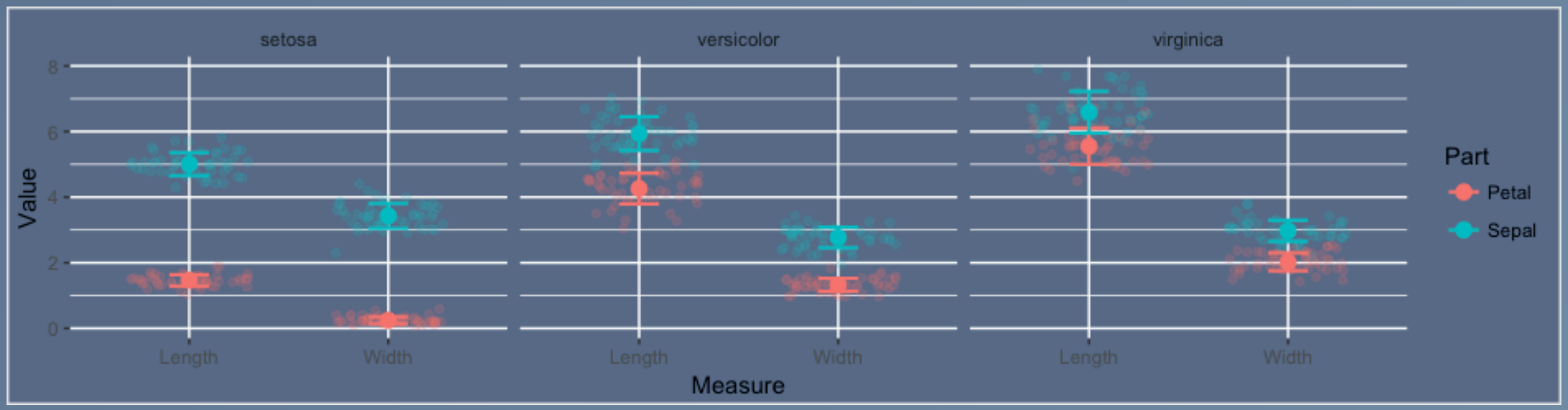


The Themes Layer

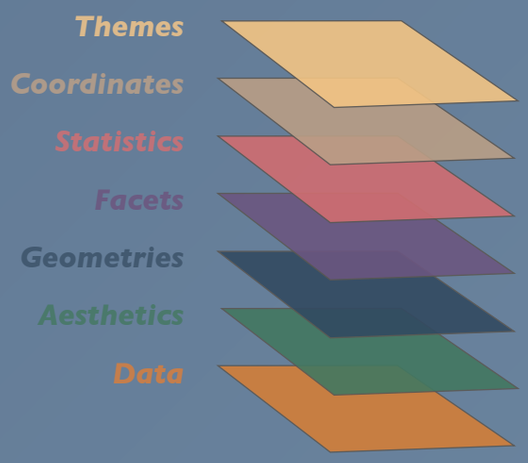


Creating a custom theme: Strips and legend

```
myPlot +
  theme(plot.background = element_rect(fill = backgroundBlue, color = textGrey, size = 1),
        panel.background = element_rect(fill = backgroundBlue),
        strip.background = element_blank(),
        legend.background = element_blank(), legend.key = element_blank())
```

