Chapter 4 - The grammar of graphics and the `ggplot2` package

Edson Raul Cepeda Marquez

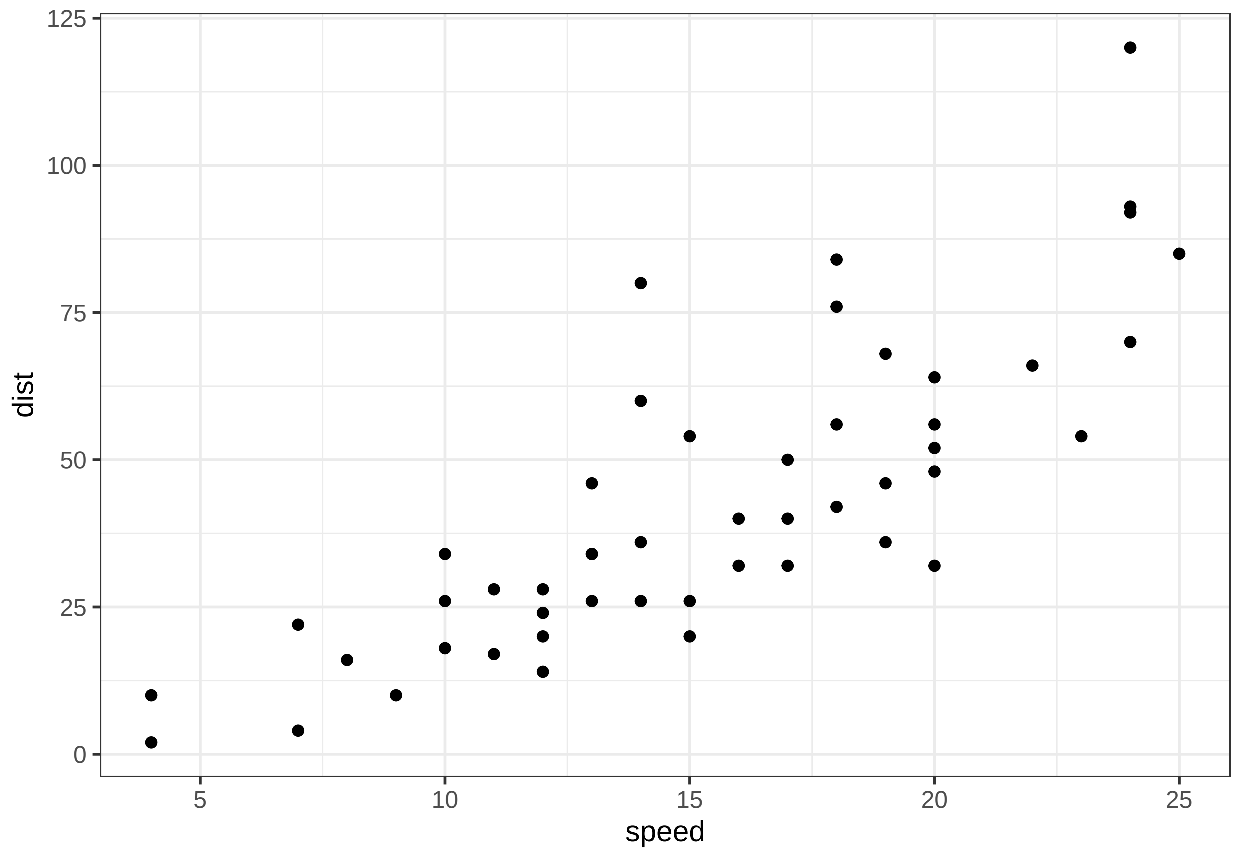
The *ggplot* package provides an alternative to the basic graphics that is based on what is called the "grammar of graphics". **The idea here is that the system gives you small domain-specific language for creating plots**. You construct plots through a list of function calls. *This functions calls do not directly write on a canvas independedntly of each other*. **They manipulate a plot by either modifying it or adding layers of visualization to the plot**.

```
library(ggplot2)
library(magrittr)
library(tidyr)
library(dplyr)
library(knitr)
# library(help = "ggplot2")
```

Using qplot()

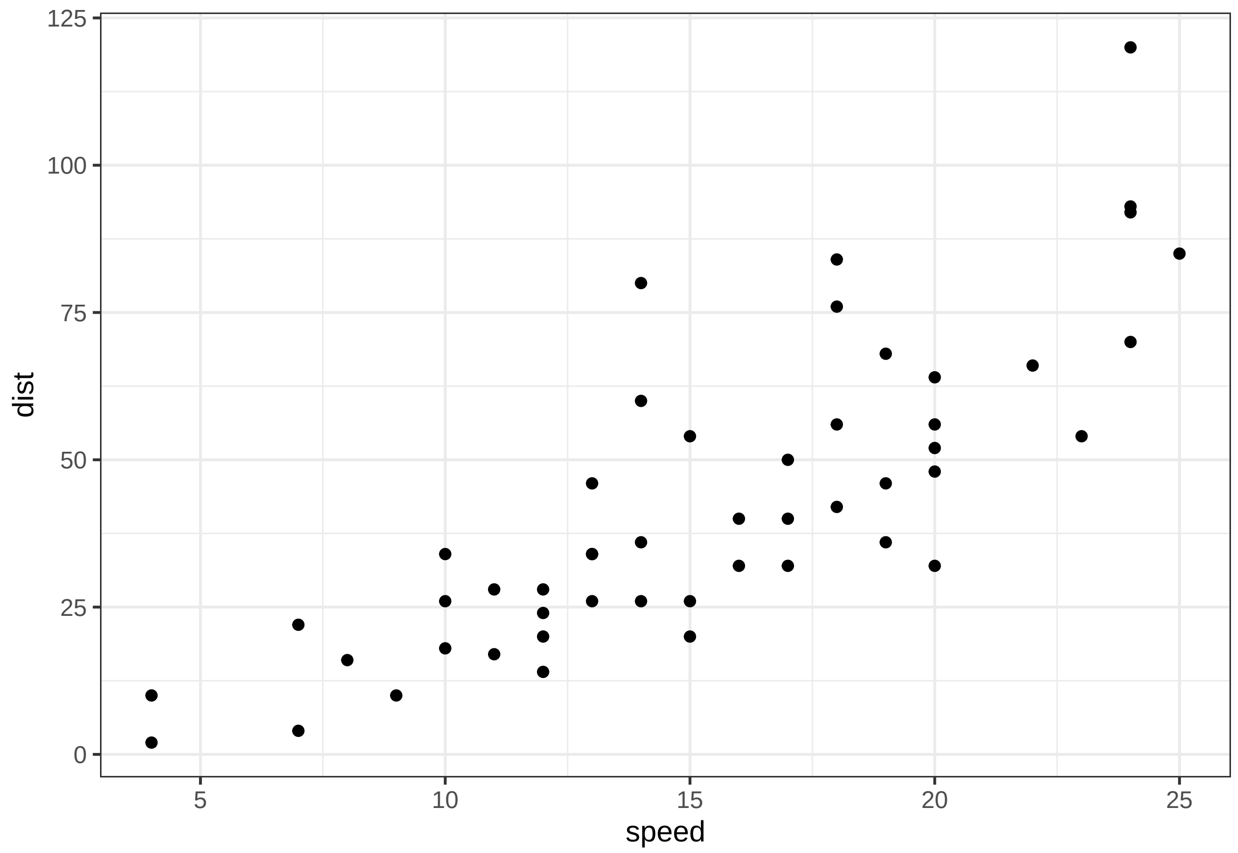
The `qplot()` function can be used to plot simple scatterplots the same way as the `plot()` function.

```
cars %>% qplot(speed, dist, data = .)
```



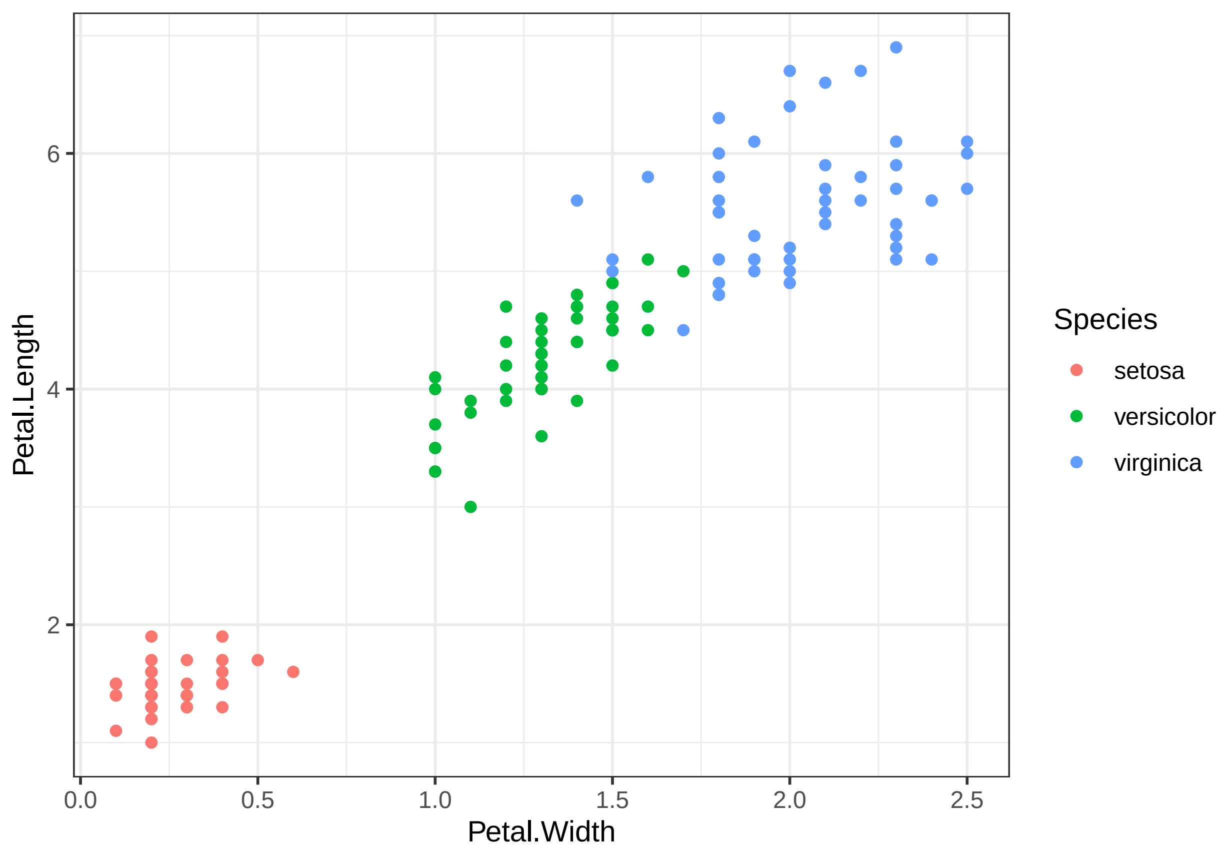
The `qplot()` creates a `ggplot` object rather than directly plotting. THe previous is equivalent to this:

```
p <- cars %>% qplot(speed, dist, data = .)
p
```



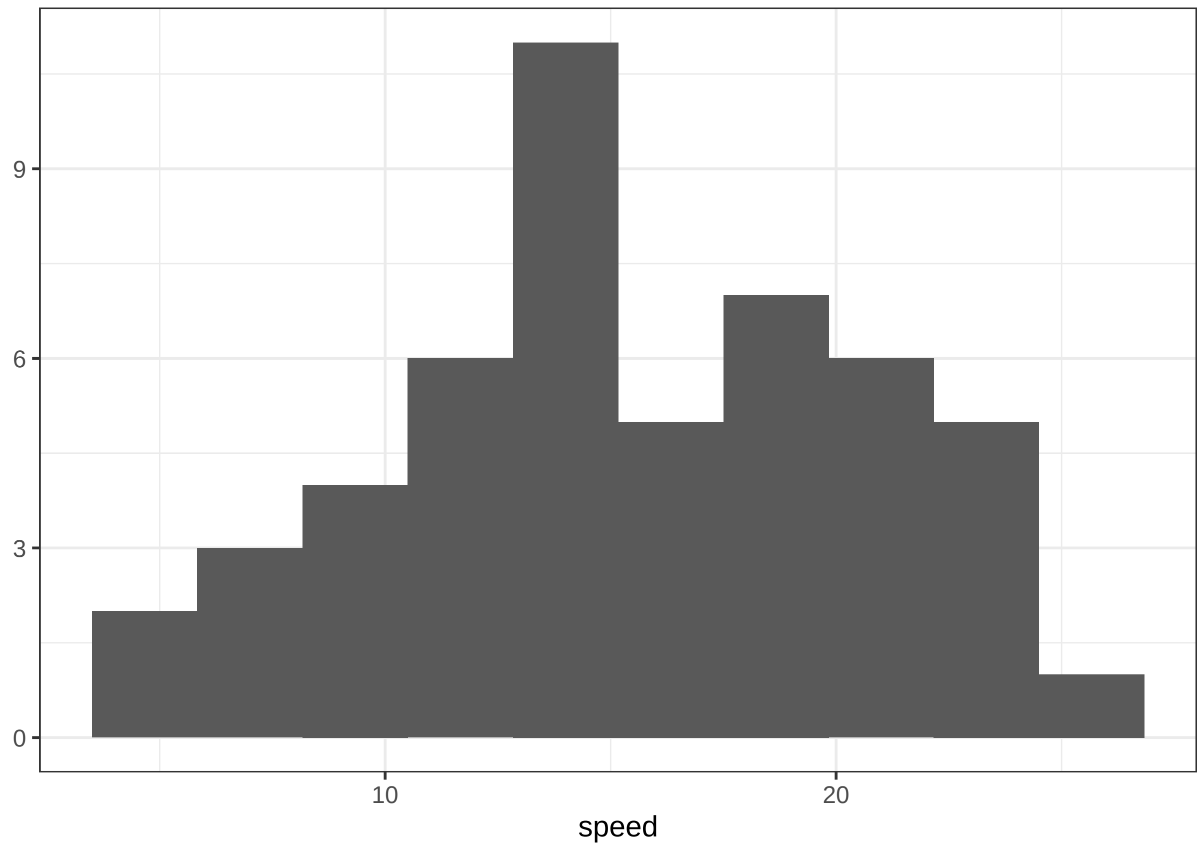
It is not recommended to use `qplot`, it is done here, to show the functionality. With `qplot()`, you can make the visualization of data points depend on data variables in a simpler way. With `qplot()`, you just specify that you want the color to depend on the Species variable:

```
iris %>% qplot(Petal.Width, Petal.Length, color = Species, data = .)
```



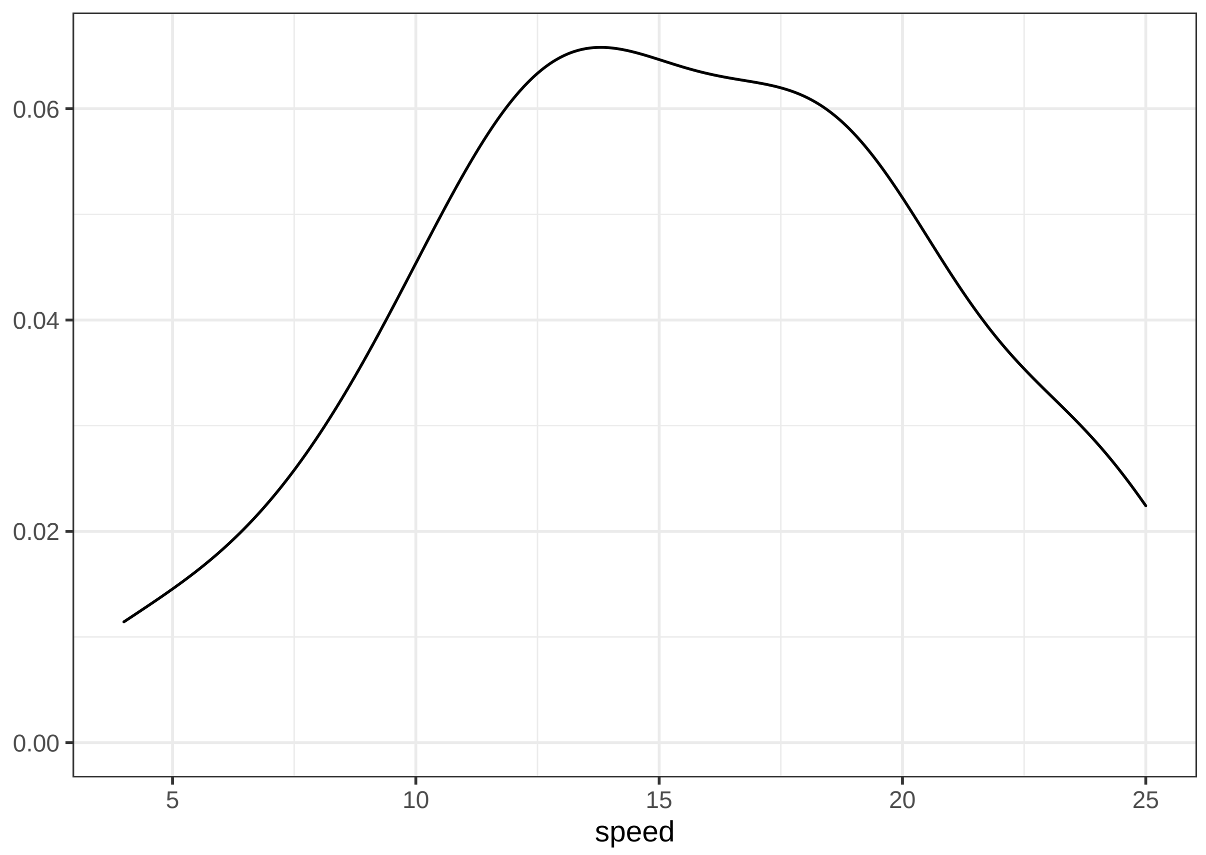
You can use `qplot()` for other types of plots other than scatterplots. If you give it a single variable to plot, it will assume that you want a histogram instead of a scatterplot.

```
cars %>% qplot(speed, data = ., bins = 10)
```



