Statistical Computing with R

Lecture 12: the `dplyr` package; data visualization with `ggplot2`

Mirko Signorelli
: mirkosignorelli.github.io
: statcompr [at] gmail.com

Mathematical Institute
Leiden University

Master in Statistics and Data Science (2023-2024)

Universiteit Leiden

Recap

Lecture 11:

- mixture models
- the EM algorithm
- dealing with repetitive and time-consuming computations
- `replicate( )`
- EM algorithm + `replicate`
- parallelization and the `foreach` package (*self-study*)

Today:

- the `dplyr` package
- data visualization with `ggplot2`

Before we start

- Feedback on assignment 2 published. Do you find this type of formative assignments with personal feedback useful?
- Do you have questions about last week's class (EM algorithm)?

The `dplyr` package

Data visualization with `ggplot2`

The `dplyr` package

The `dplyr` package:

- contains many useful functions for data manipulation
- popular and widely used

Not enough time to cover all `dplyr` functions ⇒ I will showcase some of the most common ones (but there are many more!)

dplyr versus base R

- Many base R functions / operators have dplyr equivalents
  1. the results are often similar / identical
  2. but sometimes they differ substantially
- A few examples:

dplyr function	Base R alter ego
`filter(df, condition)`	`subset(df, condition)`
`arrange(df, var1, var2)`	`df[order(df$var1, df$var2), ]`
`select(df, var1, var2)`	`subset(df, T, c('var1', 'var2'))`
`mutate(df, var1+var2)`	`df$var1 + df$var2`
`count(df, var1, var2)`	`table(df$var1, df$var2)`

filter( )

- `filter( )` can be used to select rows in a data frame:

```
library(dplyr)
df1 = filter(iris, Species == 'setosa') # dplyr
df2 = subset(iris, Species == 'setosa') # base R
identical(df1, df2)
```

```
## [1] TRUE
```

arrange( )

- arrange( ) allows to order rows in a data frame:
  1. possible to specify > 1 ordering variable
  2. use desc( ) for descending order

```
df3 = arrange(iris, Sepal.Length)
head(df3, 3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          4.3         3.0          1.1         0.1
## 2          4.4         2.9          1.4         0.2
## 3          4.4         3.0          1.3         0.2
##   Species
## 1  setosa
## 2  setosa
## 3  setosa
```

```
df4 = arrange(iris, desc(Species), Sepal.Length)
head(df4, 3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          4.9         2.5          4.5         1.7
## 2          5.6         2.8          4.9         2.0
## 3          5.7         2.5          5.0         2.0
##     Species
## 1 virginica
## 2 virginica
```

select( )

- `select( )` subsets variables in a data frame:

```
df5 = select(iris, Sepal.Length, Species) #dplyr
head(df5, 3)
```

```
##   Sepal.Length Species
## 1          5.1  setosa
## 2          4.9  setosa
## 3          4.7  setosa
```

```
df6 = iris[ , c('Sepal.Length', 'Species')] # base R v1
df7 = subset(iris, T, c('Sepal.Length', 'Species')) # base R v2
identical(df5, df6)
```

```
## [1] TRUE
```

```
identical(df5, df7)
```

```
## [1] TRUE
```

mutate( )

- mutate( ) computes new variables from existing ones:

```
library(brolgar) # for the heights and wages dfs
heights = mutate(heights, height_m = height_cm / 100)
head(heights, 2)
```

```
## # A tsibble: 2 x 5 [!]
## # Key:       country [1]
##   country     continent  year height_cm height_m
##   <chr>       <chr>     <dbl>     <dbl>    <dbl>
## 1 Afghanistan Asia       1870      168.     1.68
## 2 Afghanistan Asia       1880      166.     1.66
```

```
wages = mutate(wages, wage = exp(ln_wages)) |>
  select(id, ln_wages, xp, wage)
head(wages, 2)
```

```
## # A tsibble: 2 x 4 [!]
## # Key:       id [1]
##      id ln_wages    xp  wage
##   <int>    <dbl> <dbl> <dbl>
## 1    31     1.49 0.015  4.44
## 2    31     1.43 0.715  4.19
```

Frequency tables: `table( )`

- Base `R` relies on `table( )` to compute frequency distributions
- Inputs can be vectors, or variables in data frames:

```
v = sample(1:5, size = 1000, replace = T,
           prob = c(0.1, 0.2, 0.3, 0.3, 0.1))
table(v) # input is a vector
```

```
## v
##   1   2   3   4   5
##  88 193 313 309  97
```

```
table(iris$Species) # input is a variable in a df
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

Frequency tables: `count( )`

- ⚠ `count( )` function from `dplyr` only accepts data frames as inputs ⇒ if you have a vector, you first need to store that vector in a data frame! ⚠

```
count(iris, Species)
```

```
##      Species  n
## 1     setosa 50
## 2 versicolor 50
## 3  virginica 50
```

- `count( )` outputs a data frame, whereas `table( )` returns an object of type `table`:

```
is(table(iris$Species))
```

```
## [1] "table"    "oldClass"
```

```
is(count(iris, Species))
```

```
## [1] "data.frame" "list"       "oldClass"   "vector"
```

Frequency tables: multiple variables

```r
set.seed(12)
sex = sample(c('M', 'F'), 100, replace = T)
vaccinated = sample(c('yes', 'no'), 100, replace = T,
                    prob = c(0.9, 0.1))
table(sex, vaccinated)
```

```
##    vaccinated
## sex no yes
##   F  3  55
##   M  3  39
```

```r
df1 = data.frame(sex, vaccinated)
count(df1, sex, vaccinated)
```

```
##   sex vaccinated  n
## 1   F         no  3
## 2   F        yes 55
## 3   M         no  3
## 4   M        yes 39
```

Recoding a variable with `case_when( )`

- Sometimes, you may need to recode a variable
- Example: BMI ("body-mass index")

$$BMI = \frac{W}{H^2}, \text{ where W = weight in kg, H = height in m}$$

$$\text{BMI class} = \begin{cases} \text{underweight} & \text{if } BMI < 18.5 \\ \text{normal weight} & \text{if } 18.5 \leq BMI < 25 \\ \text{overweight} & \text{if } 25 \leq BMI < 30 \\ \text{obese} & \text{if } BMI \geq 30 \end{cases}$$

- Handy `dplyr` function to do this: `case_when( )`

Recoding a variable with `case_when( )` (cont'd)

```r
df_bmi
```

```
##   id age height_m weight_kg
## 1  1  24     1.87        61
## 2  2  54     1.78        82
## 3  3  32     1.65        65
## 4  4  45     1.76        70
## 5  6  37     1.72        95
```

```r
# step 1: compute the BMI
df_bmi = mutate(df_bmi, bmi = weight_kg / (height_m^2))
df_bmi
```

```
##   id age height_m weight_kg      bmi
## 1  1  24     1.87        61 17.44402
## 2  2  54     1.78        82 25.88057
## 3  3  32     1.65        65 23.87511
## 4  4  45     1.76        70 22.59814
## 5  6  37     1.72        95 32.11195
```

Recoding a variable with case_when( ) (cont'd)