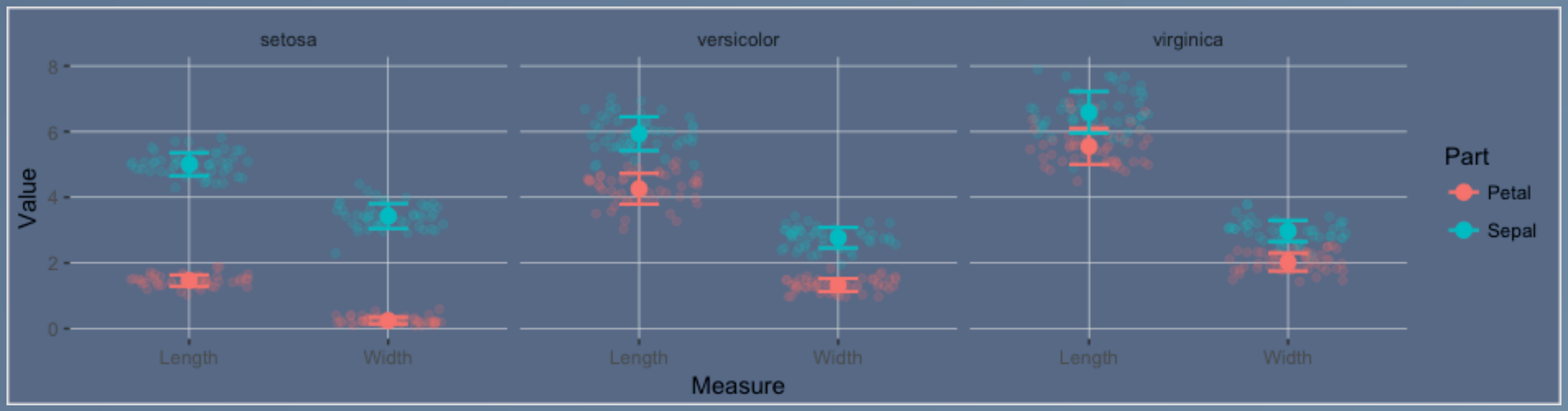


The Themes Layer

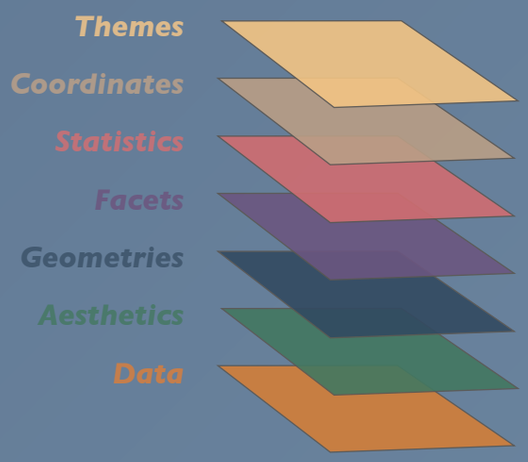


Creating a custom theme: Panel grids

```
myPlot +
  theme(plot.background = element_rect(fill = backgroundBlue, color = textGrey, size = 1),
        panel.background = element_rect(fill = backgroundBlue),
        strip.background = element_blank(),
        legend.background = element_blank(), legend.key = element_blank(),
        panel.grid.major = element_line(color = textGrey, size = 0.2),
        panel.grid.minor = element_blank())
```