If you want a density plot instead:

```
cars %>% qplot(speed, data = ., geom = "density")
```

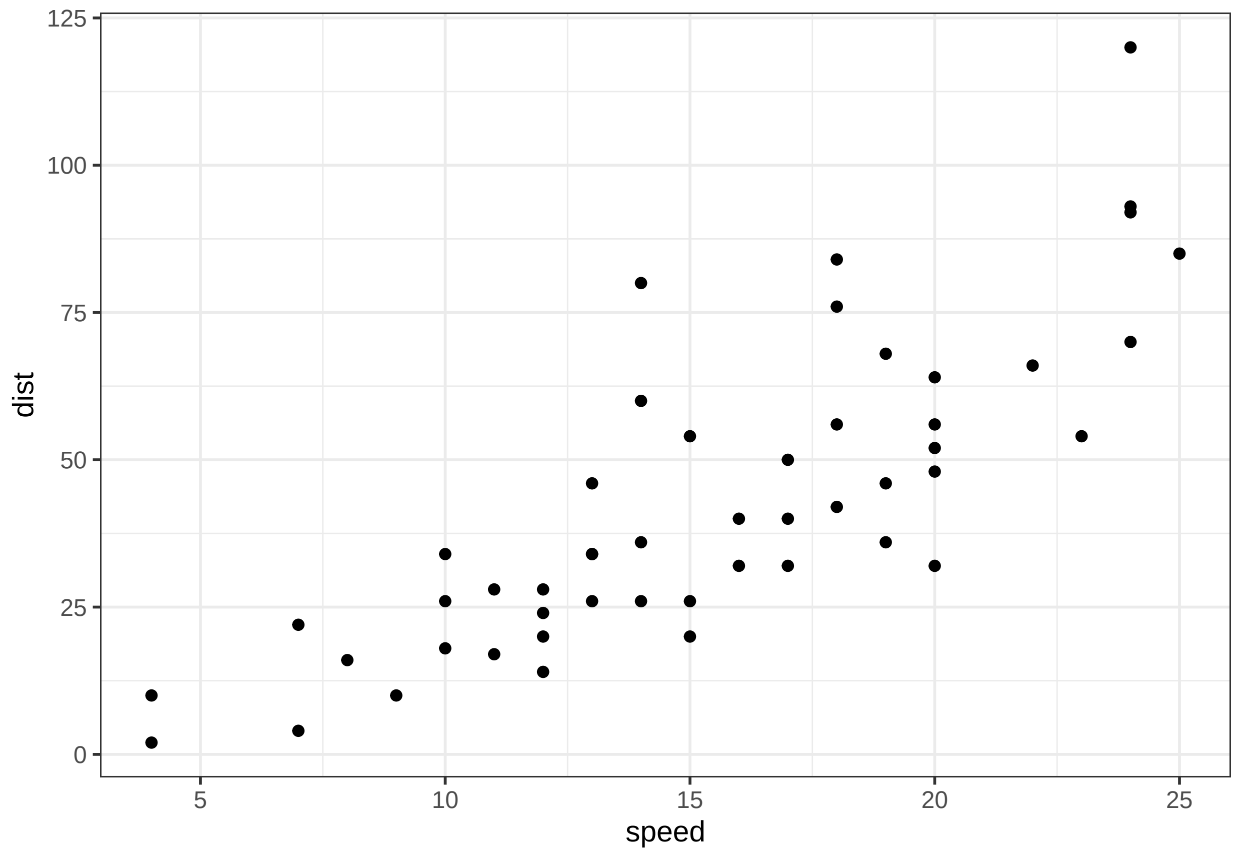


Using Geometries

To create this plot using explicit geometries we want a `ggplot` object, we need to map the *speed* parameter from the data frame to the x-axis and the *dist* parameter to the y-axis, and we need to plot the data as points.

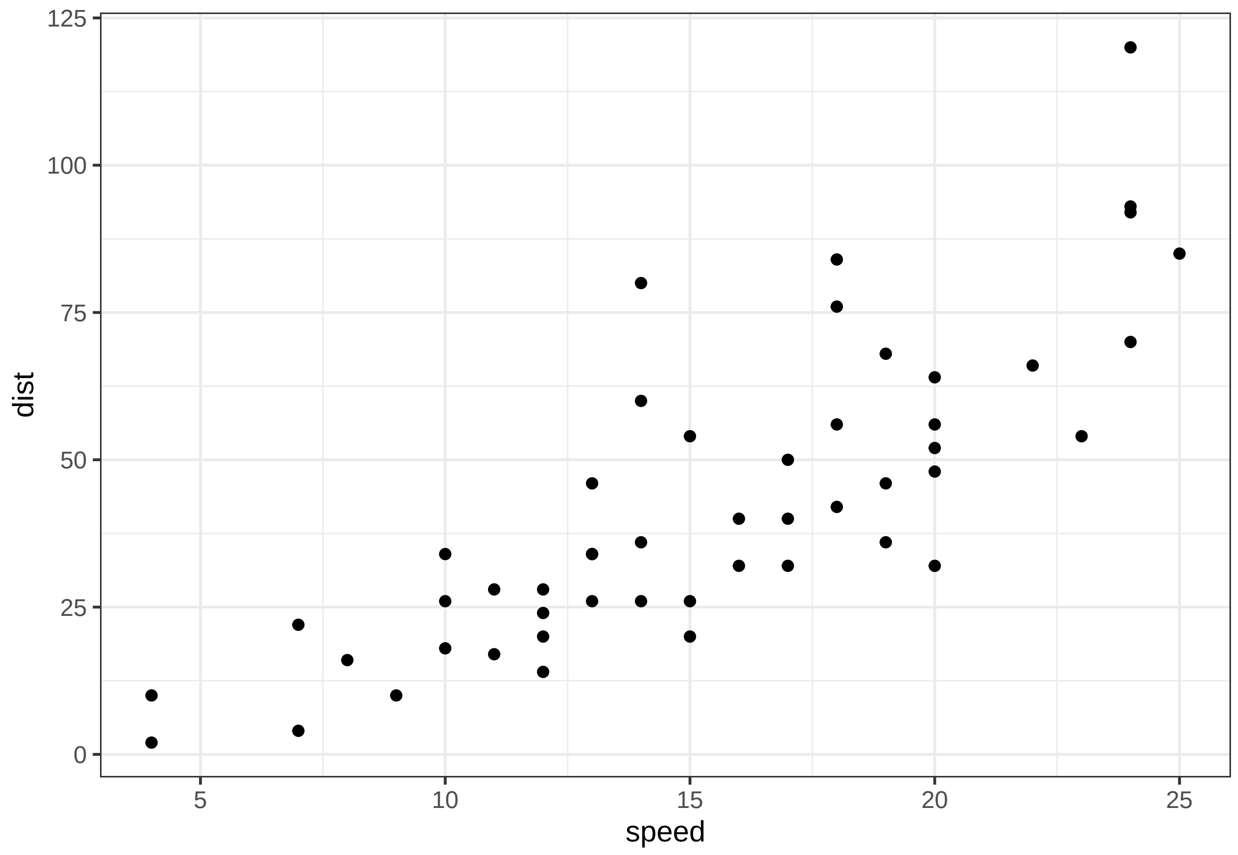
The procedure is as follows (Different data mapping to different geometries): - We create a `ggplot` object and we give *cars* (data frame) as input. - We specify that we want this to be plotted as points with `geom_points()`. - Finally we use the `aes()` function inside `geom_points()` to map the data as the `geom_point()` function needs a x and y value. The `aes()` function is responsible for mapping the data to graphics.

```
ggplot(cars) + geom_point(aes(x = speed, y = dist))
```



When we want to share aesthetics between functions, we cn set it in the `ggplot()` function. Then the next functions can acces to the data and we don't have to specify it for each subsequent function.

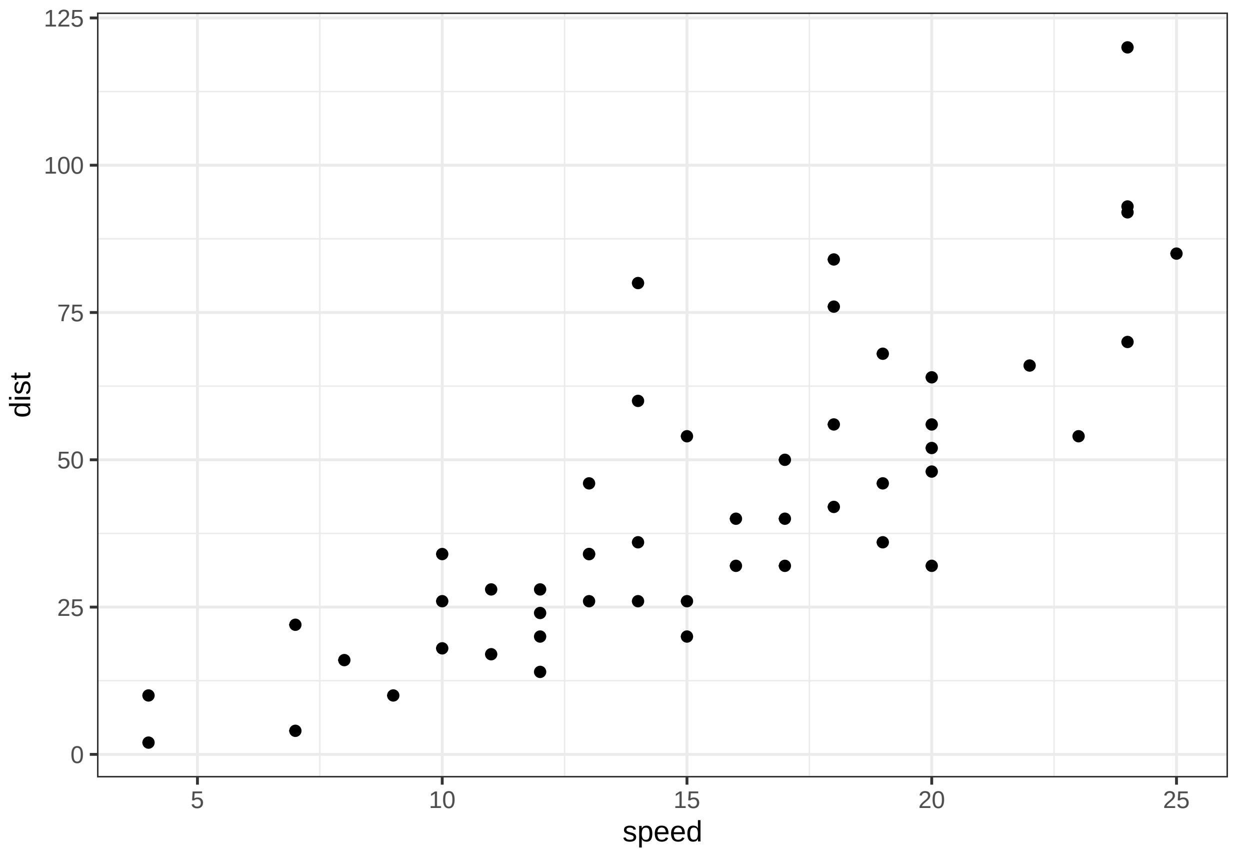
```
ggplot(cars, aes(x = speed, y = dist)) + geom_point()
```



You use + to string together a series of commands to modify a `ggplot` object.

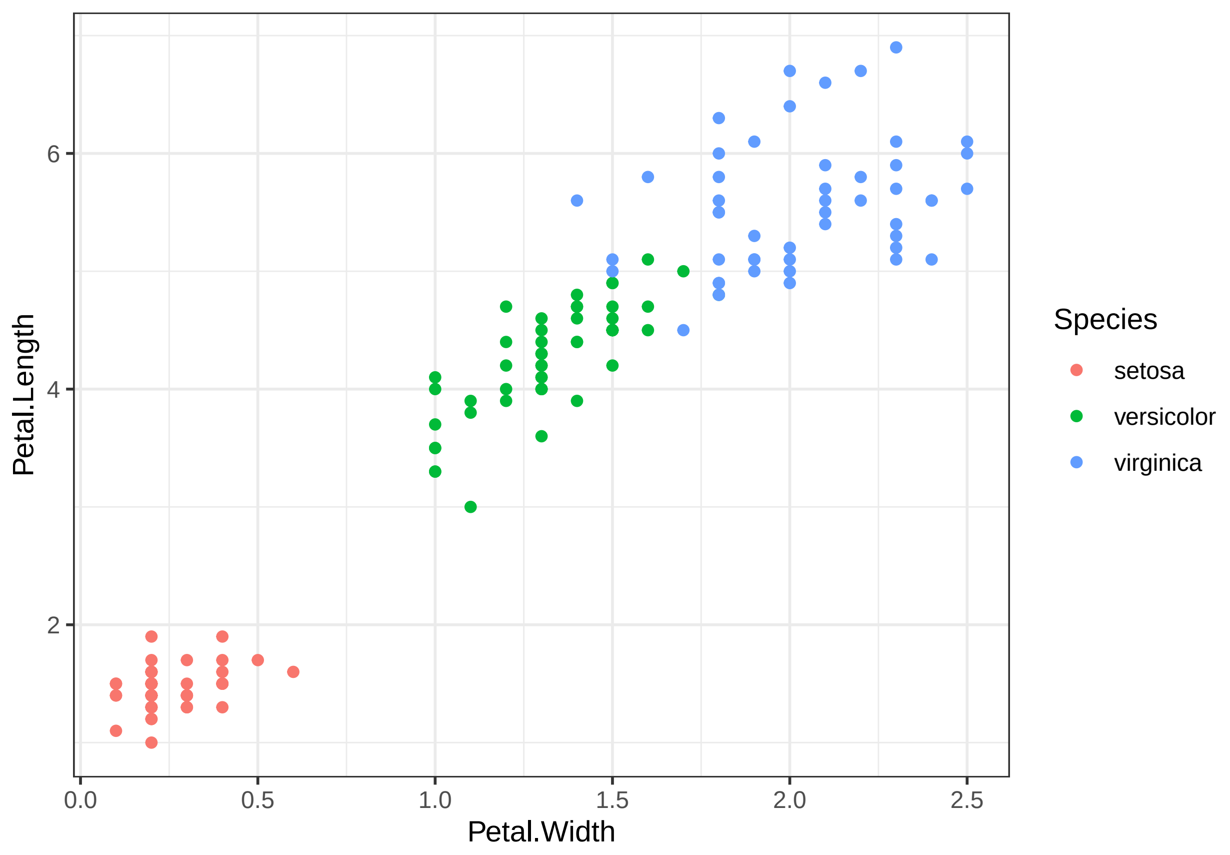
A typical pattern is to first modify data in a string of %>% operations and then give it to `ggplot()` and follow that with a series of + operations.

```
cars %>% ggplot(aes(x = speed, y = dist)) + geom_point()
```



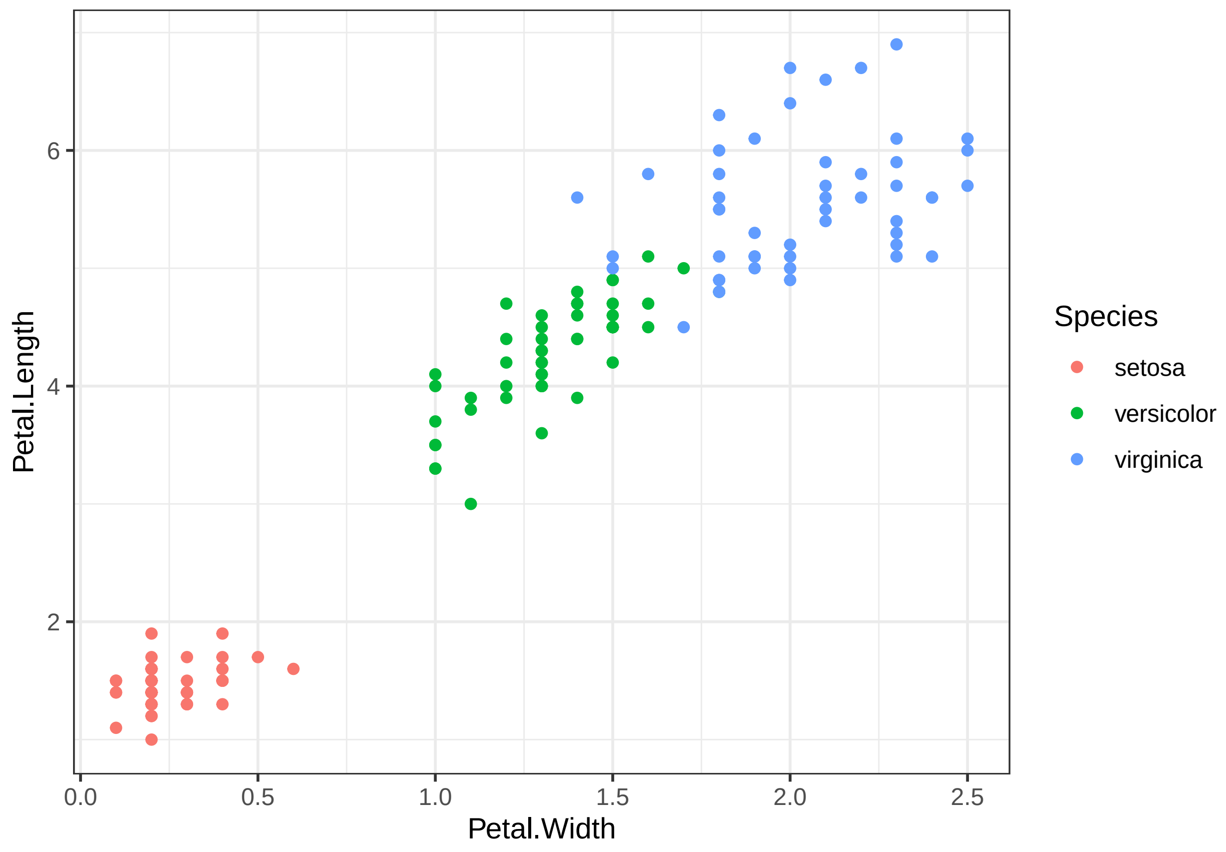
For the *iris* data we used `qplot()` call to create a scatter plot with colors determined by the *Species* variable.

```
iris %>% qplot(Petal.Width, Petal.Length, color = Species, data = .)
```