```r
# step 2: create the categories with case_when( )
df_bmi$class = case_when(
  df_bmi$bmi < 18.5 ~ 'underweight',
  df_bmi$bmi >= 18.5 & df_bmi$bmi < 25 ~ 'normal weight',
  df_bmi$bmi >= 25 & df_bmi$bmi < 30 ~ 'overweight',
  df_bmi$bmi >= 30 ~ 'obese'
)
# check results:
df_bmi
```

```
##   id age height_m weight_kg      bmi          class
## 1  1  24     1.87        61 17.44402   underweight
## 2  2  54     1.78        82 25.88057    overweight
## 3  3  32     1.65        65 23.87511 normal weight
## 4  4  45     1.76        70 22.59814 normal weight
## 5  6  37     1.72        95 32.11195         obese
```

summarize( ) and group_by( )

- summarize( ) computes functions of existing variables:

```
summarize(iris, mean = mean(Sepal.Length),
          median = median(Sepal.Length))
```

```
##       mean median
## 1 5.843333    5.8
```

- Summary statistics by group can be computed by first declaring groups through group_by( ):

```
iris |> group_by(Species) |>
  summarise(mean = mean(Sepal.Length),
            median = median(Sepal.Length))
```

```
## # A tibble: 3 x 3
##   Species     mean median
##   <fct>      <dbl>  <dbl>
## 1 setosa      5.01    5
## 2 versicolor  5.94    5.9
## 3 virginica   6.59    6.5
```

Base R equivalent of summarize( ) + group_by( )

- dplyr's summarize( ) + group_by( ):

```
iris |> group_by(Species) |>
  summarise(average = mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    average
##   <fct>        <dbl>
## 1 setosa        5.01
## 2 versicolor    5.94
## 3 virginica     6.59
```

- Base R equivalent:

```
aggregate(Sepal.Length ~ Species, FUN = mean, data = iris)
```

```
##      Species Sepal.Length
## 1     setosa        5.006
## 2 versicolor        5.936
## 3  virginica        6.588
```

Your turn

Exercises

The `airquality` dataset (part of base `R`) contains measurements of air quality in New York gathered in 1973. Check out `?airquality` for more details. **Use functions from the `dplyr` package** to:

1. select only observations from June to August;
2. convert the `Temp` variable from Fahrenheit to Celsius;
3. compute the mean temperature by month, both in Fahrenheit to Celsius.

Then, **solve the same exercise using only base `R` functions**.

⚠ To solve this exercise, you need to know that

$$T_C = 5\frac{T_F - 32}{9},$$

where T_C and T_F respectively denote temperatures expressed in Celsius and Fahrenheit degrees. ⚠

Solution using dplyr functions

```r
# Question 1
df = filter(airquality, Month >=6 & Month <=8)
# Question 2
df = mutate(df, Temp_Celsius = 5*(Temp - 32) / 9)
# Question 3
df |>
  group_by(Month) |>
  summarize(mean_Fahr = mean(Temp),
            mean_Cels = mean(Temp_Celsius))
```

```
## # A tibble: 3 x 3
##   Month mean_Fahr mean_Cels
##   <int>     <dbl>     <dbl>
## 1     6      79.1      26.2
## 2     7      83.9      28.8
## 3     8      84.0      28.9
```

Solution using base R functions

```r
# Question 1
df = subset(airquality, Month >=6 & Month <=8)
# Question 2
df$Temp_Celsius = 5*(df$Temp - 32) / 9
# Question 3
aggregate(cbind(Temp, Temp_Celsius) ~ Month, df, mean)
```