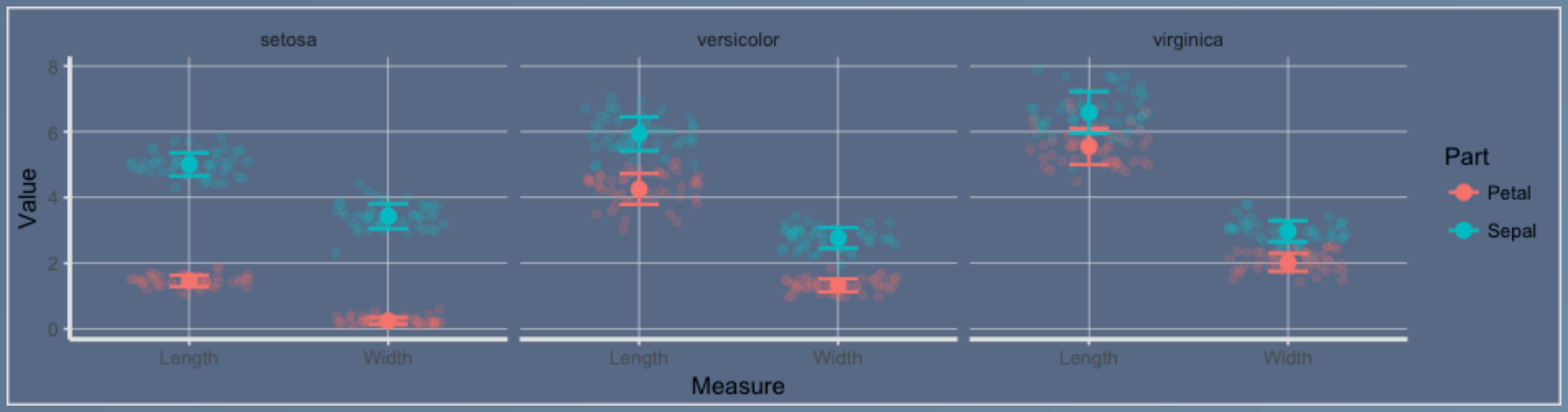


The Themes Layer

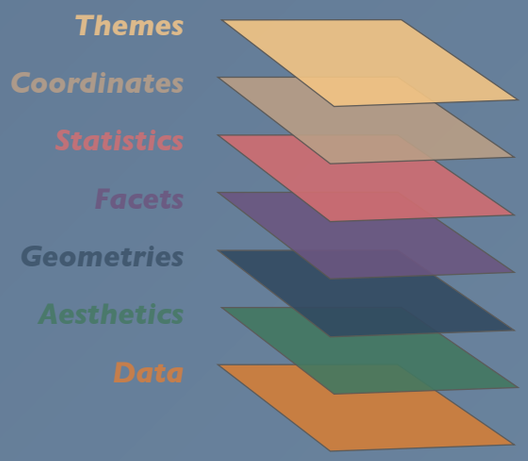


Creating a custom theme: Axis lines and ticks

```
myPlot +
  theme(plot.background = element_rect(fill = backgroundBlue, color = textGrey, size = 1),
        panel.background = element_rect(fill = backgroundBlue),
        strip.background = element_blank(),
        legend.background = element_blank(), legend.key = element_blank(),
        panel.grid.major = element_line(color = textGrey, size = 0.2),
        panel.grid.minor = element_blank(),
        axis.line = element_line(color = textGrey, size = 1),
        axis.ticks = element_line(color = textGrey))
```