The corresponding code using `ggplot()`and `geom_points()` looks like this:

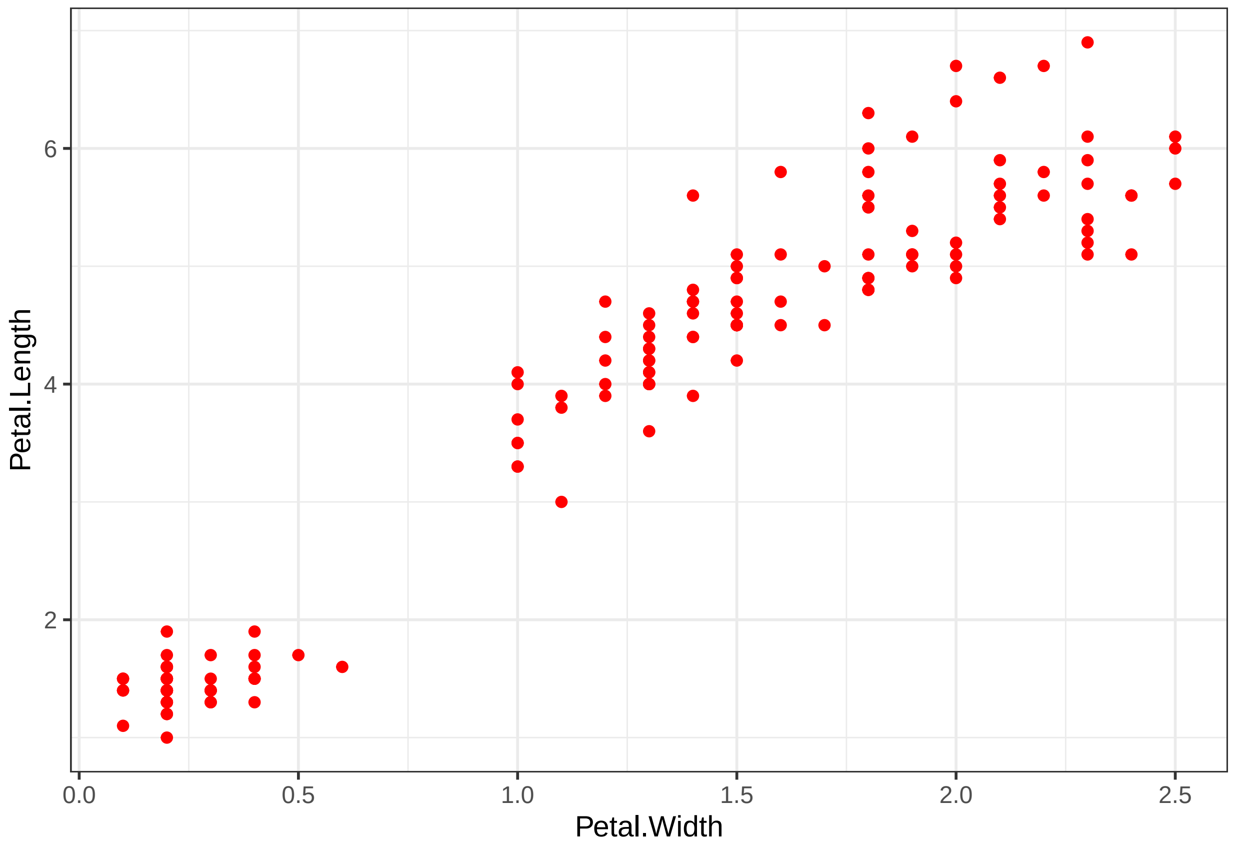
```
iris %>% ggplot + geom_point(aes(x = Petal.Width, y = Petal.Length, color = Species))
```



```
# ggplot(iris) + geom_point(aes(x = Petal.Width, y = Petal.Length, color=Species))
```

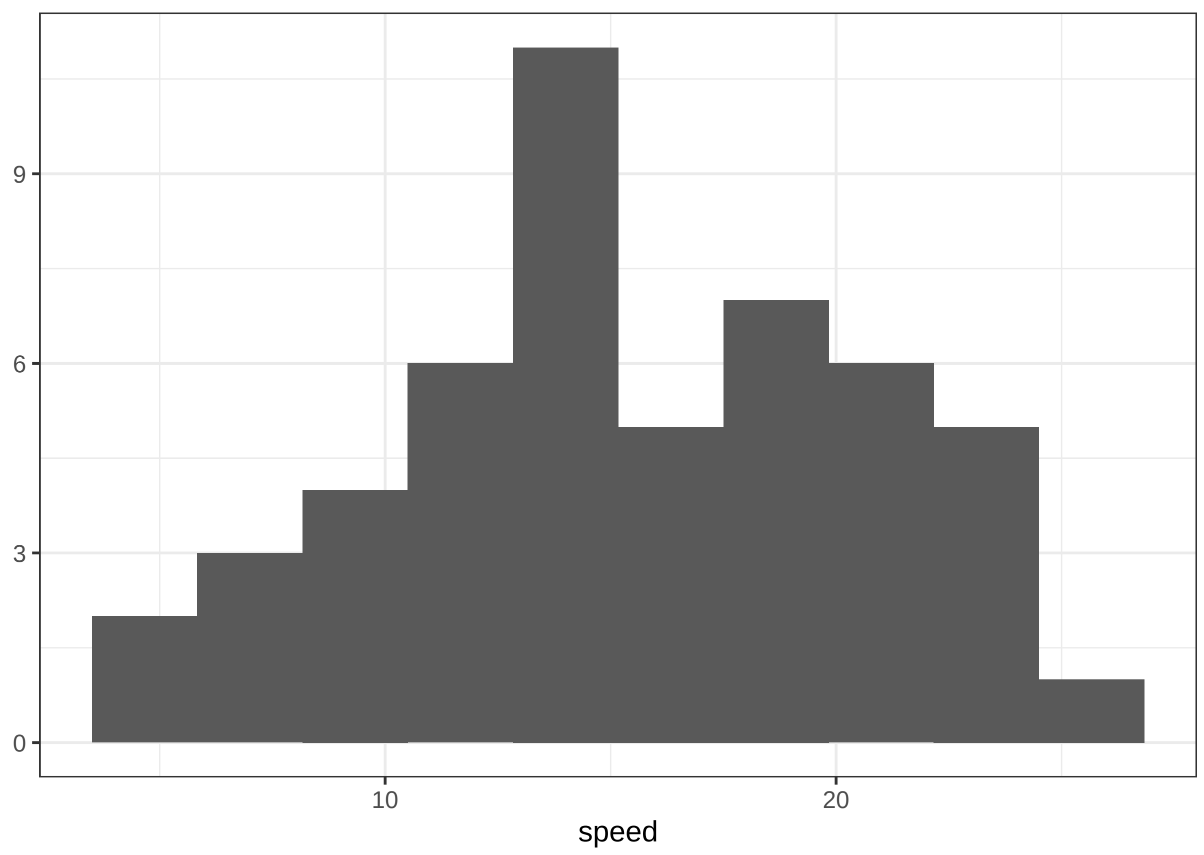
If you instead want to hard-wire the color, or any graphics parameter in general, you simply have to move the parameter assignment outside the `aes()` call.

```
iris %>% ggplot + geom_point(aes(x = Petal.Width, y = Petal.Length), color = "red")
```

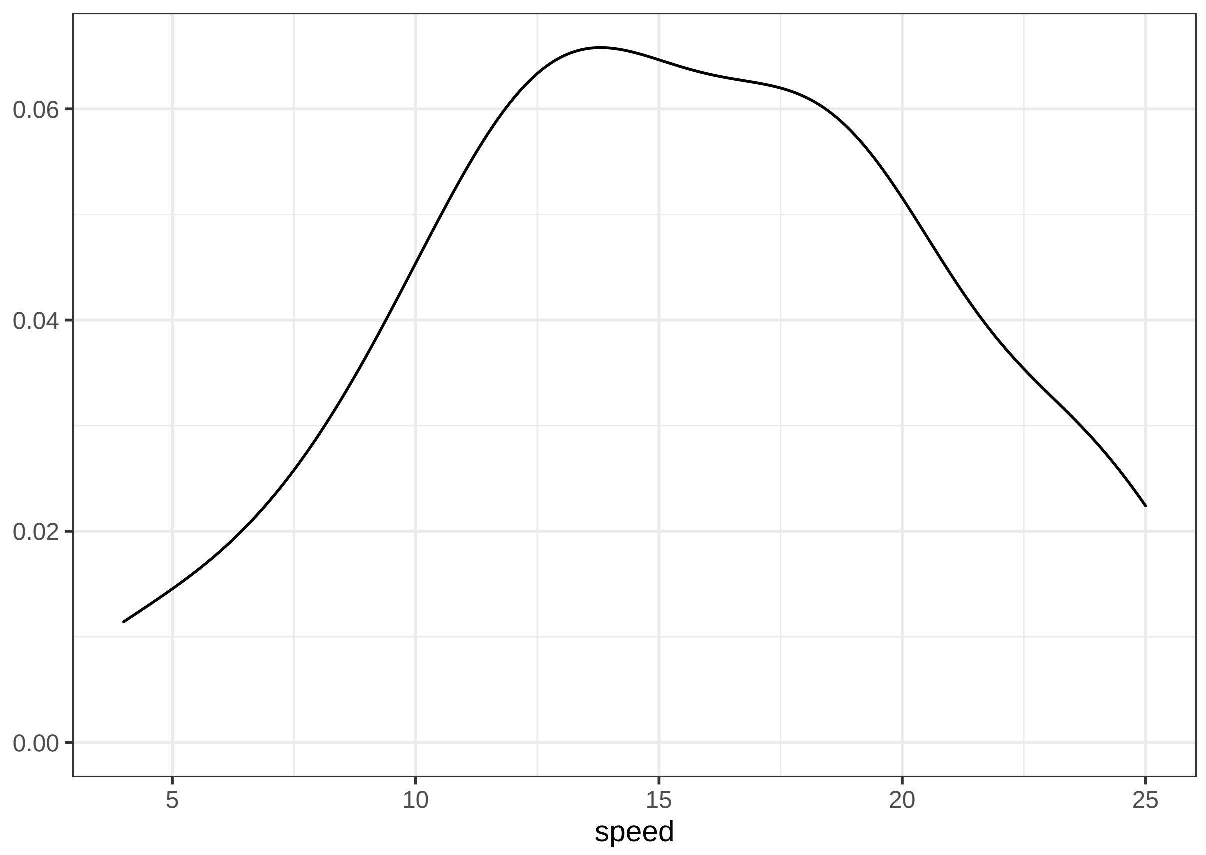


The `qplot()` code for plotting the histogram and a density plot:

```
cars %>% qplot(speed, data = ., bins = 10)
```

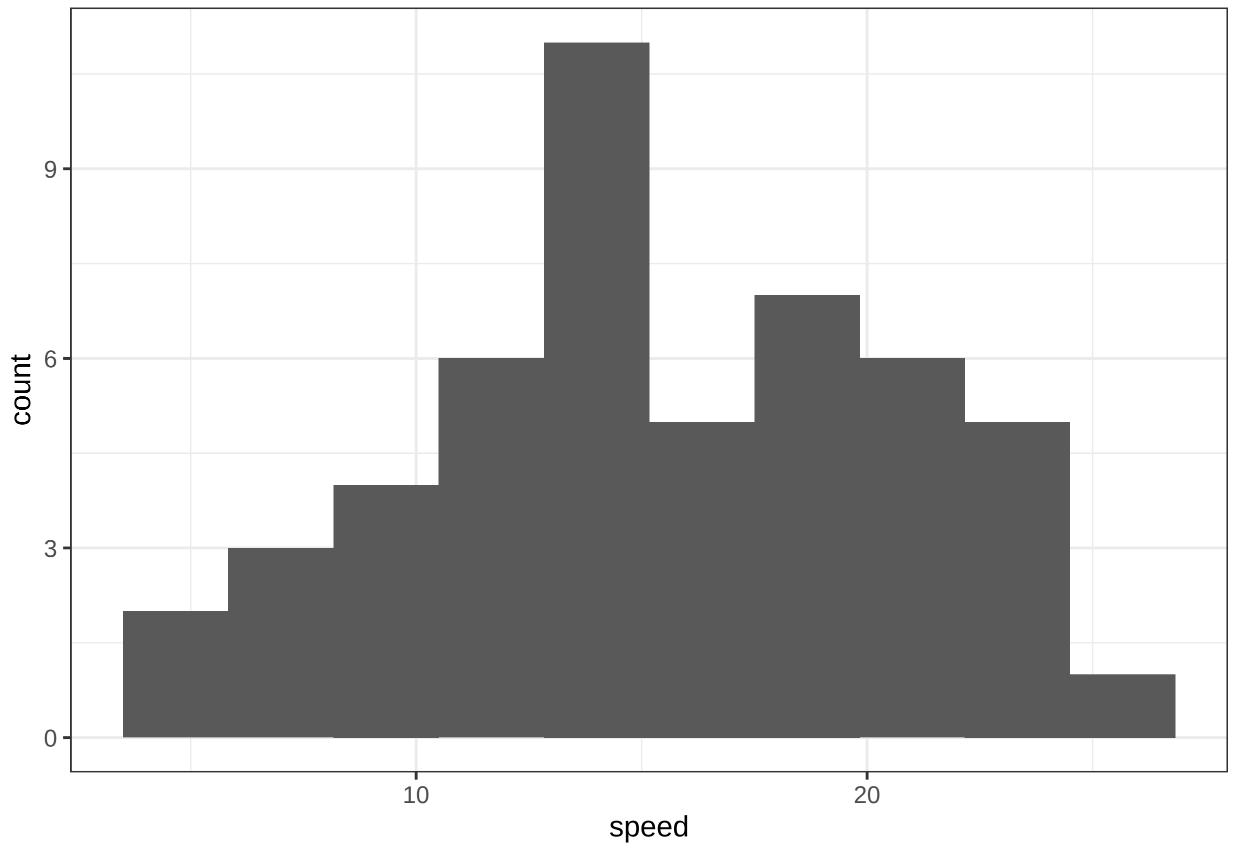


```
cars %>% qplot(speed, data = ., geom = "density")
```

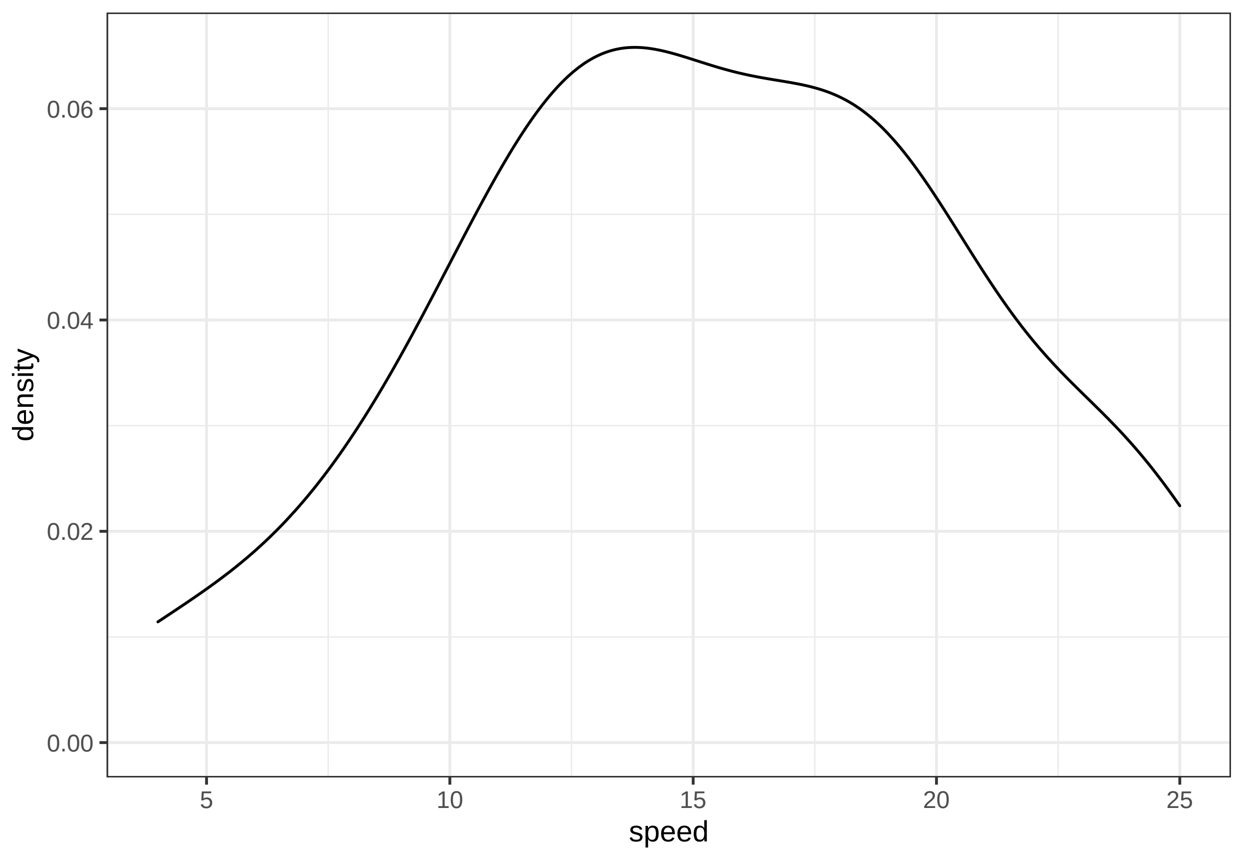


Can be constructed using `geom_histogram()` and `geom_density()`:

```
cars %>% ggplot + geom_histogram(aes(x = speed), bins = 10)
```



```
cars %>% ggplot + geom_density(aes(x = speed))
```