```
##   Month     Temp Temp_Celsius
## 1     6 79.10000     26.16667
## 2     7 83.90323     28.83513
## 3     8 83.96774     28.87097
```

More about `dplyr`

1. CRAN manual and vignettes
2. dedicated `dplyr`'s website
3. `dplyr`'s "cheatsheet"

Data visualization with ggplot2

The ggplot2 package

- The ggplot2 package is a widely used R package for data visualization

A few PROs:

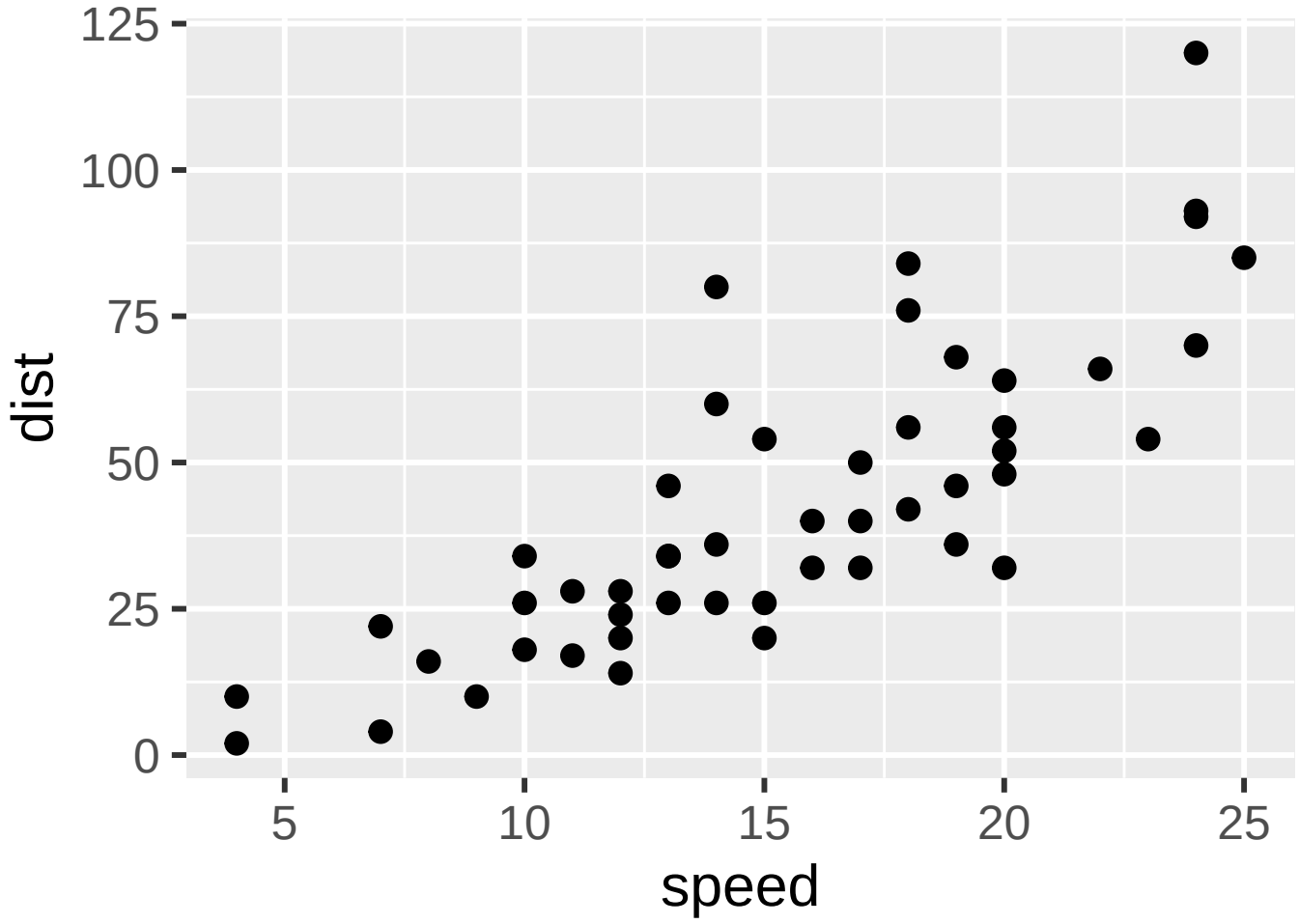
- very flexible; lots of options to customize your charts
- legend often added automatically ☺☺☺
- makes it possible to create complex and beautiful charts

Some CONs:

- syntax is not very R-like
- documentation often cryptic
- creating charts with ggplot2 often time-consuming

Example 1: a simple scatter plot

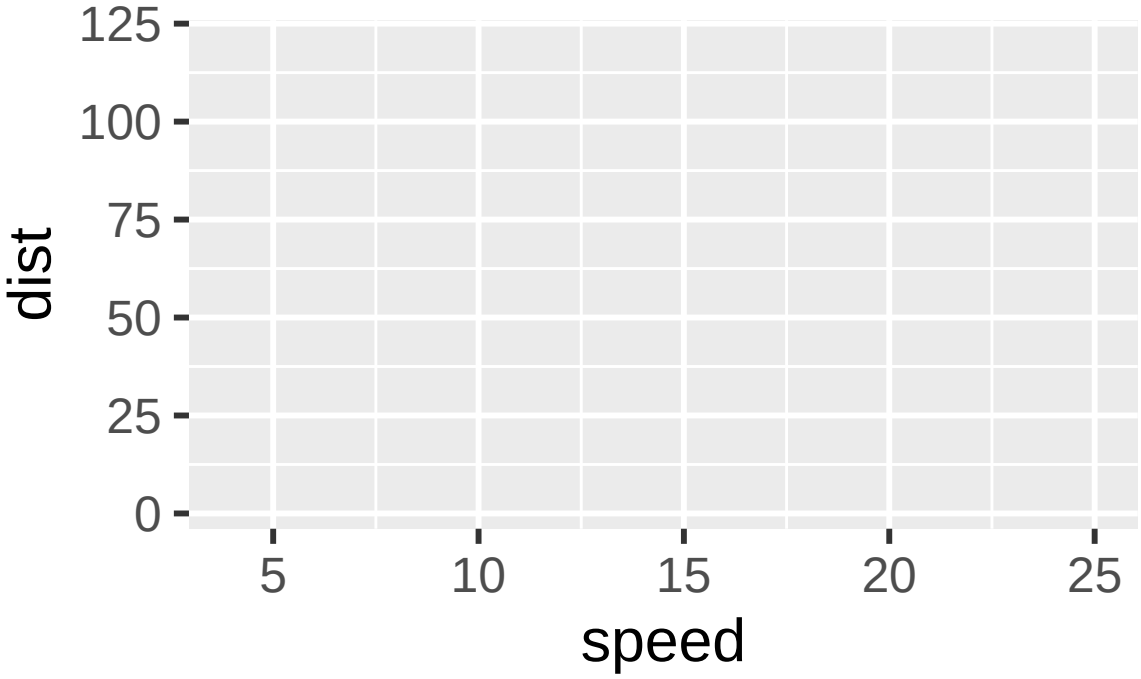
```
library(ggplot2)
ggplot(cars, aes(x = speed, y = dist)) +
  geom_point()
```



Understanding the code

1. `ggplot(cars, aes(x = speed, y = dist))` creates a Cartesian plane with `cars$speed` on the x axis, and `cars$dist` on the y axis:

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

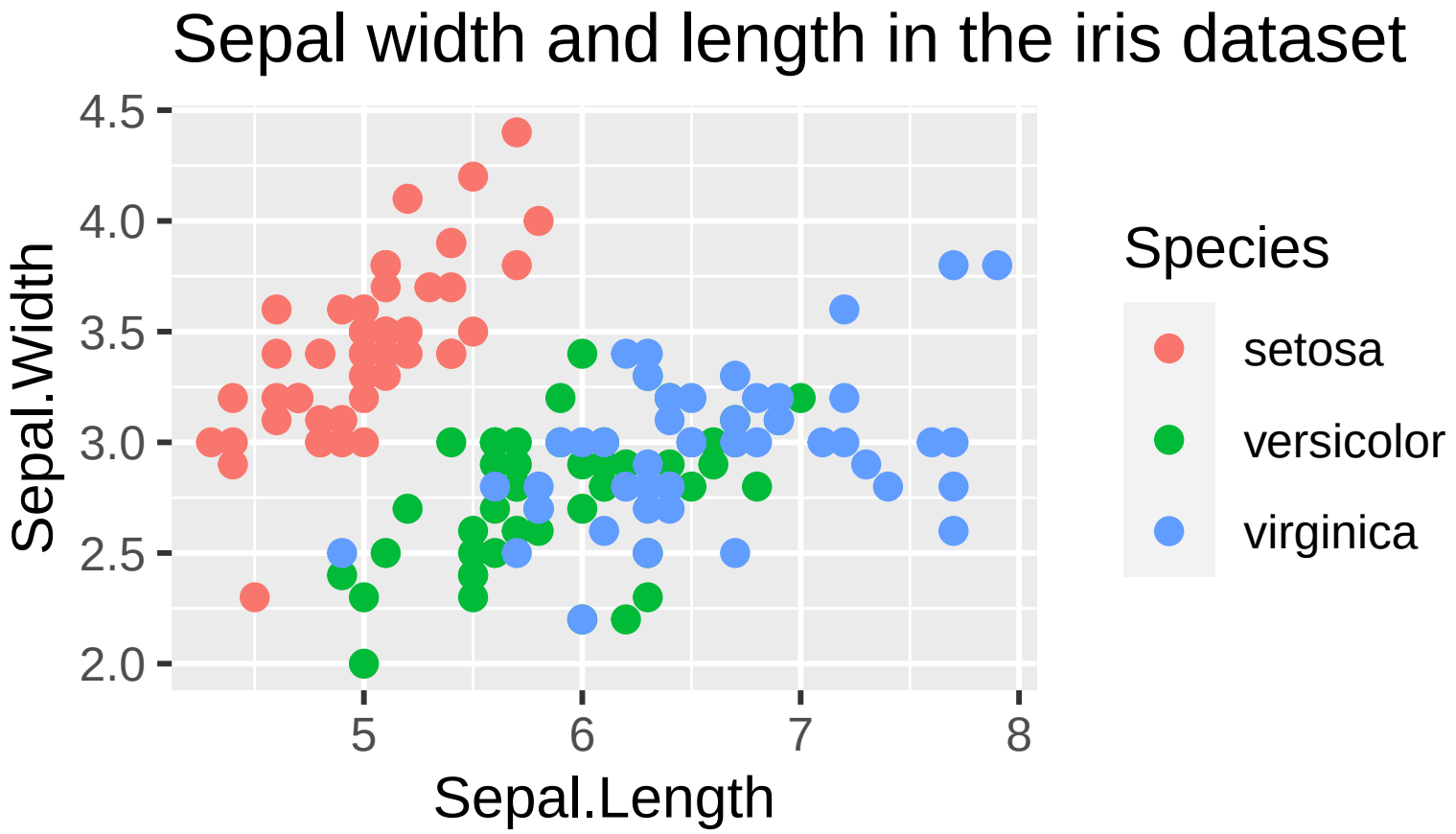


2. `geom_point()` adds the dots that produce the scatter plot

Example 2: fancier scatter plots

1. Use `color` argument to colour dots by group and add a legend
2. Use `size` argument to change size of the dots
3. Use `ggtitle( )` to add a title

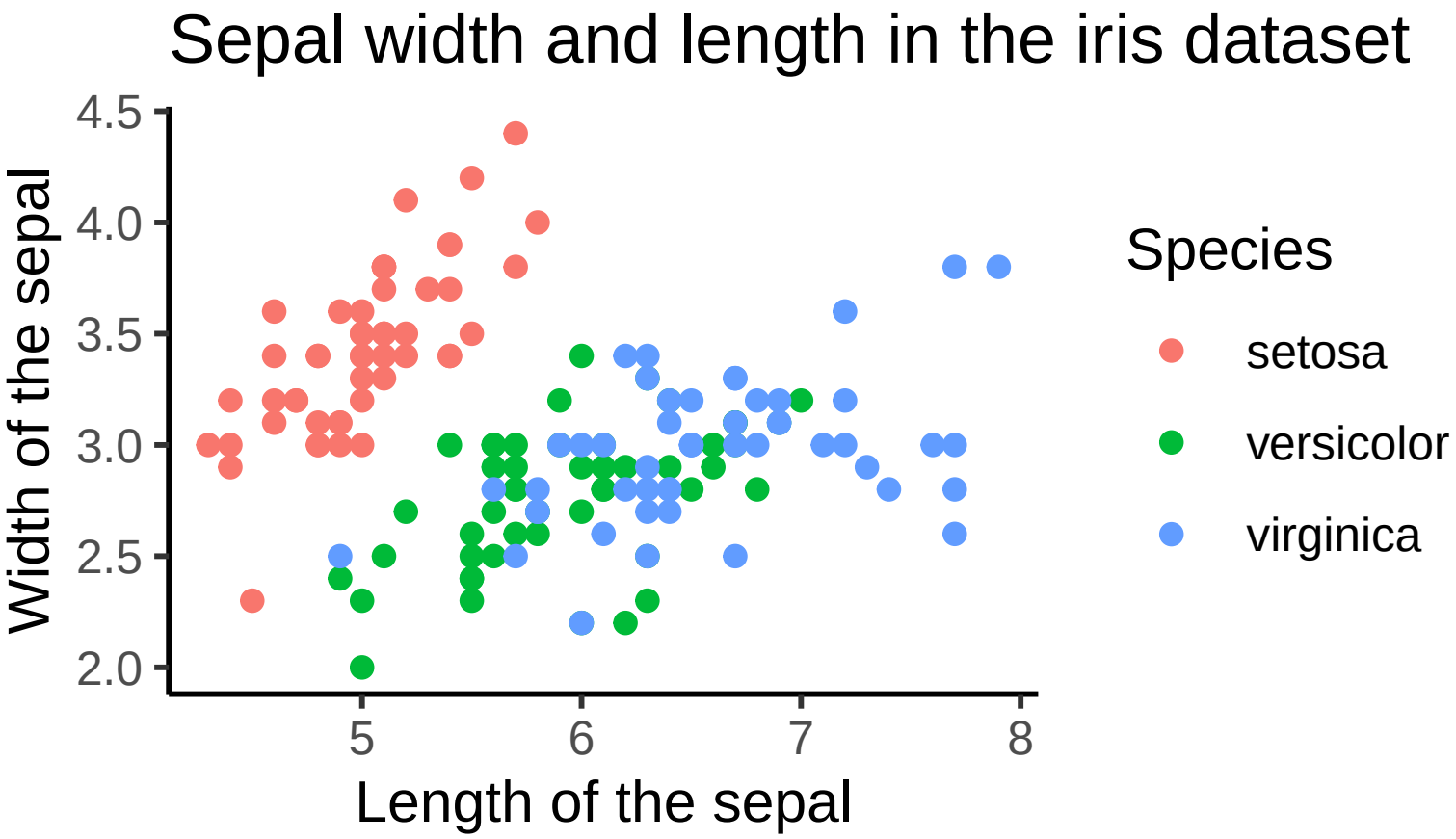
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point(size = 2) +
  ggtitle('Sepal width and length in the iris dataset')
```