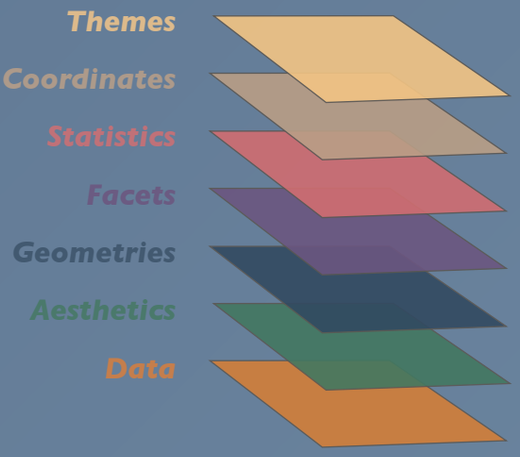
The Themes Layer



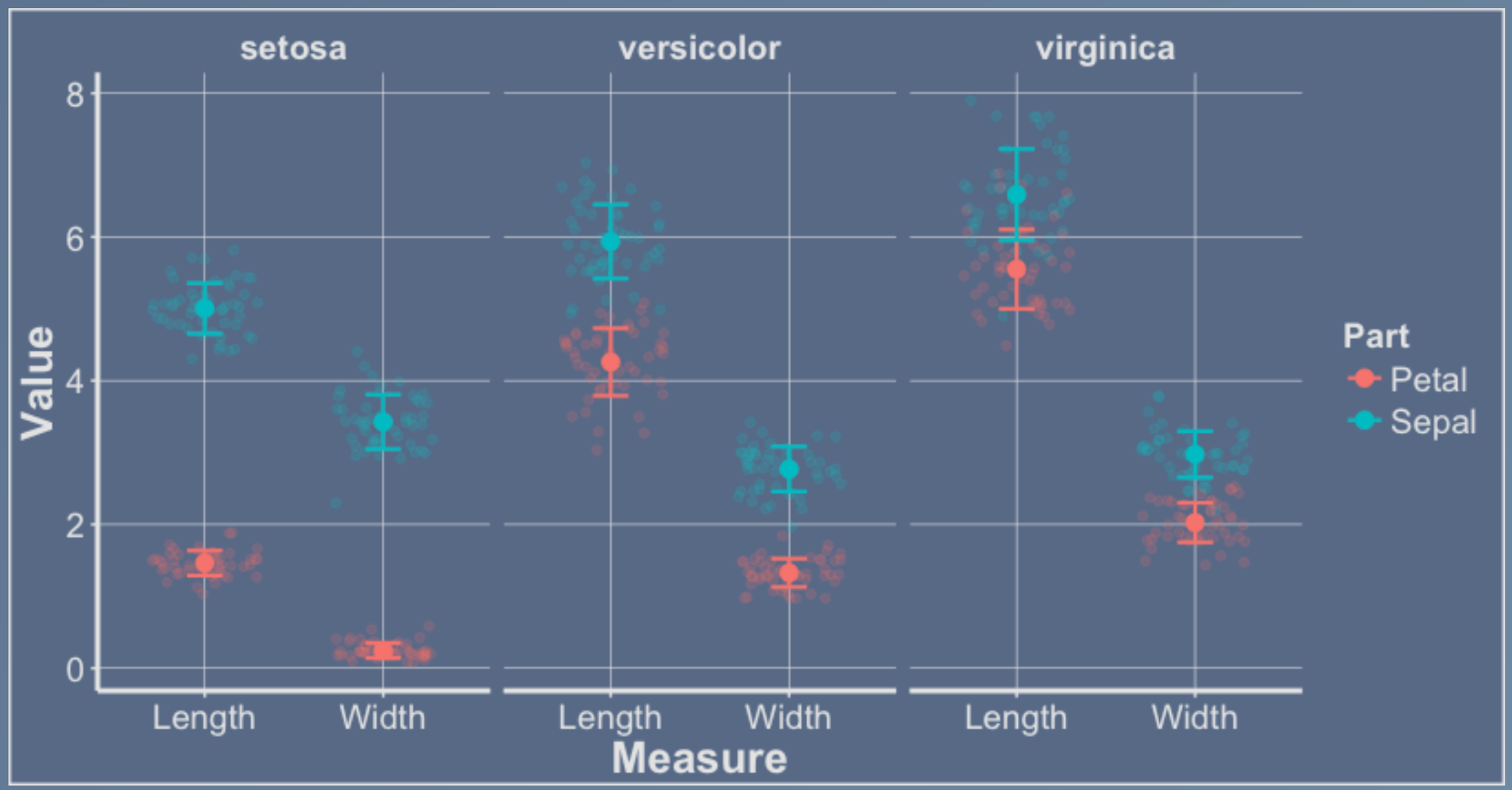
Creating a custom theme: Plot text

```
myPlot +
  theme(plot.background = element_rect(fill = backgroundBlue, color = textGrey, size = 1),
        panel.background = element_rect(fill = backgroundBlue),
        strip.background = element_blank(),
        legend.background = element_blank(), legend.key = element_blank(),
        panel.grid.major = element_line(color = textGrey, size = 0.2),
        panel.grid.minor = element_blank(),
        axis.line = element_line(color = textGrey, size = 1),
        axis.ticks = element_line(color = textGrey),
        axis.text = element_text(color = textGrey, size = 14),
        axis.title = element_text(color = textGrey, face = "bold", size = 18, vjust = 10),
        legend.text = element_text(color = textGrey, size = 14),
        legend.title = element_text(color = textGrey, face="bold", size = 14),
        strip.text = element_text(color = textGrey, face = "bold", size = 14))
```