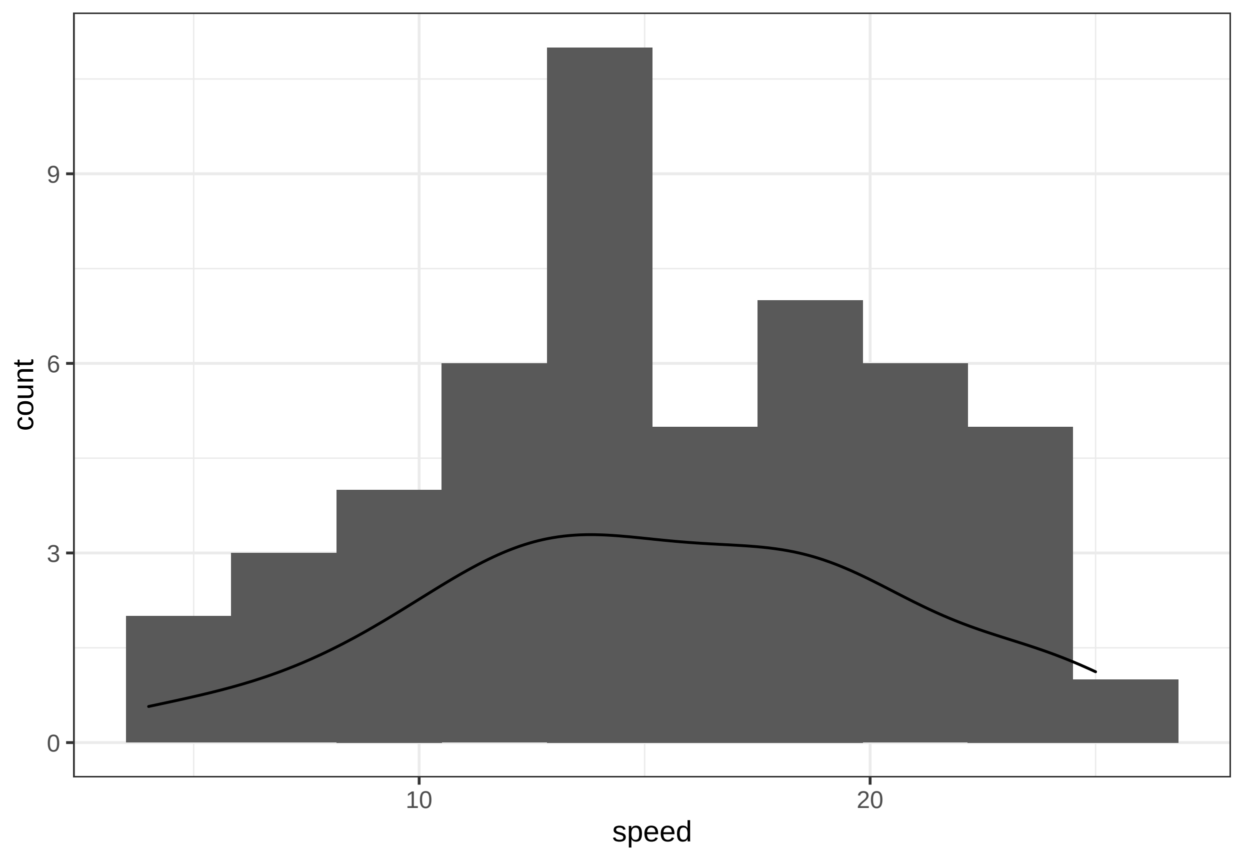


You can combine more geometries to display the data in more than one way.

You do also need to add an extra aesthethics option for the y value. This is because histograms by default will show the counts of how many observationss gall within a bin on the y-axis, while densities integrate to one.

By setting y = ..count.., you tell both geometries to use counts as the y-axis. To get densities instead, you can use y = ..density…

```
cars %>% ggplot(aes(x = speed, y = ..count..)) +
    geom_histogram(bins = 10) +
    geom_density()
```

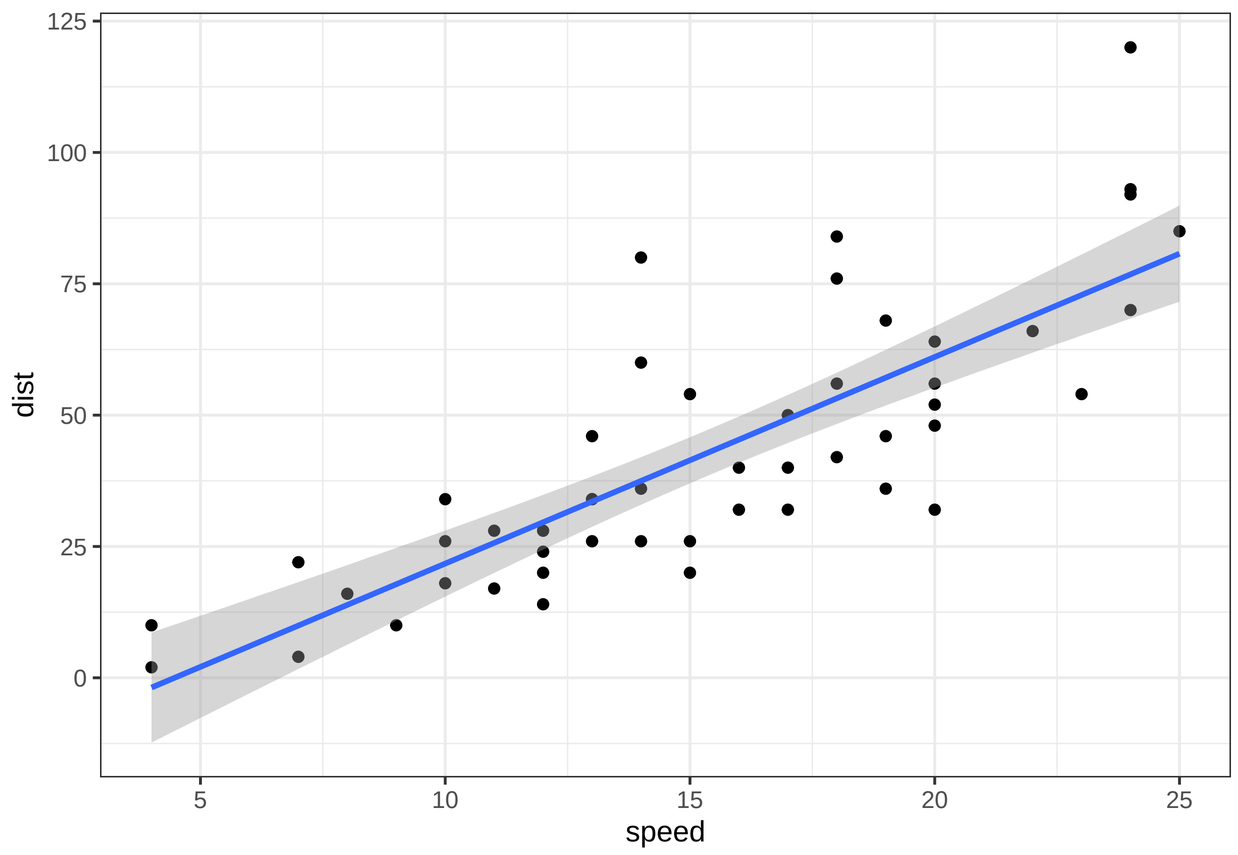


```
# cars %>% ggplot(aes(x = speed, y = ..dednsity..)) +
#     geom_histogram(bins = 10) +
#     geom_density()
```

You can also use combinations of geometries to shwo summary statistics of data together with a scatterplot. We added the result of a linear fit of the data to the scatterplot created for the cars. To do the same with `ggplot2`, you add `geom_smooth`.

```
cars %>% ggplot(aes(x = speed, y = dist)) +
    geom_point() + geom_smooth(method = "lm")
```

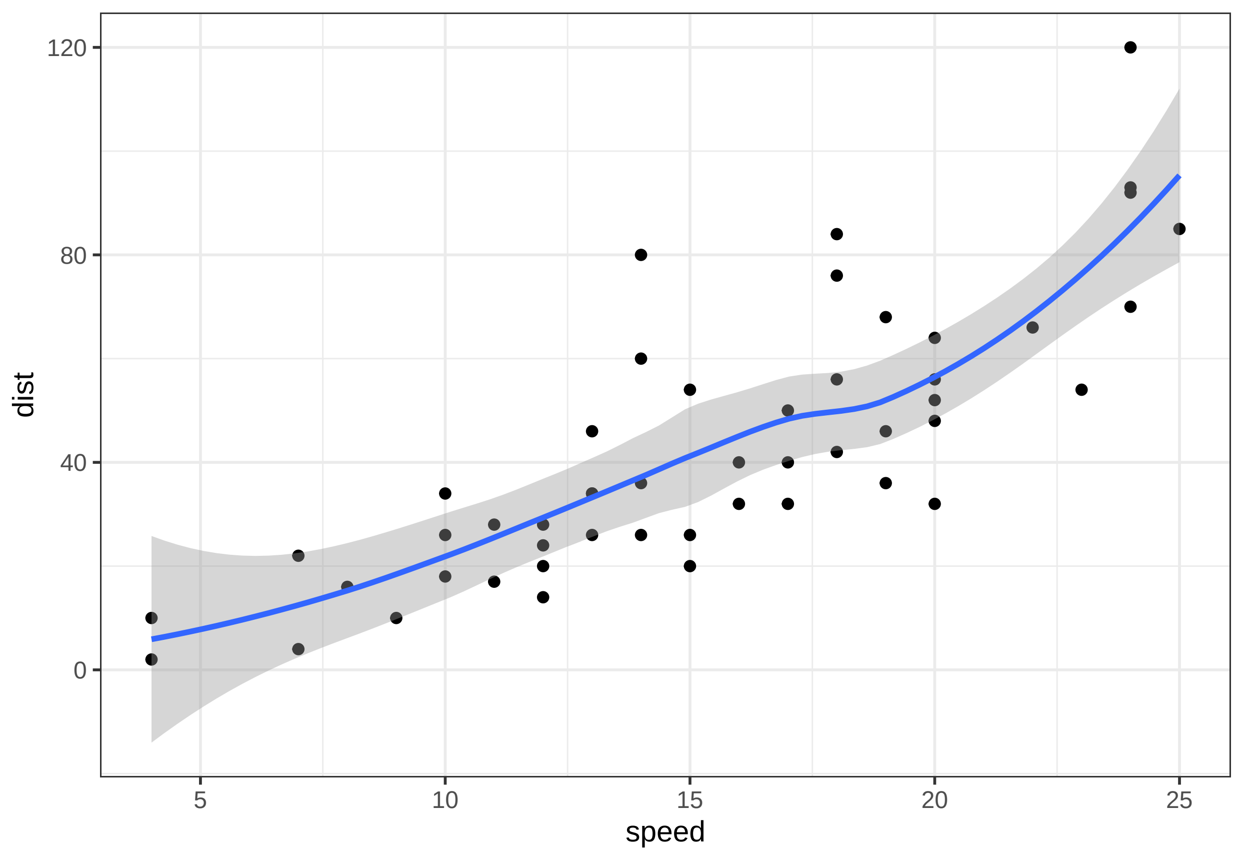
```
## `geom_smooth()` using formula 'y ~ x'
```



If you don't specify `geom_smooth()` to use the linear modedl method, it will instead plot a loess smoothing.

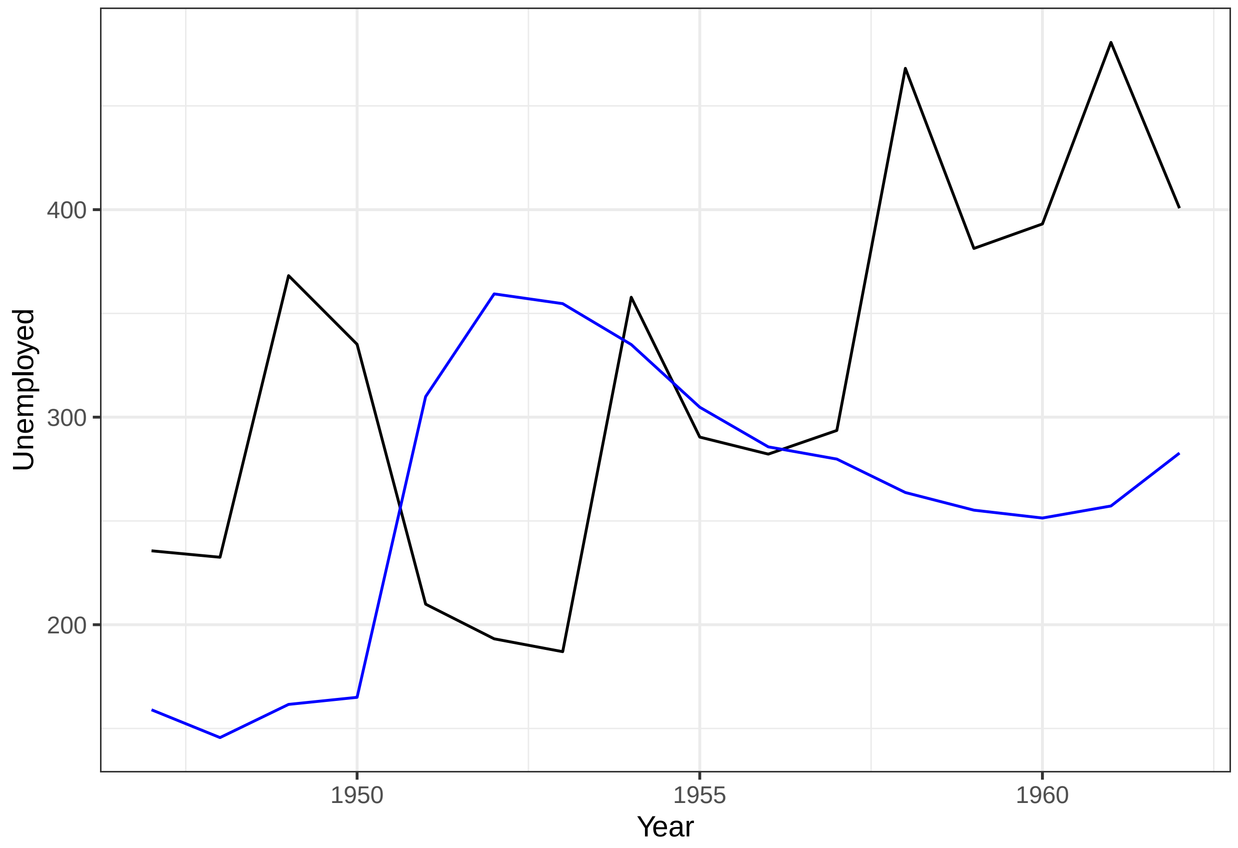
```
cars %>% ggplot(aes(x = speed, y = dist)) +
    geom_point() + geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



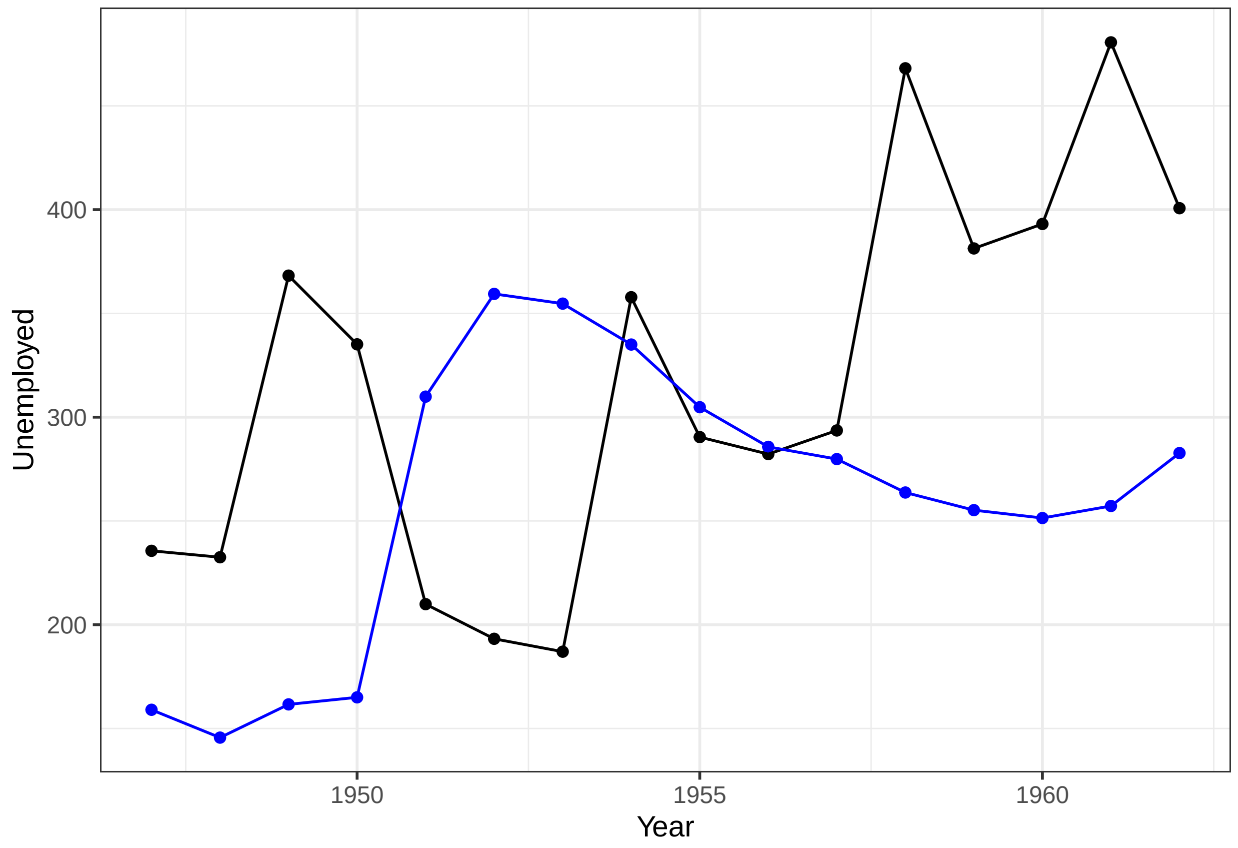
You can also use more than one geometry to plot more than one variable. For the *longley* data you could use two diffent `geom_line()` to plot the *Unemployed* and the *Armed.Forces* data.