Example 2: fancier scatter plots (cont'd)

4. Use `labs( )` to change labels of x and y axis
5. Use `theme_classic( )` to remove gray background and add axes

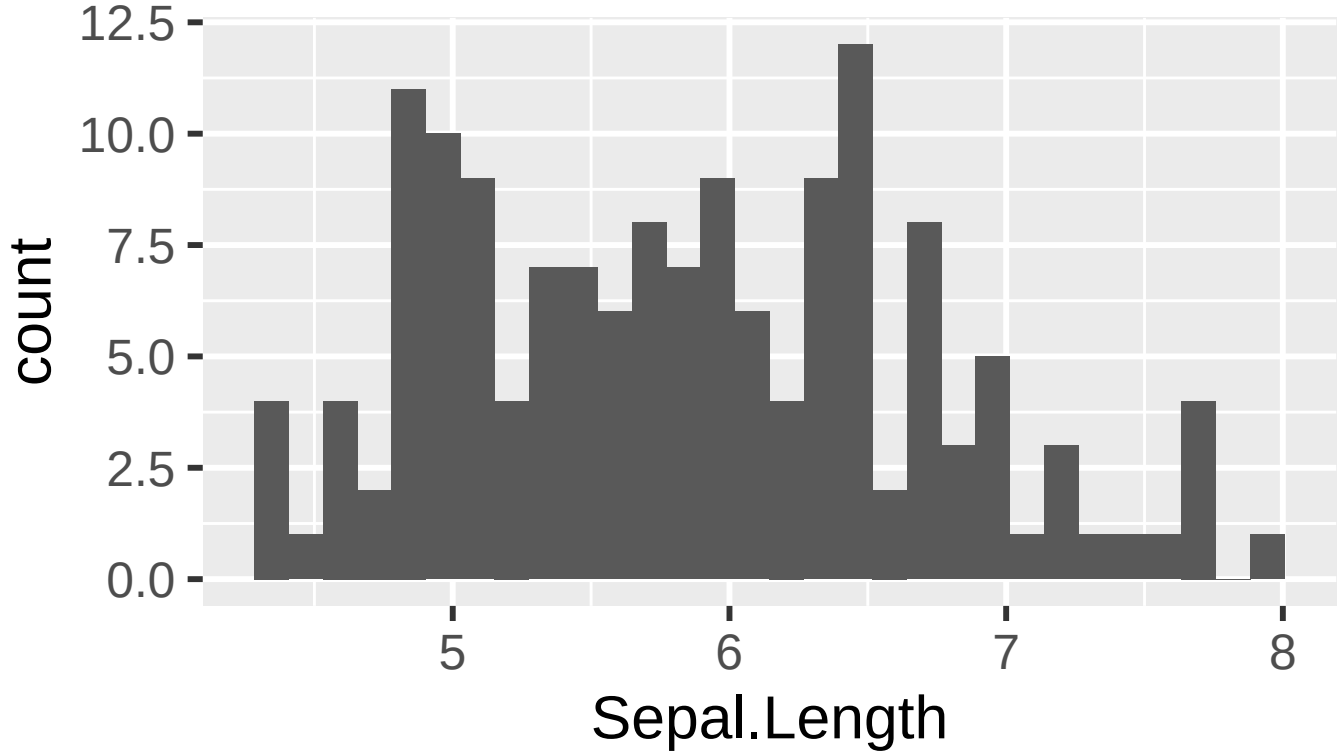
```
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, color=Species)) +
  geom_point() + theme_classic() +
  ggtitle('Sepal width and length in the iris dataset') +
  labs(x = 'Length of the sepal', y = 'Width of the sepal',)
```