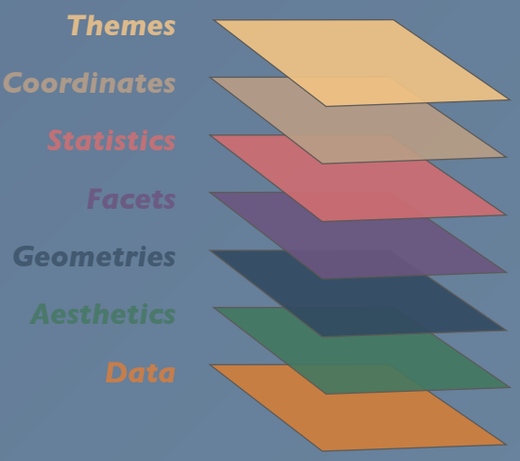
The Themes Layer



Creating a custom theme: Plot text



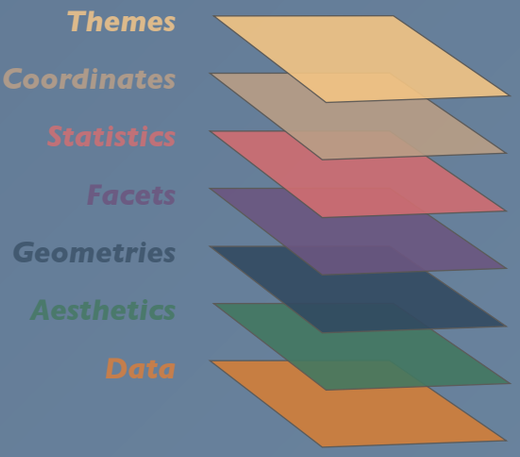
The Themes Layer



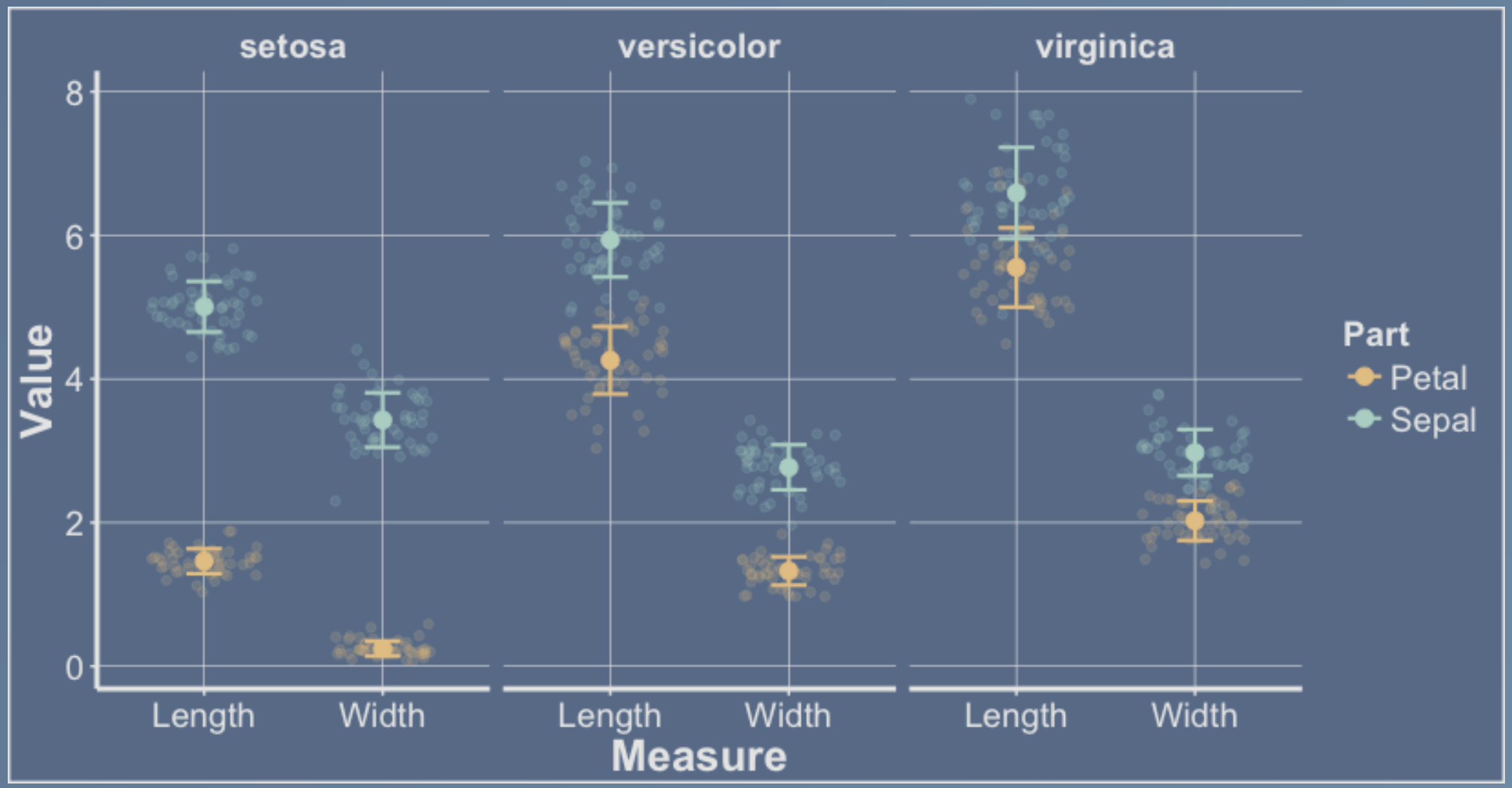
Creating a custom theme: Colour scale

```
myPlot +
  scale_colour_manual(labels = c("Petal", "Sepal"),
                      values = c(headerOrange, textGreen)) +
  theme(plot.background = element_rect(fill = backgroundBlue, color = textGrey, size = 1),
        panel.background = element_rect(fill = backgroundBlue),
        strip.background = element_blank(),
        legend.background = element_blank(), legend.key = element_blank(),
        panel.grid.major = element_line(color = textGrey, size = 0.2),
        panel.grid.minor = element_blank(),
        axis.line = element_line(color = textGrey, size = 1),
        axis.ticks = element_line(color = textGrey),
        axis.text = element_text(color = textGrey, size = 14),
        axis.title = element_text(color = textGrey, face = "bold", size = 18, vjust = 10),
        legend.text = element_text(color = textGrey, size = 14),
        legend.title = element_text(color = textGrey, face="bold", size = 14),
        strip.text = element_text(color = textGrey, face = "bold", size = 14))
```