```
longley %>% ggplot(aes(x = Year)) +
    geom_line(aes(y = Unemployed)) +
    geom_line(aes(y = Armed.Forces), color = "blue")
```



You can also combine `geom_line()` and `geom_point()` to get both lines and points for your data:

```
longley %>% ggplot(aes(x = Year)) +
    geom_point(aes(y = Unemployed)) +
    geom_point(aes(y = Armed.Forces), color = "blue") +
    geom_line(aes(y = Unemployed)) +
    geom_line(aes(y = Armed.Forces), color = "blue")
```



```
# ggplot(longley, aes(x = Year)) +
#     geom_point(aes(y = Unemployed)) +
#     geom_point(aes(y = Armed.Forces), color = "blue") +
#     geom_line(aes(y = Unemployed)) +
#     geom_line(aes(y = Armed.Forces), color = "blue")
```

The two measures *Unemployment* and *Armed.Forces* are two different measures we have per year and that we can plot together. The data is not reflecting this as something we can compute on. We can split the two measures into subplots insted of plotting them in the same frame. But a better solution is to reformat the data frame so we have one column telling us whether an observation is Unemployment or Armed.Forces and another using the values and then setting the color according to the first column and the y-axis according to the other. You can do this with the `gather()` function from `tidyr` package:

This chunk of code is for watching the result of the tidying before passing to the one line plotting with the tidying included:

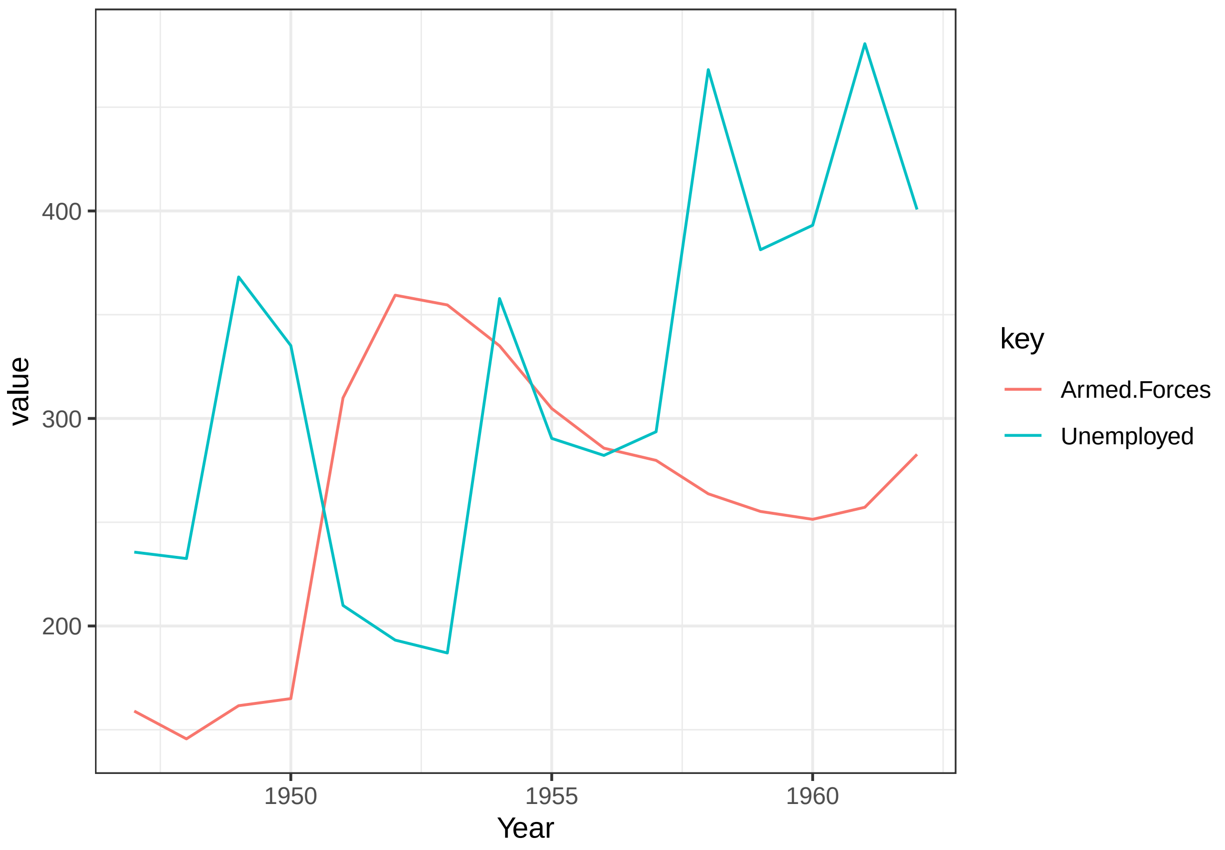
```
longley %>%
    gather(key = Status, value = Measure, Unemployed, Armed.Forces) %>%
    select(Status,Measure,Year) %>%
    head(10) %>% kable
```

Status	Measure	Year
Unemployed	235.6	1947
Unemployed	232.5	1948
Unemployed	368.2	1949
Unemployed	335.1	1950
Unemployed	209.9	1951

Status	Measure	Year
Unemployed	193.2	1952
Unemployed	187.0	1953
Unemployed	357.8	1954
Unemployed	290.4	1955
Unemployed	282.2	1956

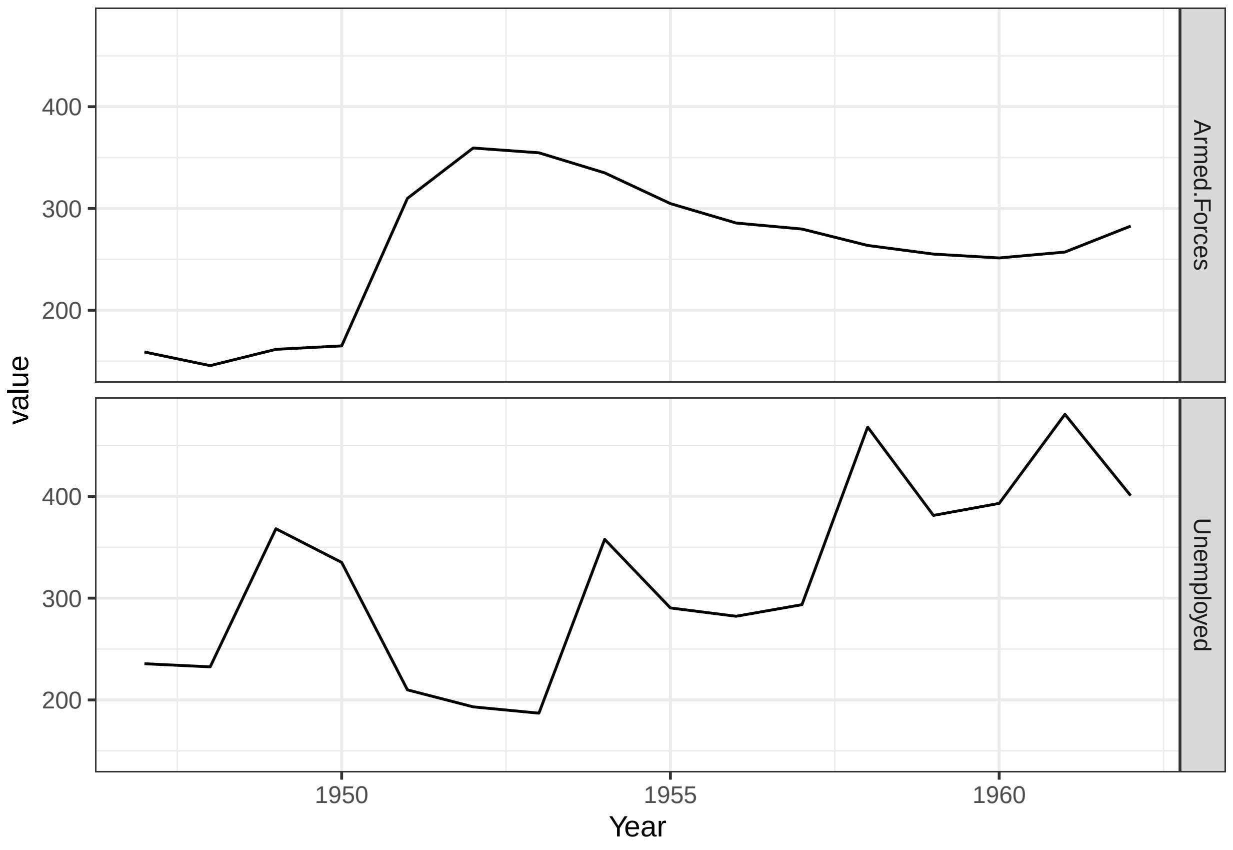
Tidying and plotting all together:

```
longley %>% gather(key,value, Unemployed, Armed.Forces) %>%
    ggplot(aes(x = Year, y = value, color = key)) + geom_line()
```



If you want the two values in different facets, we can specify this instead of setting colors:

```
# longley %>% gather(key, value, Unemployed, Armed.Forces) %>%
#     ggplot(aes(x = Year, y = value)) + geom_line() +
#     facet_grid(key ~ .)

longley %>% pivot_longer(c(Unemployed, Armed.Forces), names_to = "key", values_to = "value") %>%
    ggplot(aes(x = Year, y = value)) + geom_line() +
    facet_grid(key ~ .)
```



Facets

Facets are subplots showing different subset of the data. You can specify facects using one of two functions, `facet_grid()` creates facets from a fomrula `rows ~columns` adding `facet_wrap()` creates facets form the formula `~ variables`.

By deafult, `ggplot2` will try to put values on the same axes when you create facets using `facet_grid()`.

In the previous example, Armed.Forces values a re shown on the same x- and y-axis as Unemployment even though the y-values are not covering the same range. The parameter `scales` can be used to change this. Facets within a column will always have the same x-axis, however, and facects within a row will have the same y-axis.

```
#iris %>% gather(Measurement, Value, -Species) %>% head(10) %>% kable
# Instead
iris %>% pivot_longer(c(-Species), names_to = "Key", values_to = "Value") %>% head(10) %>% kable
```

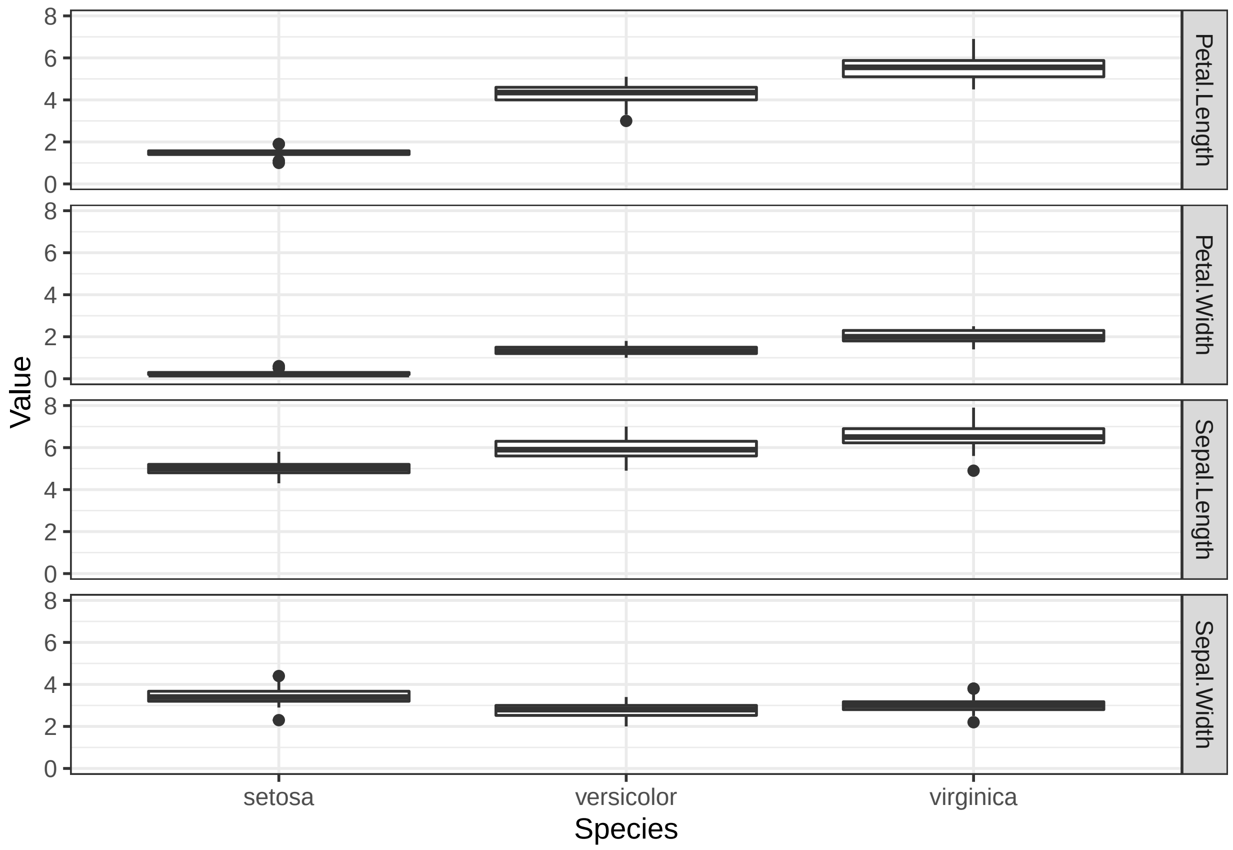
`pivot_longer()` is the new recommended function instead of `gather()`.

Species	Key	Value
setosa	Sepal.Length	5.1
setosa	Sepal.Width	3.5
setosa	Petal.Length	1.4
setosa	Petal.Width	0.2
setosa	Sepal.Length	4.9
setosa	Sepal.Width	3.0

Species	Key	Value
setosa	Petal.Length	1.4
setosa	Petal.Width	0.2
setosa	Sepal.Length	4.7
setosa	Sepal.Width	3.2

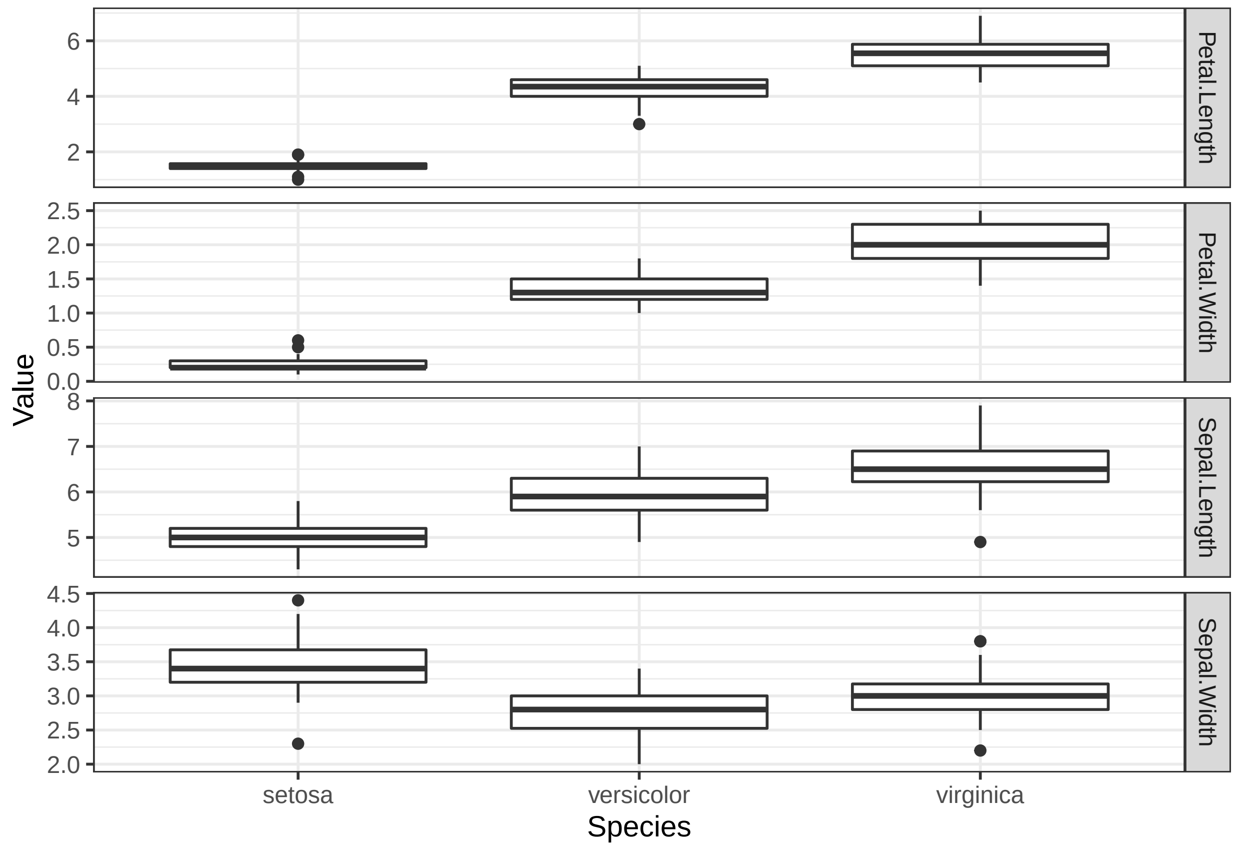
Plotting the four measurements for each species i diffent facets with the *iris* data:

```
iris %>% pivot_longer(c(-Species), names_to = "Measurement", values_to = "Value") %>%
    ggplot(aes(x = Species, y = Value)) +
    geom_boxplot() +
    facet_grid(Measurement ~ .)
```



Fixing the scale:

```
iris %>% pivot_longer(c(-Species), names_to = "Measurement",  values_to = "Value") %>%
    ggplot(aes(x = Species, y = Value)) +
    geom_boxplot() +
    facet_grid(Measurement ~ ., scale = "free_y")
```



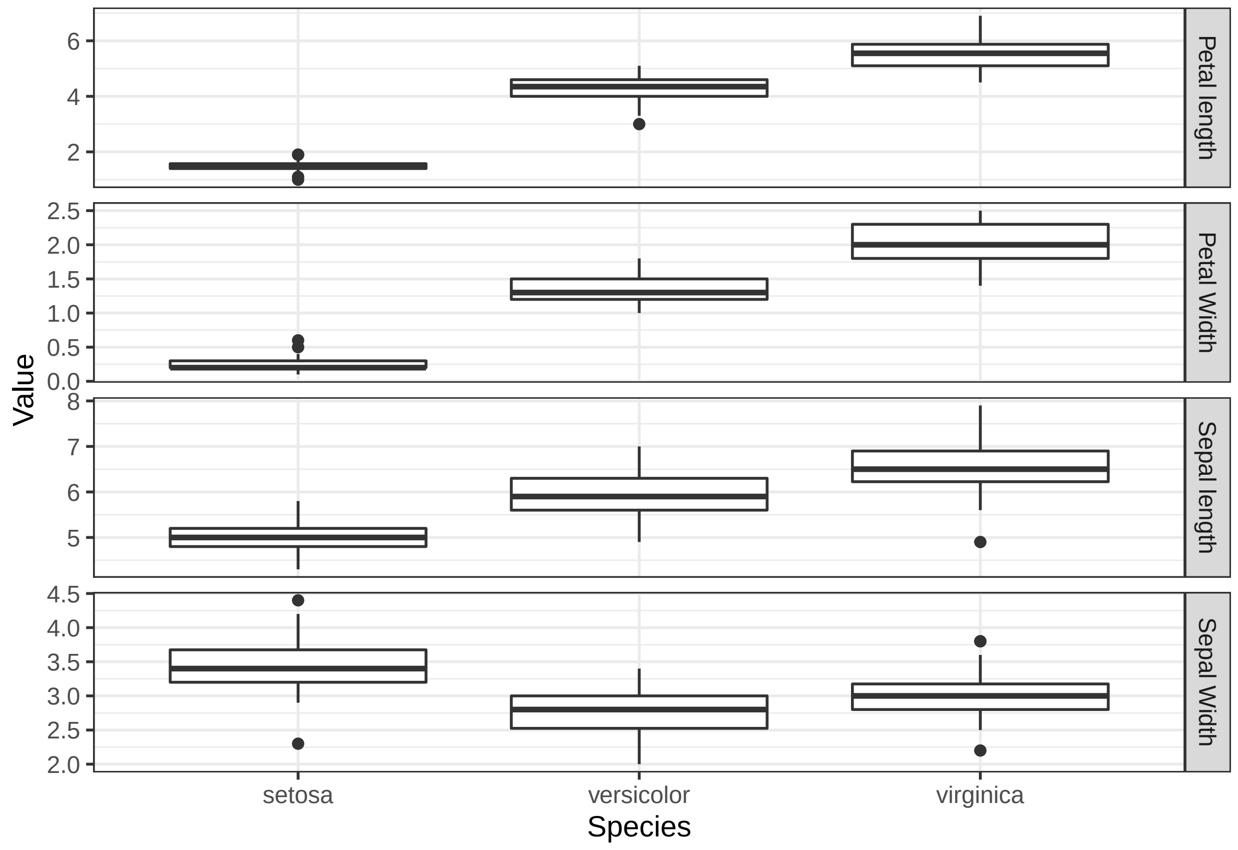
By default, all the facets will have the same size. You can modify this using `space` variable. This is mainy useful for categorical values if one facet has many more levels than another.

The labels used for facets are taken from the factors in the variables used to construct the facet.

To modify the labels you can use `labeller` paremeter to `facet_grid()`.

This paremeter takes a function (`labeller()`) as an argument that is responsible for constructing labels. You can give `labeller()`a named argument and specify a factor to make labels.

```
label_map <- c(Petal.Width = "Petal Width",
               Petal.Length = "Petal length",
               Sepal.Width = "Sepal Width",
               Sepal.Length = "Sepal length")

iris %>% pivot_longer(c(-Species), names_to = "Measurement", values_to = "Value") %>%
    ggplot(aes(x = Species, y = Value)) +
    geom_boxplot() +
    facet_grid(Measurement ~ ., scale = "free_y",
               labeller = labeller(Measurement = label_map))
```



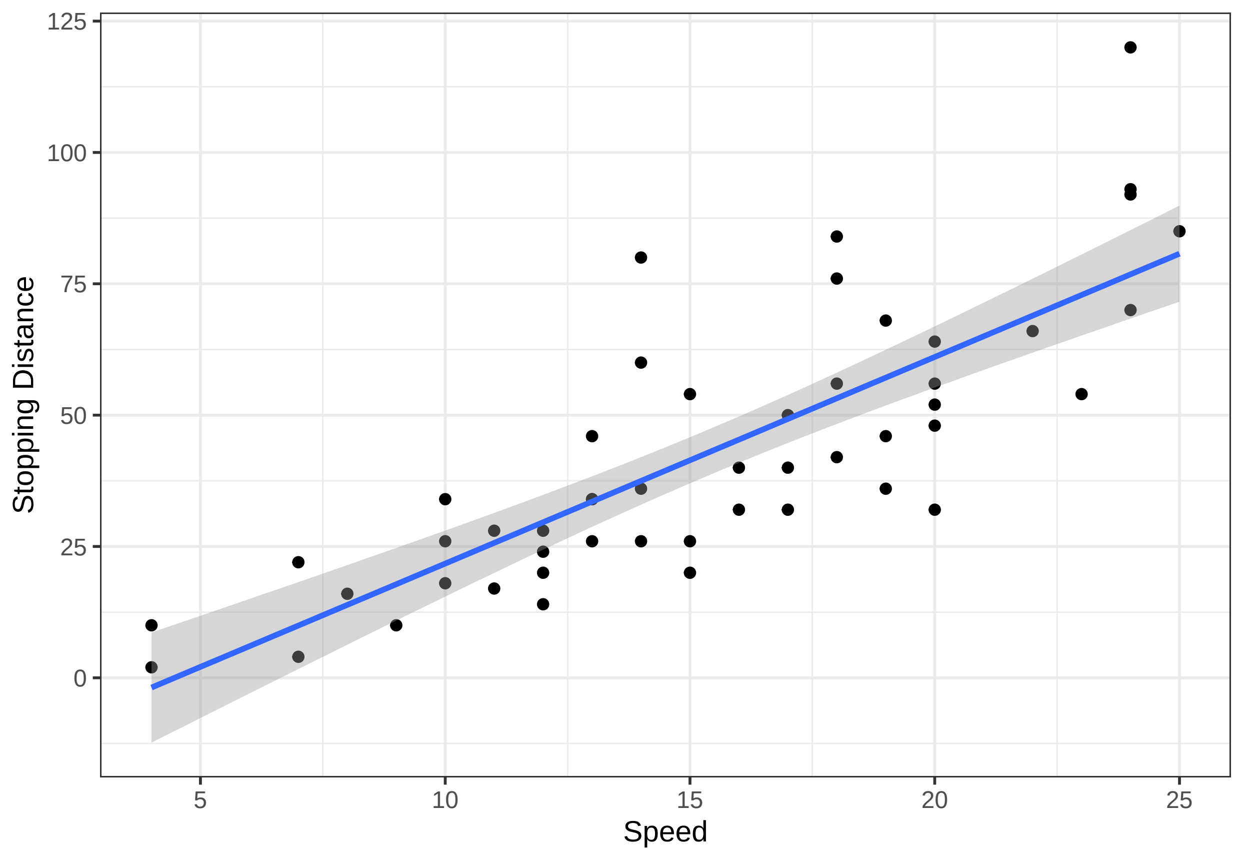
Scaling

The geometries tell `ggplot2` how you want your data mapped to visual components, like points or densities, and scales tell `ggplot2` gow dimensions should be visualized. The simplest scales are x- and y-axes, where values are mapped to positions, but scales also apply to colors.

The simplest wat to use `scales` is to put labels on the axes, you can also do this using the `xlab()` and `ylab()` functions, however in this example you see a different use of scales:

```
cars %>% ggplot(aes(x = speed, y = dist)) +
    geom_point() + geom_smooth(method = "lm") +
    scale_x_continuous("Speed") +
    scale_y_continuous("Stopping Distance")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

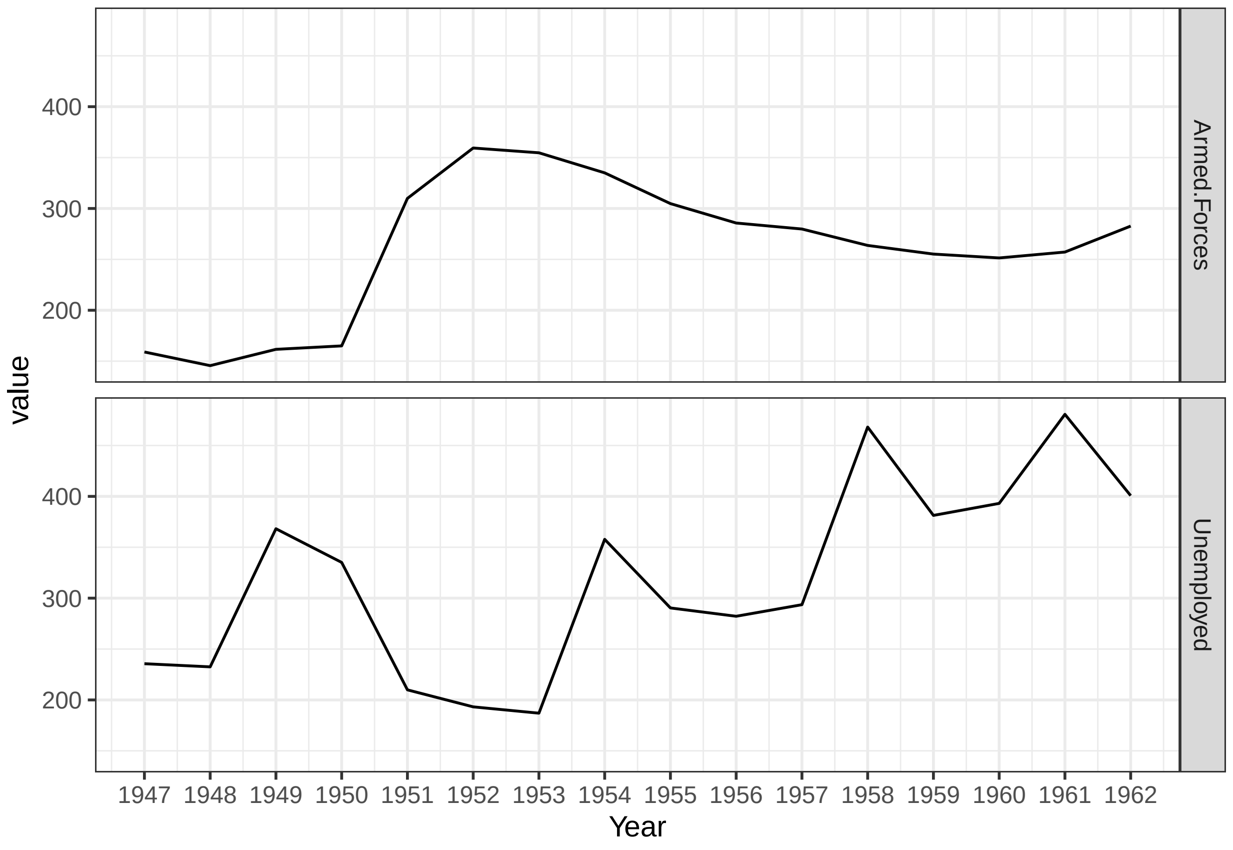


Both x- and y-axis are showing a continous value, o we scale like that and give the scale a name as the parameter.

In general, you can use `scale_x/y_continuous()` functions to control the axis graphics.

For example, to set the breakpoints, if you want to plot the *longley* data with a tickmark for every year instead of every five years:

```
longley %>% pivot_longer(c(Unemployed, Armed.Forces), names_to = "key", values_to = "value") %>%
    ggplot(aes(x = Year, y = value)) + geom_line() +
    scale_x_continuous(breaks = 1947:1962) +
    facet_grid(key ~ .)
```



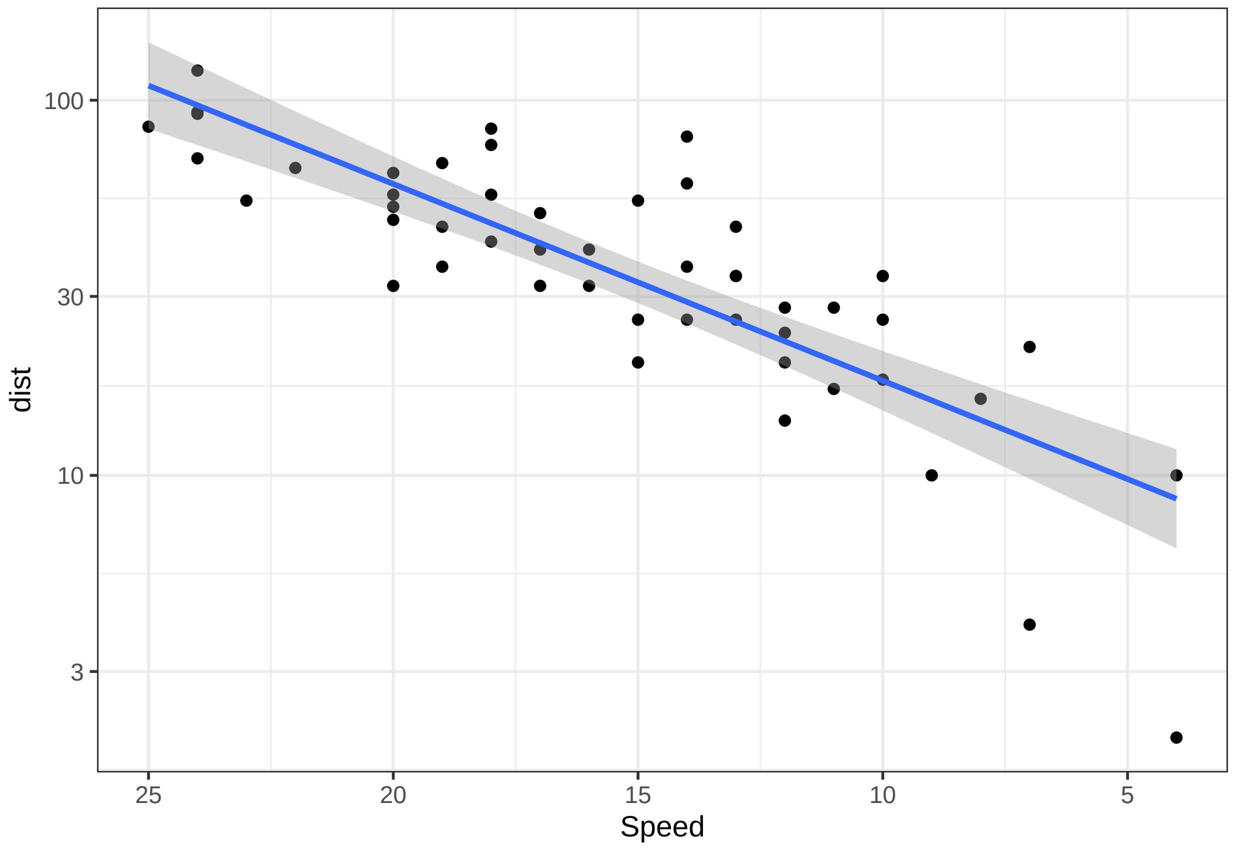
Scales are also a way to transform data shown on an axis. If you want to log-transform the x- or y-axis, you can use the `scale_x/y_log10()` functions, for instance. This usually leads to a nicer plot since the plotting code then knows that you want to show the data on a log scale rather than showing transformed data on a linear scale.