Example 3: histogram

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

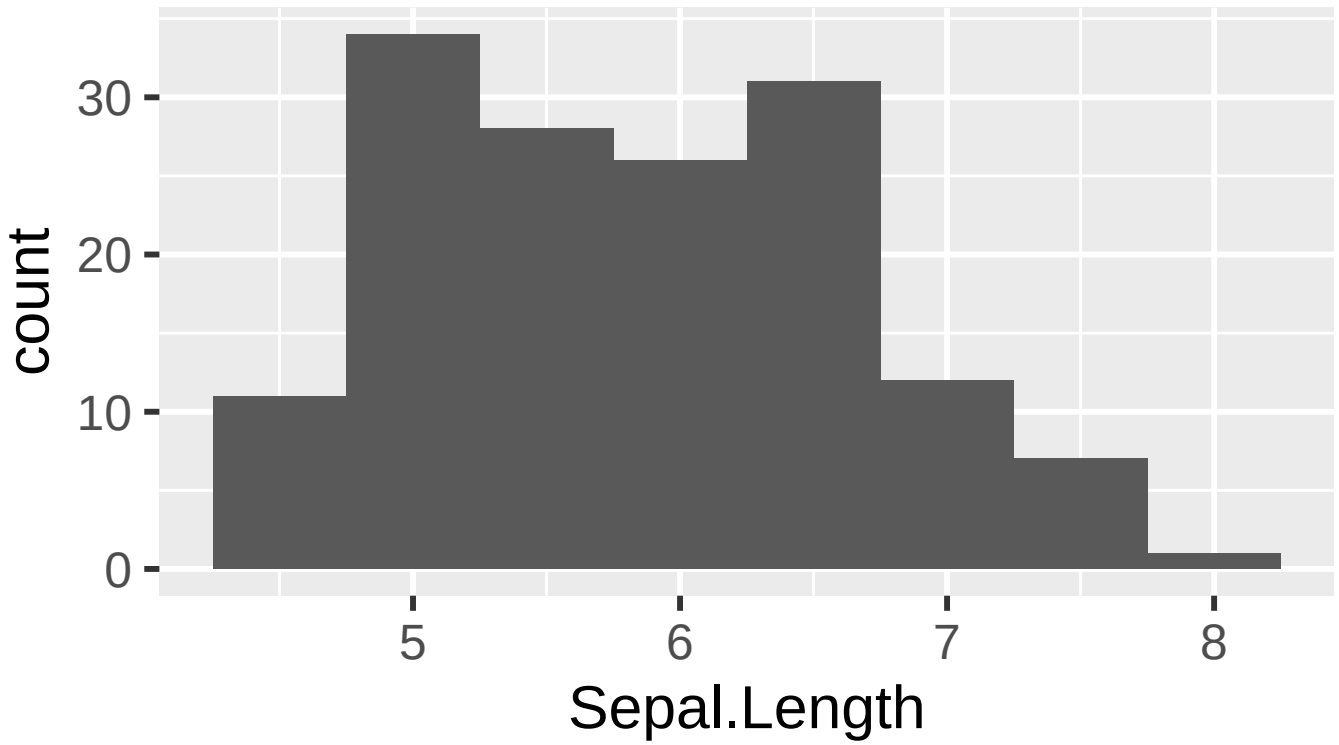
```
## `stat_bin()` using `bins = 30`. Pick better value with
## `binwidth`.
```



Example 3: histogram (cont'd)

- `binwidth` argument changes the width of the intervals on the x axis

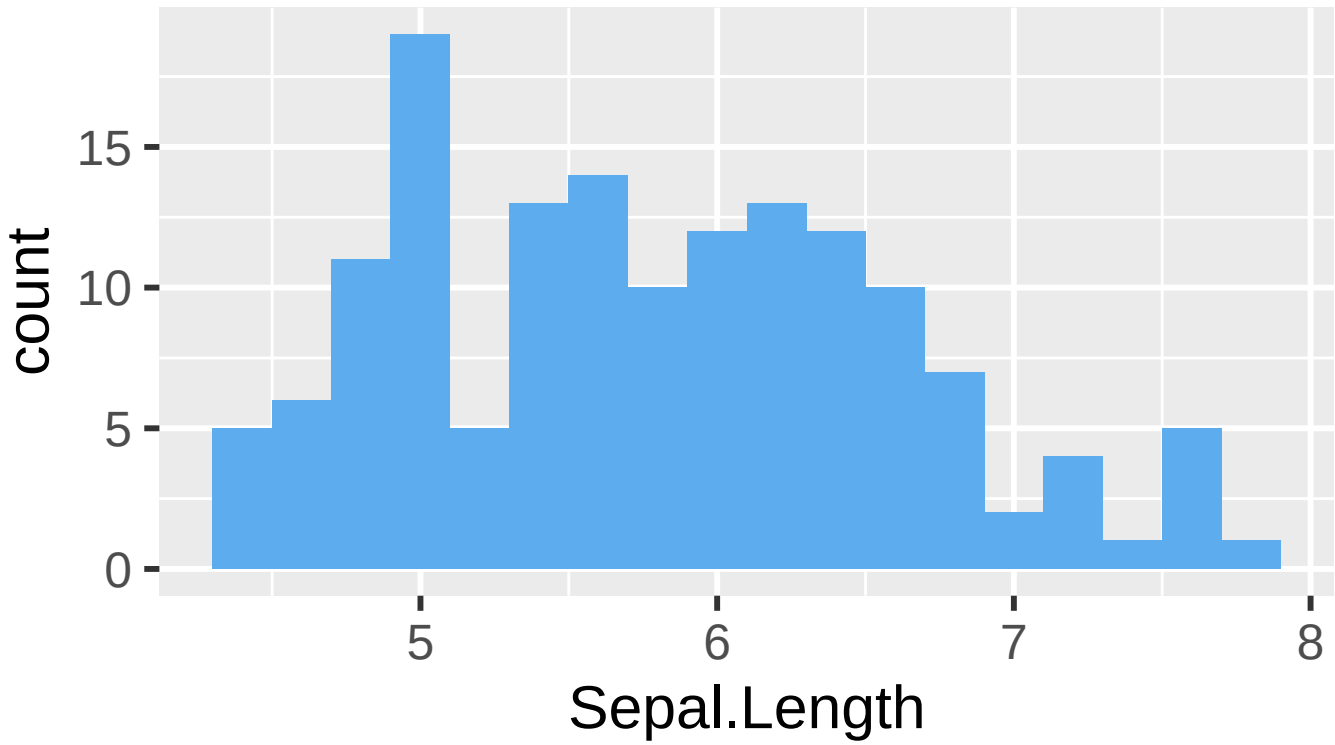
```
ggplot(iris, aes(x=Sepal.Length)) +
  geom_histogram(binwidth = 0.5)
```