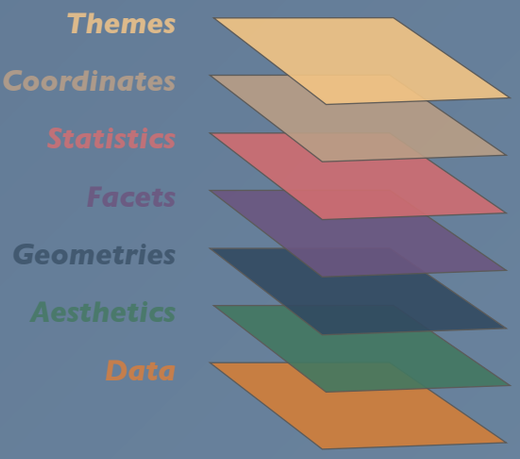
The Themes Layer



Our final plot!



The Themes Layer

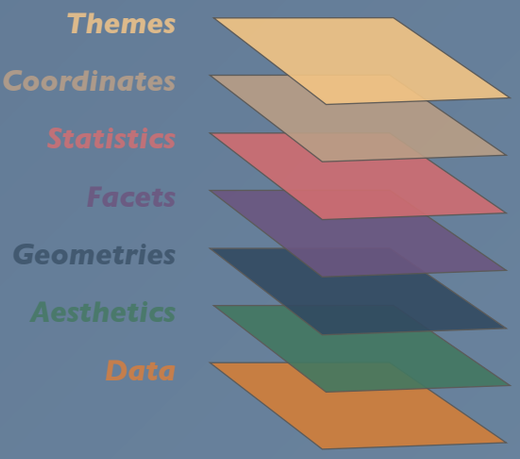


Saving themes

Like plots, themes can be saved for future use:

```
myTheme <- theme(plot.background = element_rect(fill = backgroundBlue,
                                                color = textGrey, size = 1),
                 panel.background = element_rect(fill = backgroundBlue),
                 strip.background = element_blank(),
                 legend.background = element_blank(), legend.key = element_blank(),
                 panel.grid.major = element_line(color = textGrey, size = 0.2),
                 panel.grid.minor = element_blank(),
                 axis.line = element_line(color = textGrey, size = 1),
                 axis.ticks = element_line(color = textGrey),
                 axis.text = element_text(color = textGrey, size = 14),
                 axis.title = element_text(color = textGrey, face = "bold",
                                           size = 18, vjust = 10),
                 legend.text = element_text(color = textGrey, size = 14),
                 legend.title = element_text(color = textGrey, face="bold", size = 14),
                 strip.text = element_text(color = textGrey, face = "bold", size = 14))
```

The Themes Layer



Summary

- The themes layer controls all non-data ink on the plot
- Thematic elements are made up of text, lines and rectangles
- Thematic elements are modified using arguments of the `theme()` function with the appropriate `element_` function
- Themes can be saved as variables for repeated use

Summary

- ggplot2 uses the principles of the *grammar of graphics*
- There are 7 grammatical layers, 3 of which are essential for generating a plot
- **Data** structure plays a crucial role in plotting. Tidy data is preferred
- Plots are generated by mapping variables to visual **aesthetics**
- **Geometries** prescribe how the aesthetic mappings are displayed
- **Facets** are used to create subplots based on subsets of the data
- **Statistics** can be computed and displayed on the fly
- **Coordinates** change the dimensions of the plot while scales filter the aesthetic mappings
- **Themes** allow you to control all visual elements of the graphic unrelated to the data

Resources

DataCamp (https://www.datacamp.com/) courses:

1. Data Visualization with ggplot2 (Part 1)
    - https://www.datacamp.com/courses/data-visualization-with-ggplot2-1
2. Data Visualization with ggplot2 (Part 2)
    - https://www.datacamp.com/courses/data-visualization-with-ggplot2-2

Google and Stack Overflow (https://stackoverflow.com/)