To reverse an axis use `scale_x/y_reverse()`. This is better than reversing the data mapped in the aesthetic.

```r
cars %>% ggplot(aes(x = speed, y = dist)) +
geom_point() + geom_smooth(method = "lm") +
scale_x_reverse("Speed") +
scale_y_continuous("Stoppping Distance") +
scale_y_log10()
```

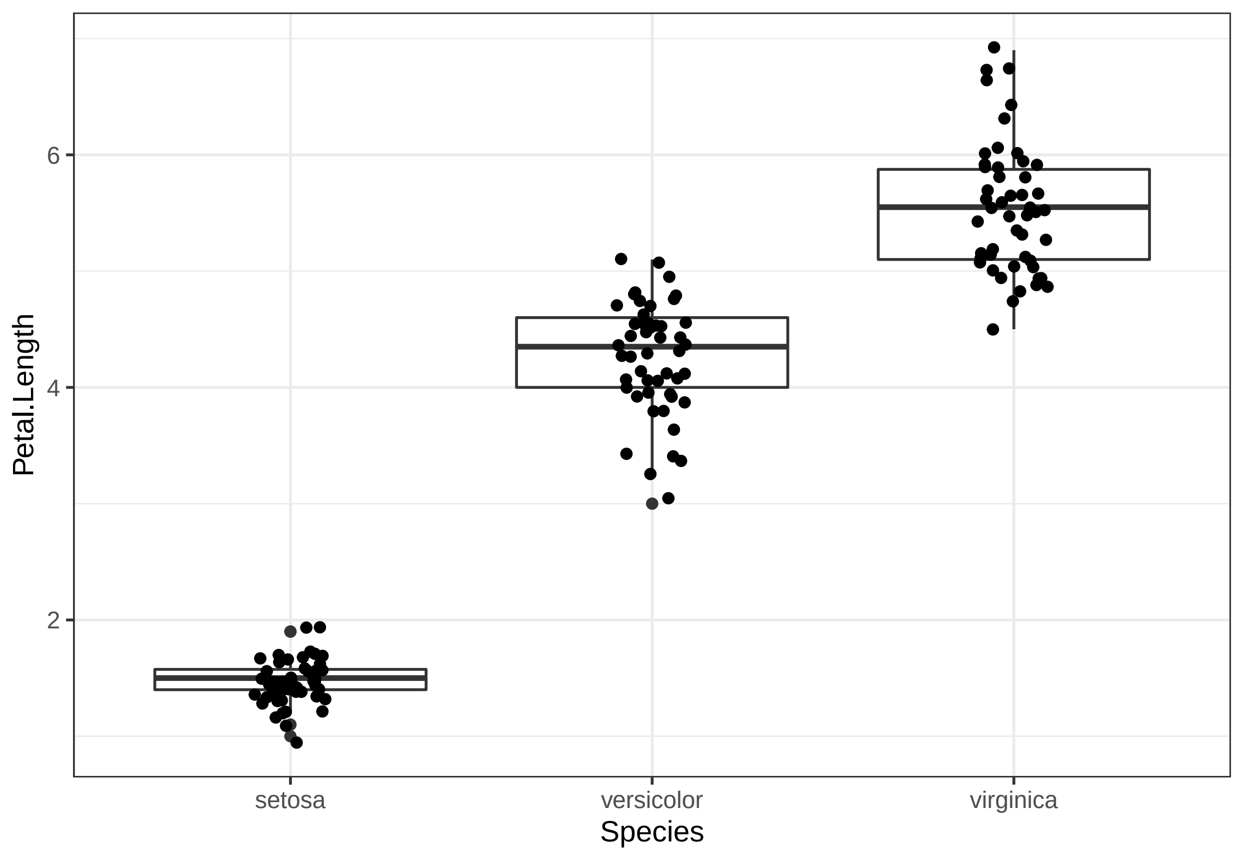
```
## Scale for 'y' is already present. Adding another
## scale for 'y', which will replace the existing
## scale.
```

```
## `geom_smooth()` using formula 'y ~ x'
```



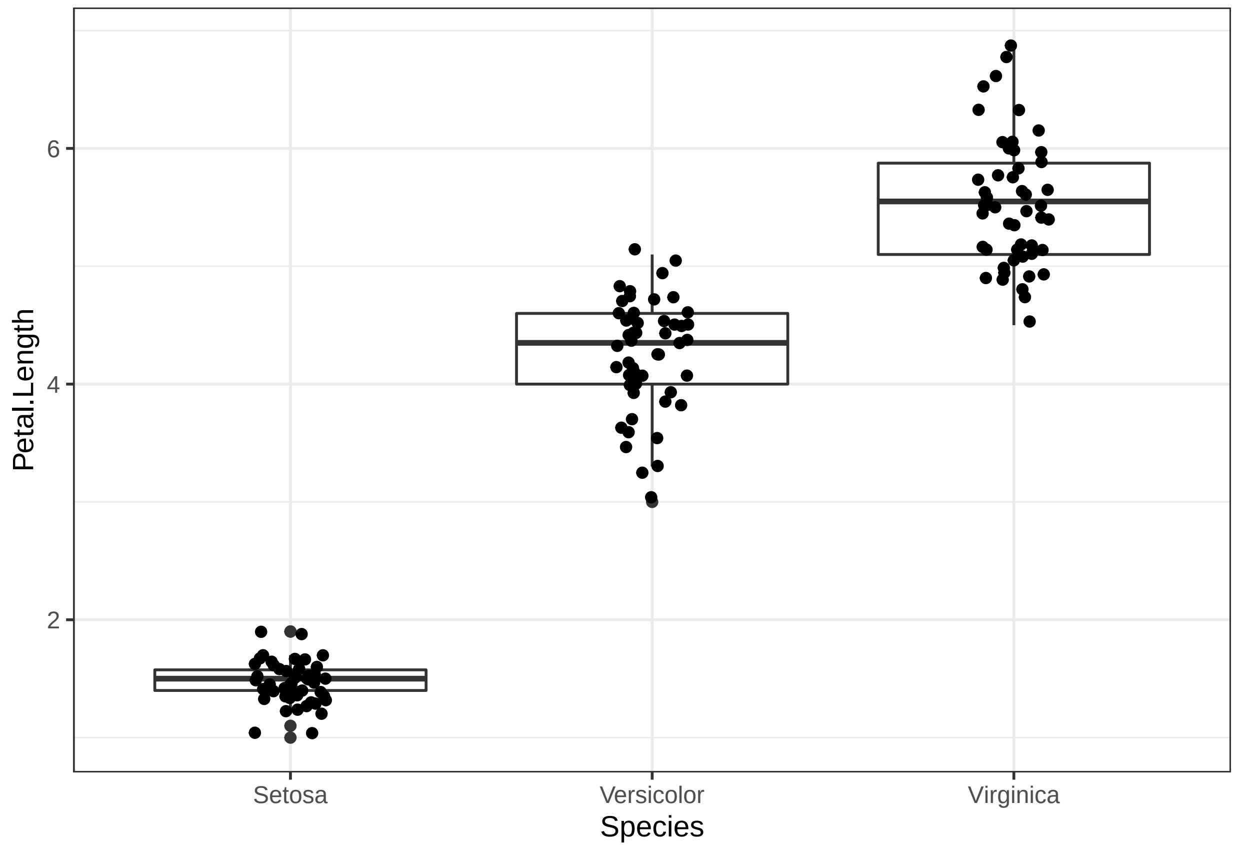
Neither axis has to be continuous. If you map a factor to x or y in the aesthetics, yoy get a discrete axis.

```
iris %>% ggplot(aes(x = Species, y = Petal.Length)) +
    geom_boxplot() + geom_jitter(width = 0.1, height = 0.1)
```



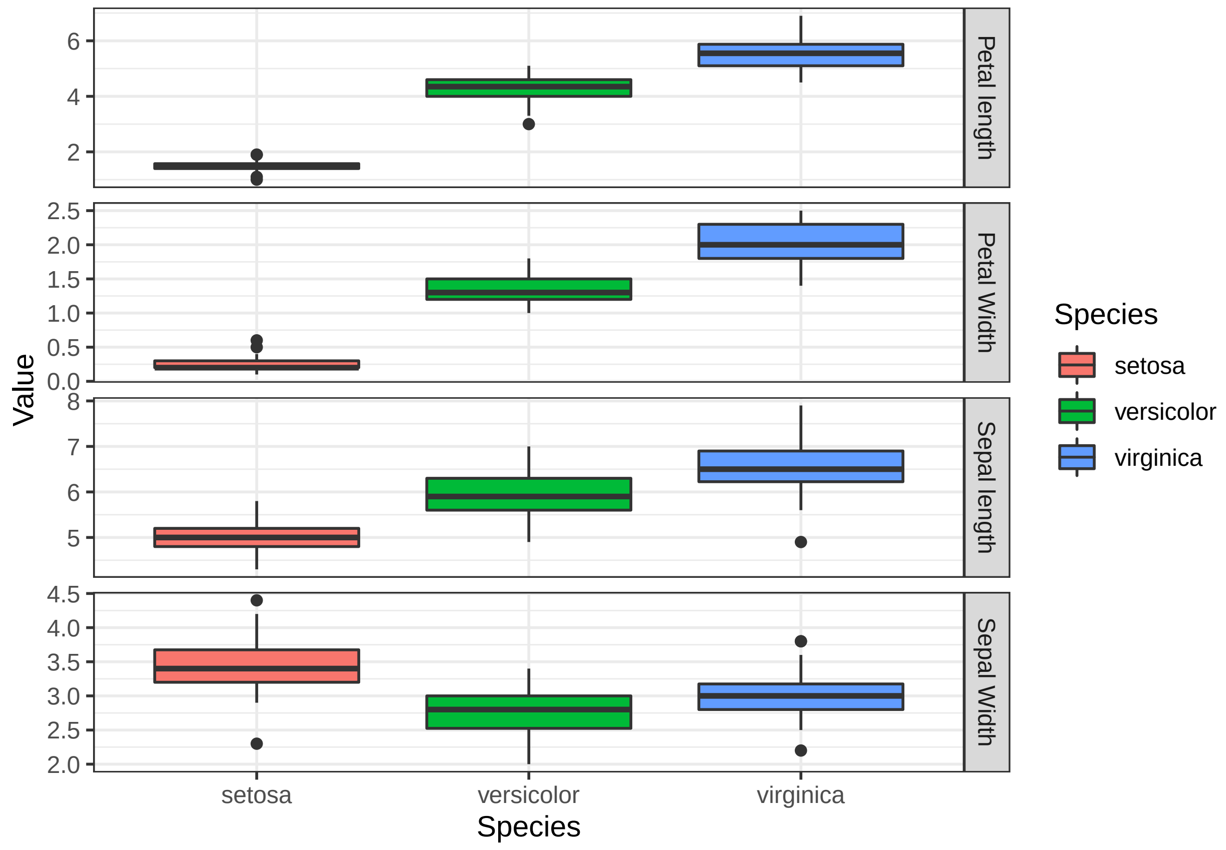
If you want to modify the x-axis, you need to use `scale_x_discrete()` instead of `scale_x_continuous()`.

```
iris %>% ggplot(aes(x = Species, y = Petal.Length)) +
    geom_boxplot() + geom_jitter(width = 0.1, height = 0.1) +
    scale_x_discrete(label = c("setosa" = "Setosa",
                               "versicolor" = "Versicolor",
                               "virginica" = "Virginica"))
```



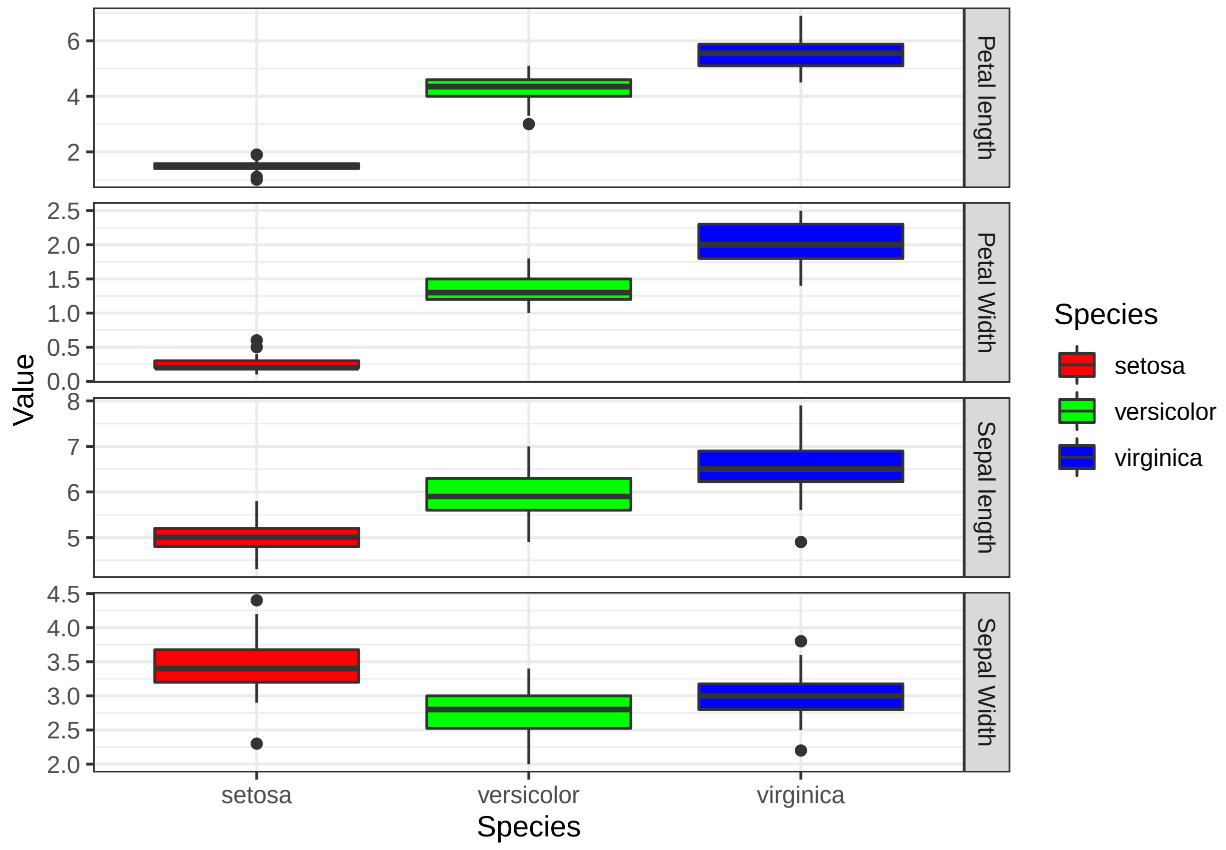
You can use various `scale_color_` functions to control the color of lines and points, and you use `scale_fill_` function to control the color fo filled areas.

```
iris %>% pivot_longer(c(-Species), names_to = "Measurement", values_to = "Value") %>%
    ggplot(aes(x = Species, y = Value, fill = Species)) +
    geom_boxplot() +
    facet_grid(Measurement ~ ., scale = "free_y",
               labeller = labeller(Measurement = label_map))
```



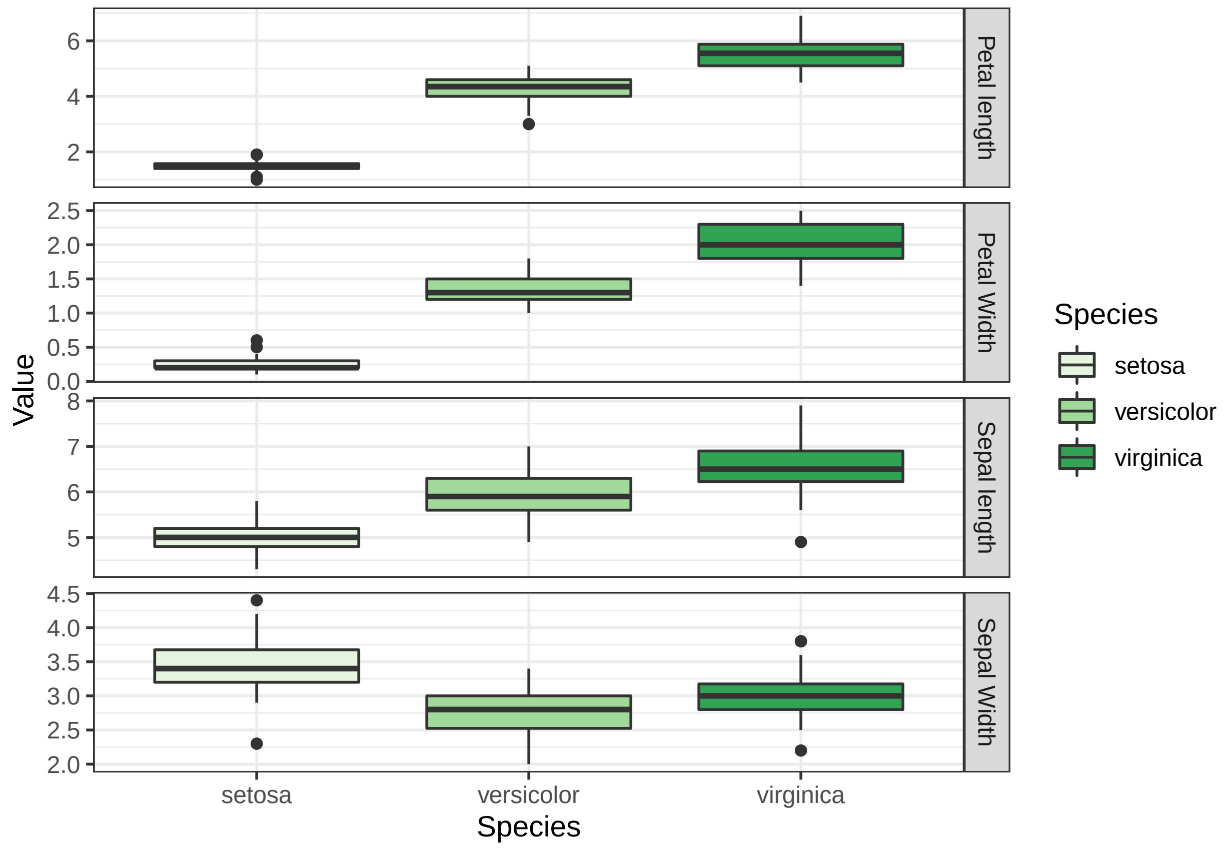
There are two classes, as there are for axes-discrete and continuous. The Species variable in *iris* is discrete. To modify the color you need one of the functions. The simplest is just to give a color per species explicitly. You can do that with the `scale_fill_manual()` function:

```
iris %>% pivot_longer(c(-Species), names_to = "Measurement", values_to = "Value") %>%
    ggplot(aes(x = Species, y = Value, fill = Species)) +
    geom_boxplot() +
    scale_fill_manual(values = c("red", "green", "blue")) +
    facet_grid(Measurement ~ ., scale = "free_y",
               labeller = labeller(Measurement = label_map))
```