Example 3: histogram (cont'd)

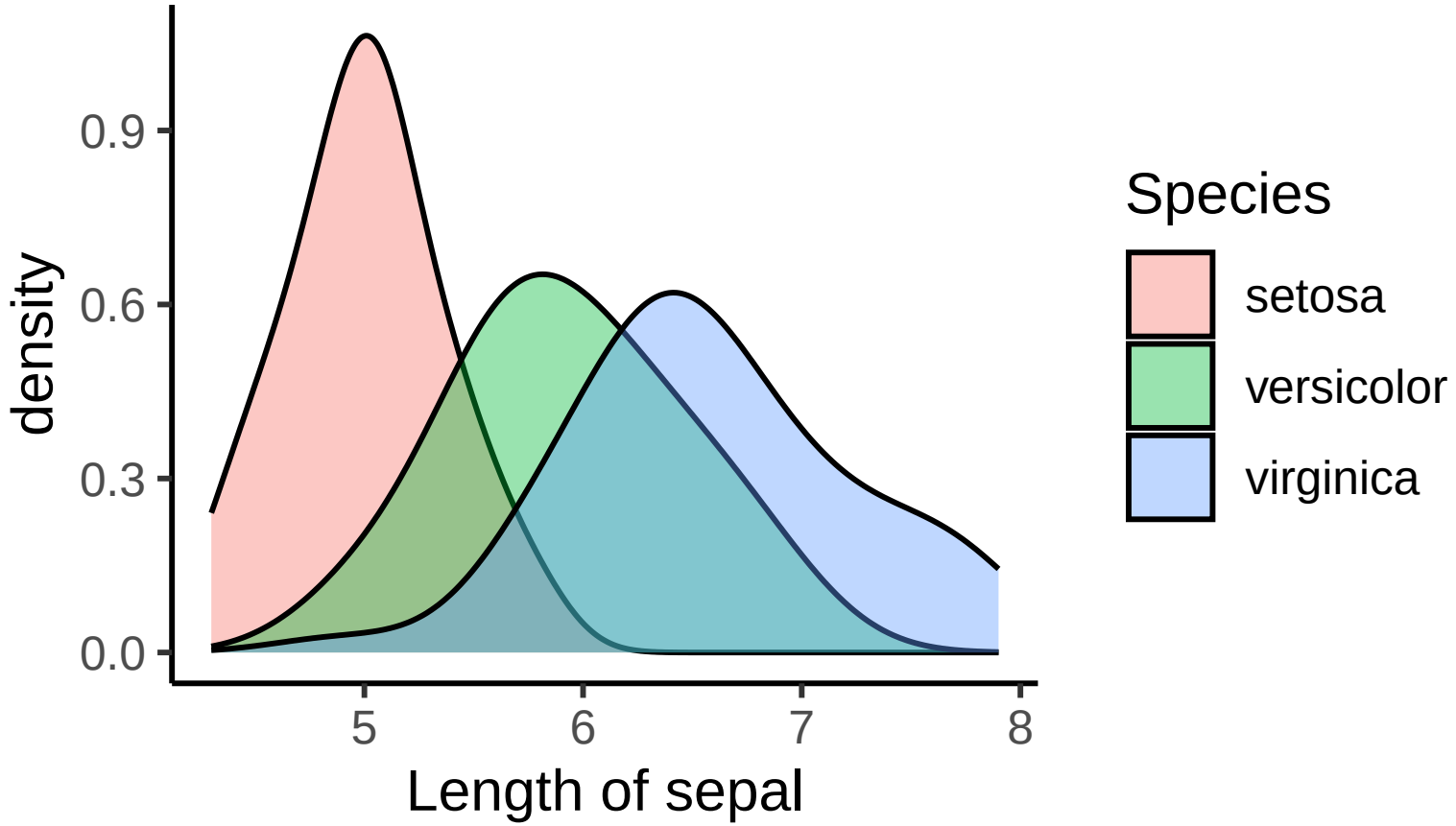
- `fill` argument changes the colour of the histogram

```
ggplot(iris, aes(x=Sepal.Length)) +
  geom_histogram(binwidth = 0.2, fill = 'steelblue2')
```



Example 4: density plot

```
ggplot(data=iris,
       aes(x=Sepal.Length, group=Species, fill=Species)) +
  geom_density(adjust=1.5, alpha=.4) +
  theme_classic() + labs(x = 'Length of sepal')
```



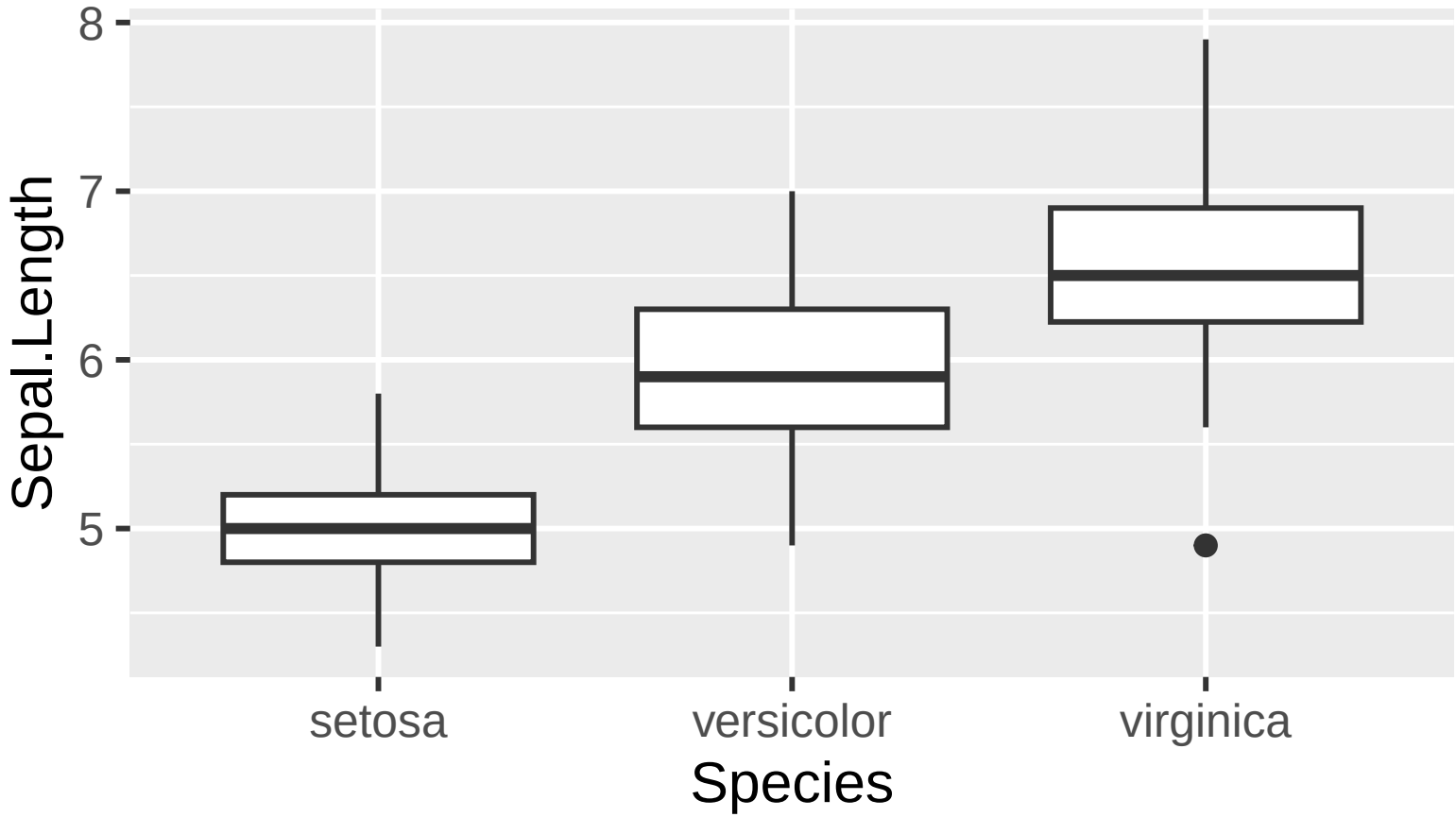
ggplot( ) charts are R objects

- Charts created with ggplot( ) are objects (lists!) that can be saved in R's global environment
- You add elements to such objects using the + operator
- Execute the code below in RStudio and observe what happens:

```
g1 = ggplot(data=iris,
            aes(x=Sepal.Length, group=Species, fill=Species))
g2 = g1 + geom_density(adjust=1.5, alpha=.4)
g3 = g2 + labs(x = 'Length of sepal')
g4 = g3 + theme_classic()
g1
g2
g3
g4
```

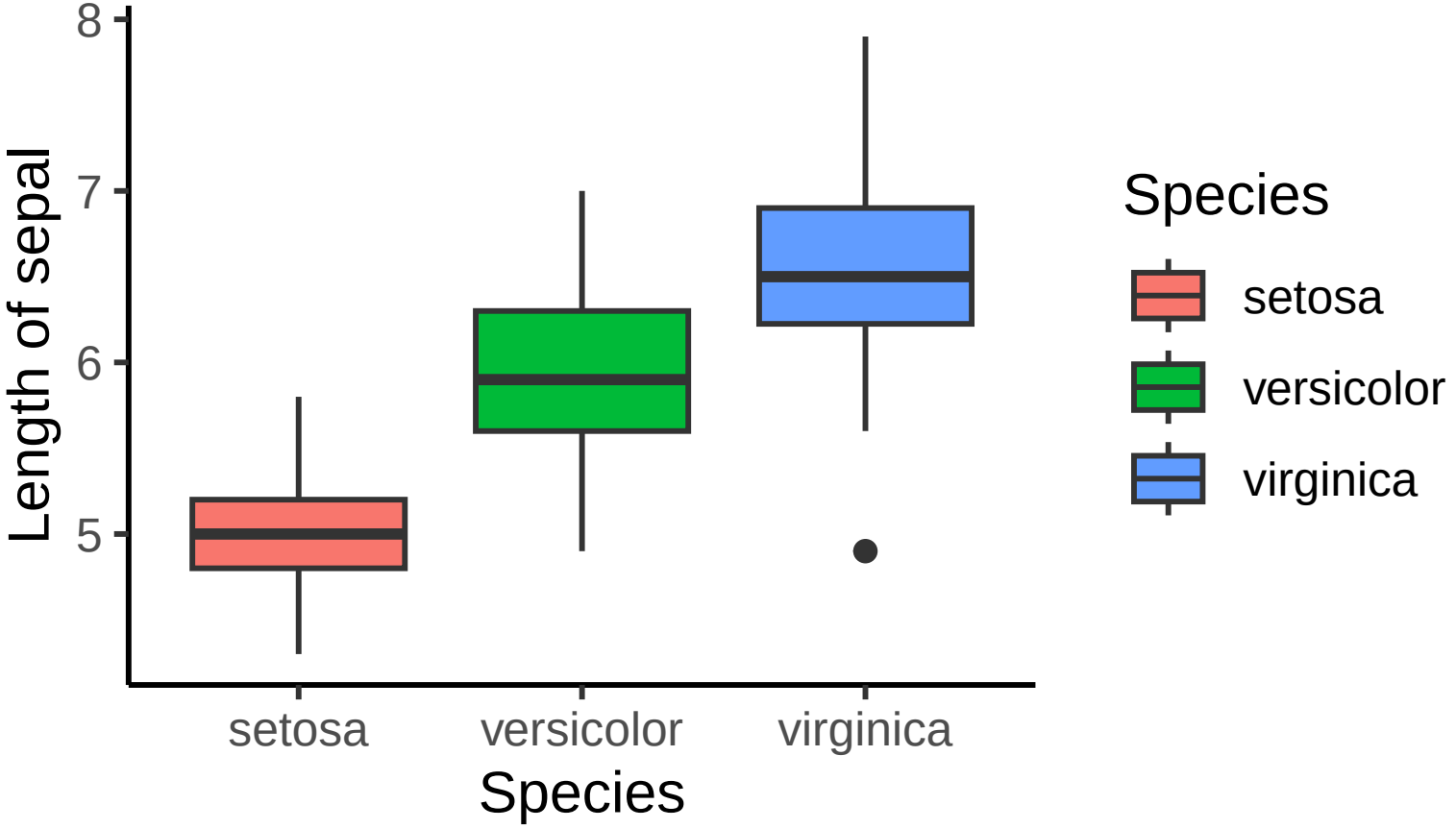
Example 5: boxplot

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



Example 5: boxplot (cont'd)