Unless you have a good feeling for how colors work together and which combinations can be problematic for color-blind, it is better to use one of the "brewer" choices. The are methods for constructing good combinations of colors, and you can use them with the `scale_fill_brewer()` function:

```
iris %>% pivot_longer(c(-Species), names_to = "Measurement", values_to = "Value") %>%
    ggplot(aes(x = Species, y = Value, fill = Species)) +
    geom_boxplot() +
    scale_fill_brewer(palette = "Greens") +
    facet_grid(Measurement ~ ., scale = "free_y",
               labeller = labeller(Measurement = label_map))
```



Themes and ther Graphics Transformations

To set a default theme:

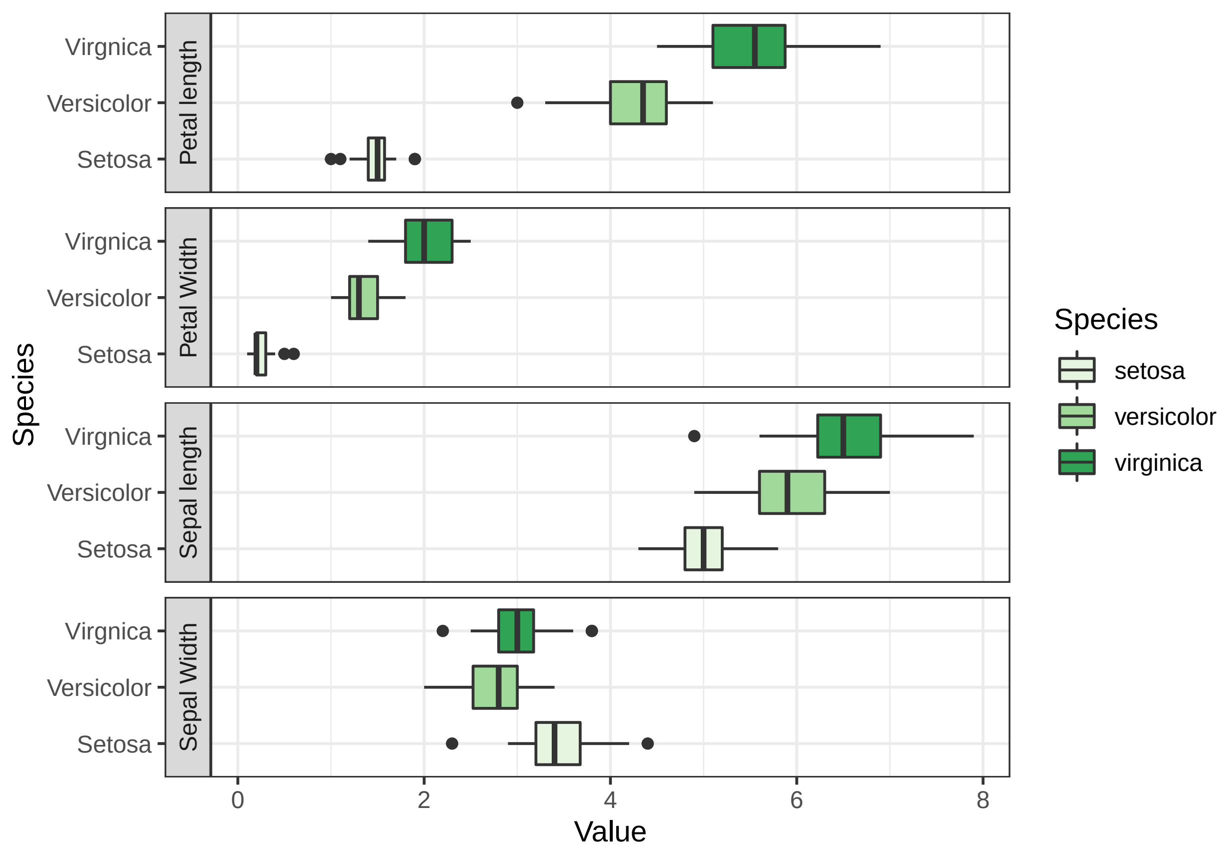
```
theme_set(theme_bw())
```

Look for functions that start with `theme_`, you can add them to the plot using + as you would any other `ggplot2` modification.

You can change coordinate systems using various `coord_` functions, the simplest is just flipping x and y with the `coord_flip()`.

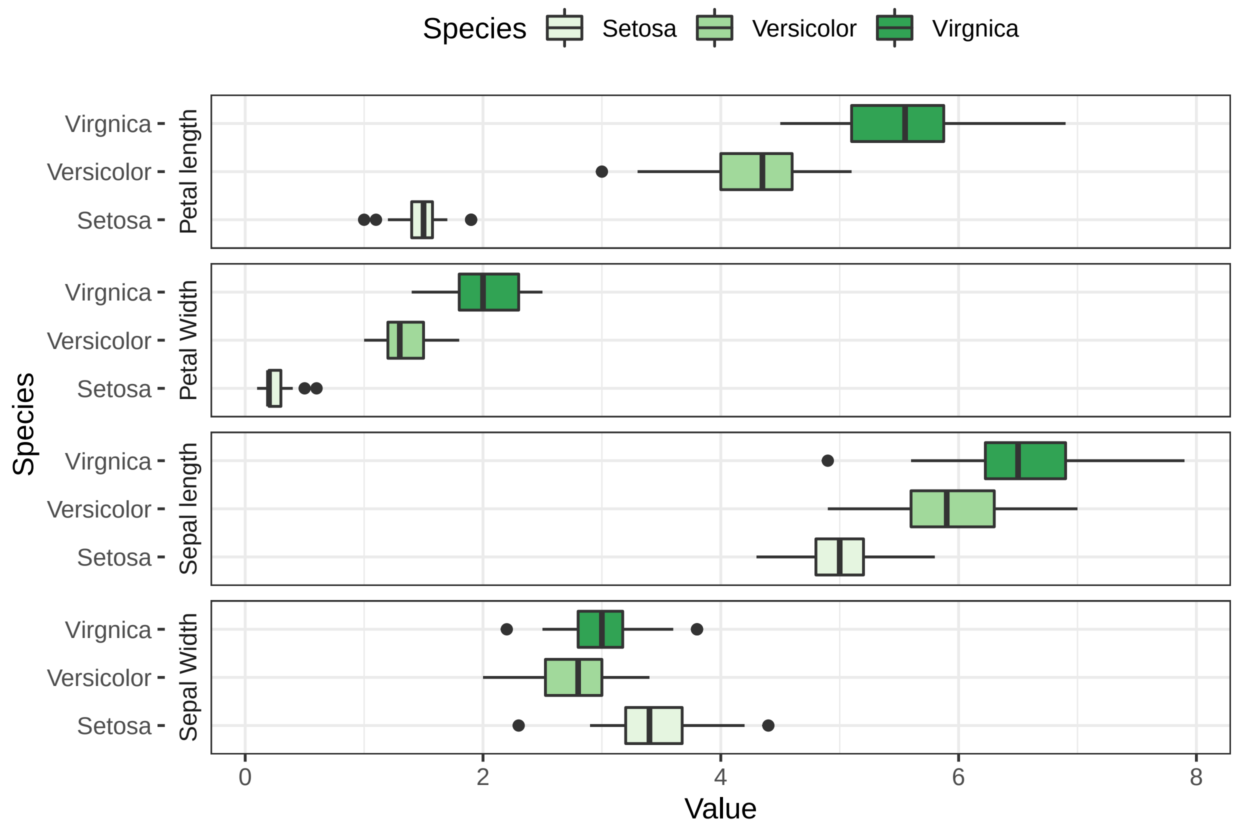
You can control the placement of facet labels using the switch option to face_grid(). Giving the `swtich` parameter the value y will switch the location of that label.

```
iris %>% pivot_longer(c(-Species), names_to = "Measurement", values_to = "Value") %>%
    ggplot(aes(x = Species, y = Value, fill = Species)) +
    geom_boxplot() +
    scale_x_discrete(labels = c("setosa" = "Setosa",
                                "versicolor" = "Versicolor",
                                "virginica" = "Virgnica")) +
    scale_fill_brewer(palette ="Greens") +
    facet_grid(Measurement ~ ., switch = "y",
               labeller = labeller(Measurement = label_map)) +
    coord_flip()
```



To change the labels color background you can use `theme(strip.background = element_blank())`. We can also move the legend label using a theme modification such as `theme(legend.position = "top")`.

```
label_map <- c(Petal.Width = "Petal Width",
               Petal.Length = "Petal length",
               Sepal.Width = "Sepal Width",
               Sepal.Length = "Sepal length")

species_map <- c("setosa" = "Setosa",
                 "versicolor" = "Versicolor",
                 "virginica" = "Virgnica")

iris %>% pivot_longer(c(-Species), names_to = "Measurement", values_to = "Value") %>%
    ggplot(aes(x = Species, y = Value, fill = Species)) +
    geom_boxplot() +
    scale_x_discrete(labels = species_map) +
    scale_fill_brewer(palette ="Greens", labels = species_map) +
    facet_grid(Measurement ~ ., switch = "y",
               labeller = labeller(Measurement = label_map)) +
    coord_flip() +
    theme(strip.background = element_blank()) +
    theme(legend.position = "top")
```



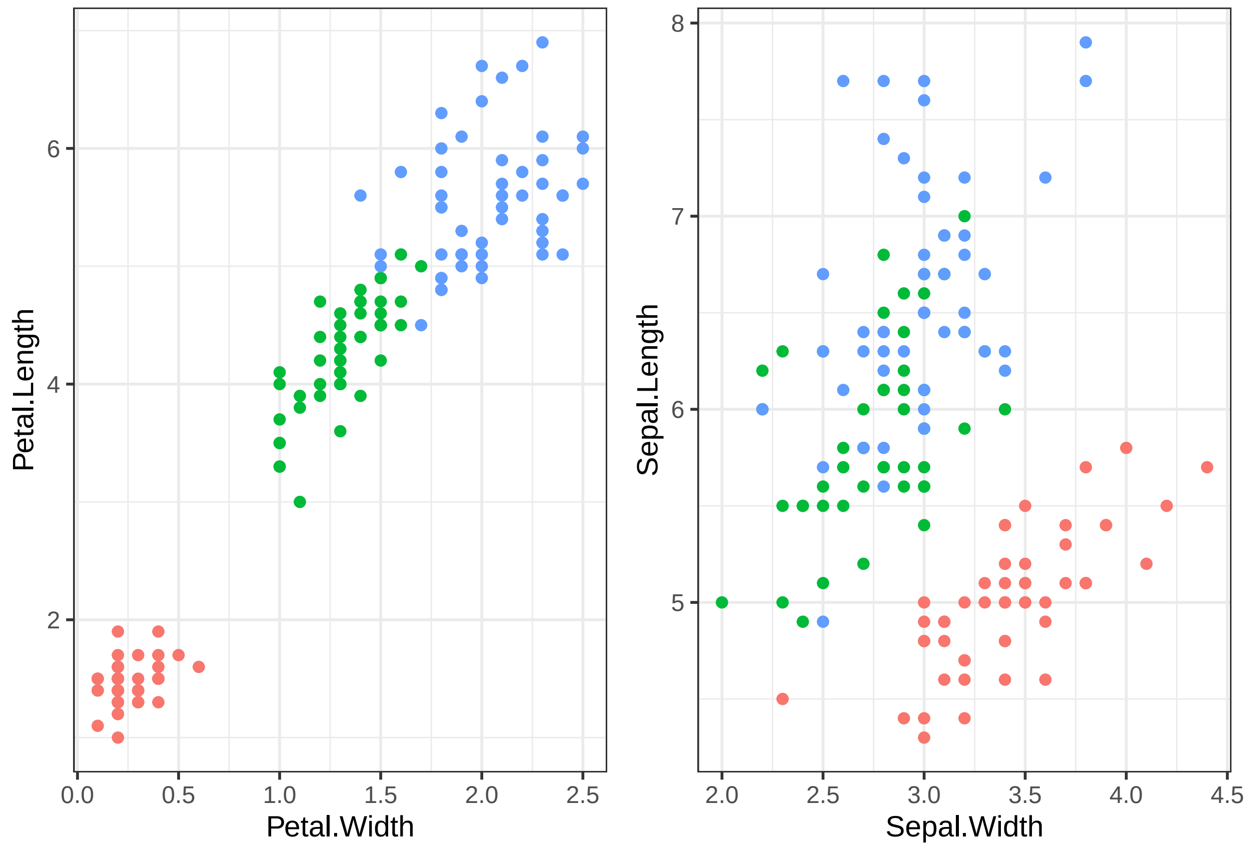
Figures with multiple plots

You use facets when you want to plot different subsets of the data. But esentially is the same plot.

The `ggplot2` package doesn't support combining multiple plots, but in cam be achieved using the underlying graphics system, `grid` high-level functions.

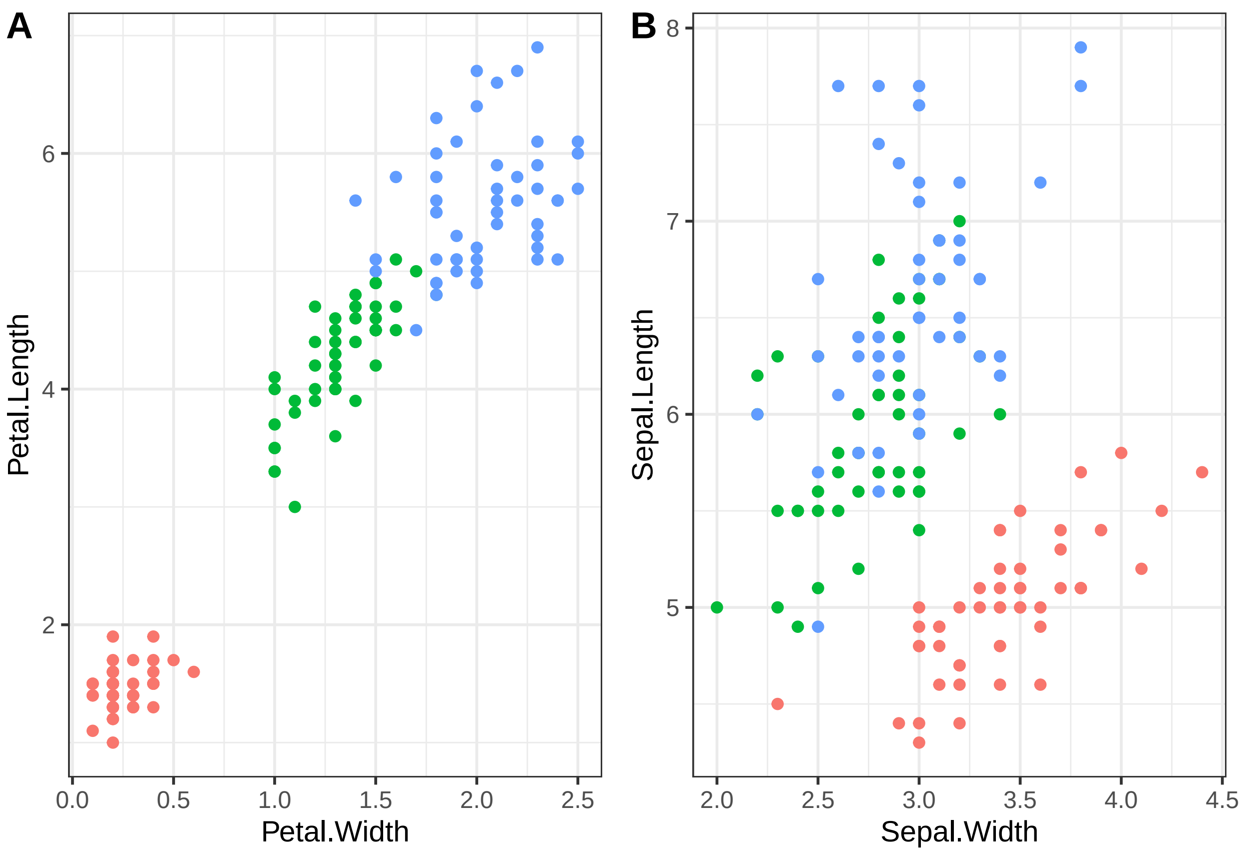
You can get this from the `gridExtra` package.

```
petal <-  iris %>% ggplot() +
    geom_point(aes(x = Petal.Width, y = Petal.Length, color = Species)) +
    theme(legend.position = "none")

sepal <- iris %>% ggplot() +
    geom_point(aes(x = Sepal.Width, y = Sepal.Length, color  = Species)) +
    theme(legend.position = "none")

library(gridExtra)

grid.arrange(petal, sepal, ncol = 2)
```



Another aproach is to use `plot_grid()` function from cowplot package. This package redefines the default `ggplot2` package. *You can use `theme_set()` function to change it back if you don't like the theme that cowplot provides.

```
library(cowplot)

plot_grid(petal, sepal, labels = c("A", "B"))
```


A
B
Petal.Length
Petal.Width
0.0
0.5
1.0
1.5
2.0
2.5
2
4
6
Sepal.Length
Sepal.Width
2.0
2.5
3.0
3.5
4.0
4.5
5
6
7
8