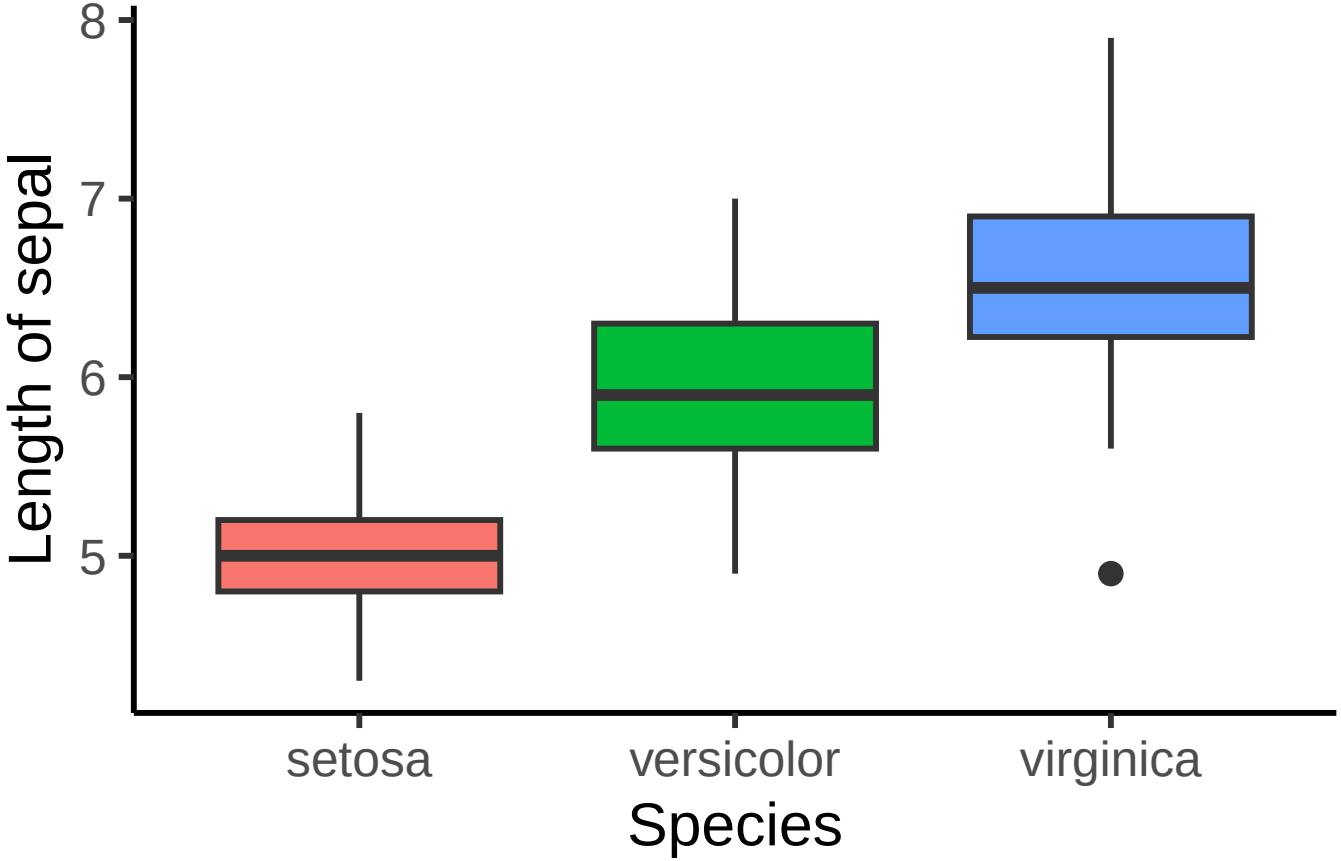
```
bp = ggplot(data=iris,
            aes(x=Species, y=Sepal.Length, fill = Species)) +
  geom_boxplot() + labs(y = 'Length of sepal') +
  theme_classic()
bp
```



How can we remove the legend?

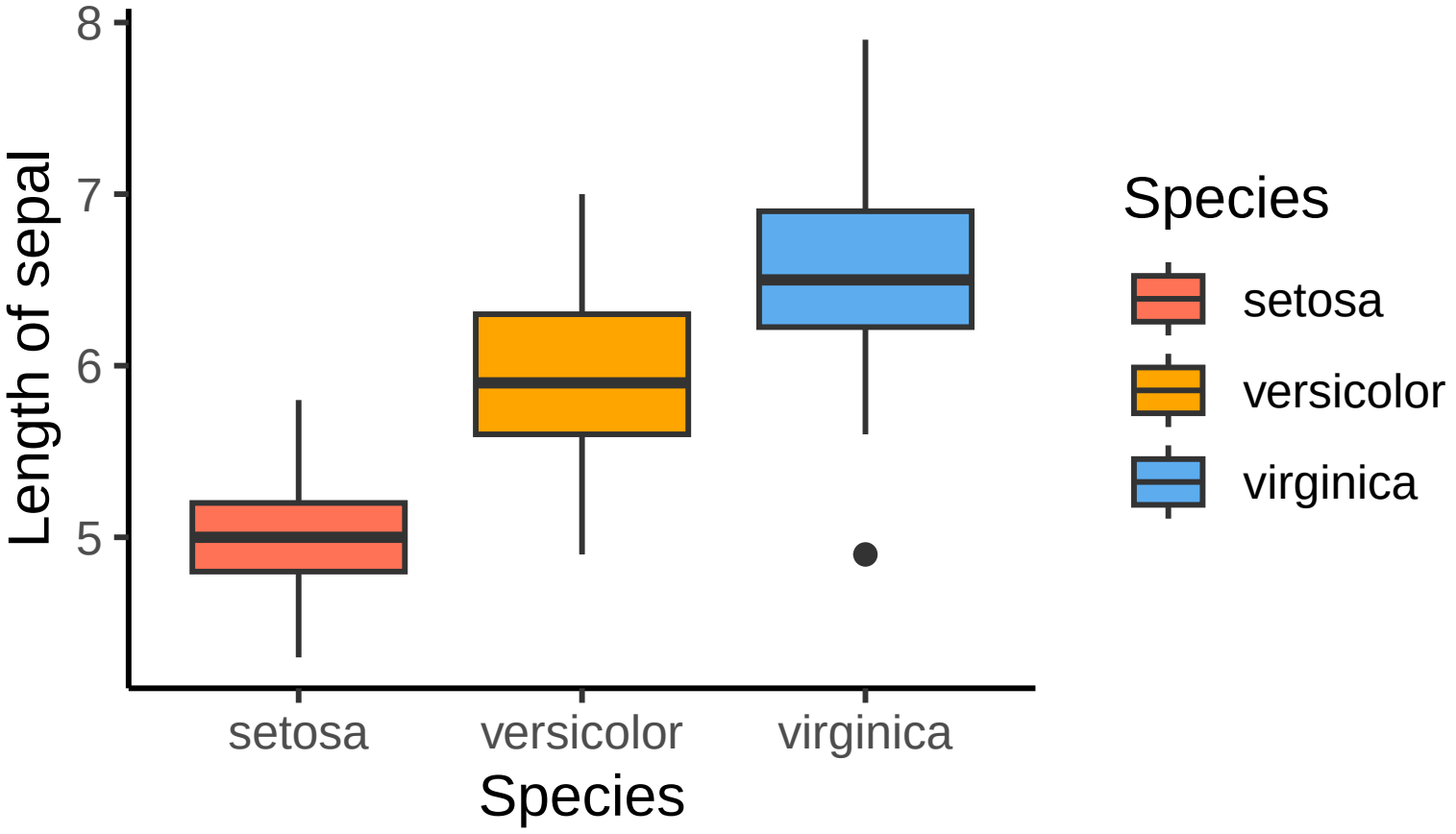
- Legend in previous chart could be seen as redundant
- You can remove a legend with `theme(legend.position = "none")`

```
bp + theme(legend.position = "none")
```



How to change the default colours?

```
bp + scale_fill_manual(
  values = c('coral1', 'orange', 'steelblue2')
)
```



More about ggplot2 and data visualization

- Useful ggplot2 references:
  1. Section 3.4 of Braun and Murdoch (2021)
  2. ggplot2's "cheatsheet"
- General "data visualization encyclopedia": the R Graph Gallery contains many examples of how to make create and customize different charts using base R, ggplot2 and several other R packages