# Basic ggplots

Clarke van Steenderen

2023-08-07

## R tutorial 1: Using ggplot to create basic plots

Here, we will use the popular **iris** dataset, collected by Edgar Anderson in 1936, to illustrate how ggplot can be used to create beautiful graphics. It contains sepal and petal measurements for 50 individuals from three *Iris* species; namely *I. setosa*, *I. versicolor*, and *I. virginica*.

Check the R dataset package for other interesting data sets to tinker with.

**Some notes on syntax:**

- **package_name::** specifies the package name, where the two colons access all the functions associated with that package
- Hashes (#) denote comments, that are not read by R. Use these to annotate your code
- In ggplot, the plus sign (+) means that you are adding additional "layers" to your plot
- An equal sign (=) is synonymous with the arrow (->). You can use either to assign names to objects
- Dollar signs ($) are like keys that access features of an object, such as columns or lists
- Use a question mark (?) to access the help file for a particular package (e.g. **?ggplot**)

**Let's get started!**

```r
if (!require("pacman"))
  install.packages("pacman")
```

```
## Loading required package: pacman
```

```r
pacman::p_load(datasets, xlsx, janitor, ggplot2, Rmisc, dplyr, ggpubr)

# access the embedded dataset
iris_data = datasets::iris
iris_data = janitor::clean_names(iris_data)

# for practice, write this data to the project folder as a .csv file
write.csv(iris_data, file = "data/iris_data.csv", quote = FALSE, row.names = FALSE)
# now read it back in
iris_data_readin = read.csv("data/iris_data.csv")

# now as an Excel file
xlsx::write.xlsx(iris_data, file = "data/iris_data.xlsx",
                 sheetName = "iris_data", row.names = FALSE)
iris_data_readin_xlsx = xlsx::read.xlsx(file = "data/iris_data.xlsx", sheetIndex = 1)

# make iris species a factor/grouping variable
iris_data$species = as.factor(iris_data$species)
levels(iris_data$species)
```

```
## [1] "setosa"     "versicolor" "virginica"
```

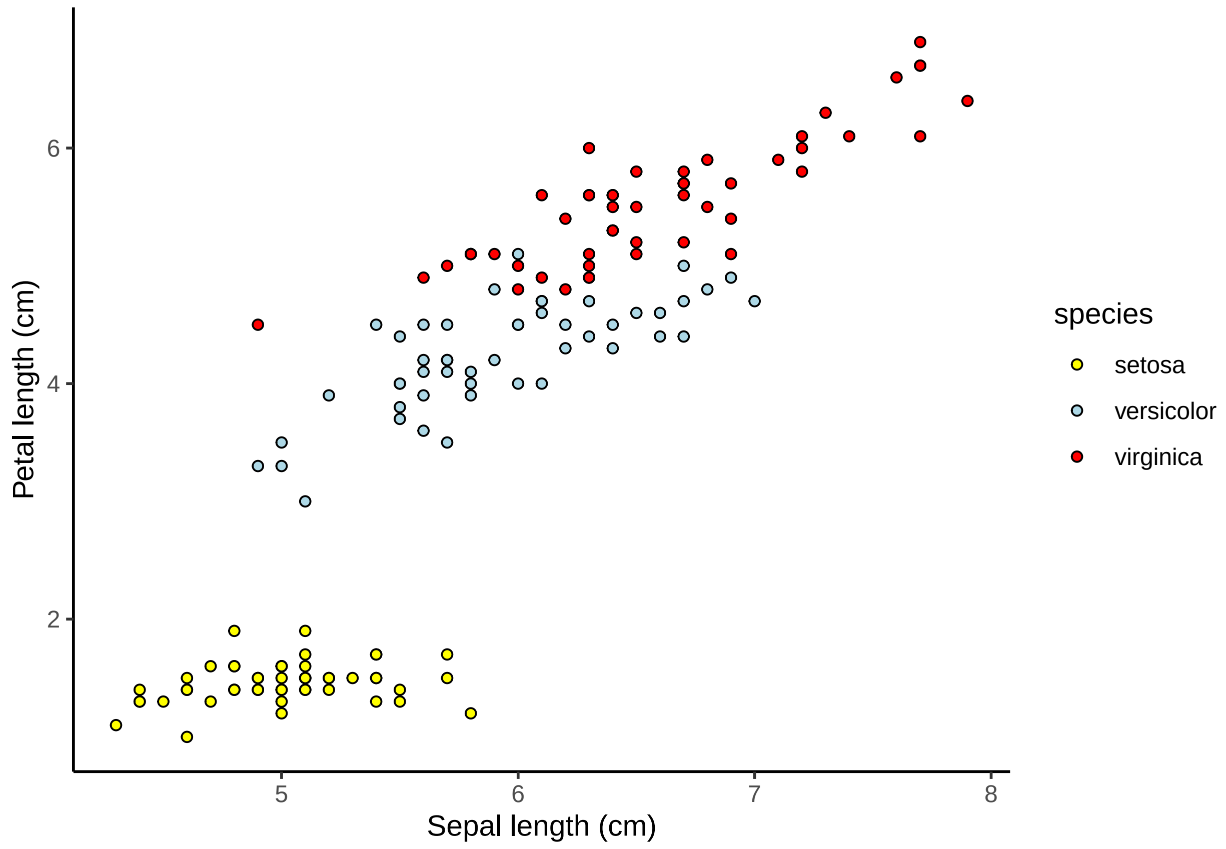
```
# summarySE function gets summary statistics for a chosen variable across groups
petal_summary = Rmisc::summarySE(data = iris_data, measurevar = "petal_length",
                                 groupvars = "species")
```

## ggplot

### Using geom_point()

geom_point() generates dot plots, like this:

```
ggplot2::ggplot(data = iris_data, aes(x=sepal_length, y=petal_length,
                                      fill=species)) +
  geom_point(shape = 21) +
  scale_fill_manual(values = c("yellow", "lightblue", "red")) +
  xlab("Sepal length (cm)") +
  ylab("Petal length (cm)") +
  theme_classic()
```



Here, we create a subset for each *Iris* species, using the **filter()** function in the **dplyr** package:

```
setosa = dplyr::filter(iris_data, species=="setosa")
versicolor = dplyr::filter(iris_data, species=="versicolor")
virginica = dplyr::filter(iris_data, species=="virginica")
```

Now we can run a quick correlation test to see if there is a significant correlation between sepal and petal length in *I. versicolor*:

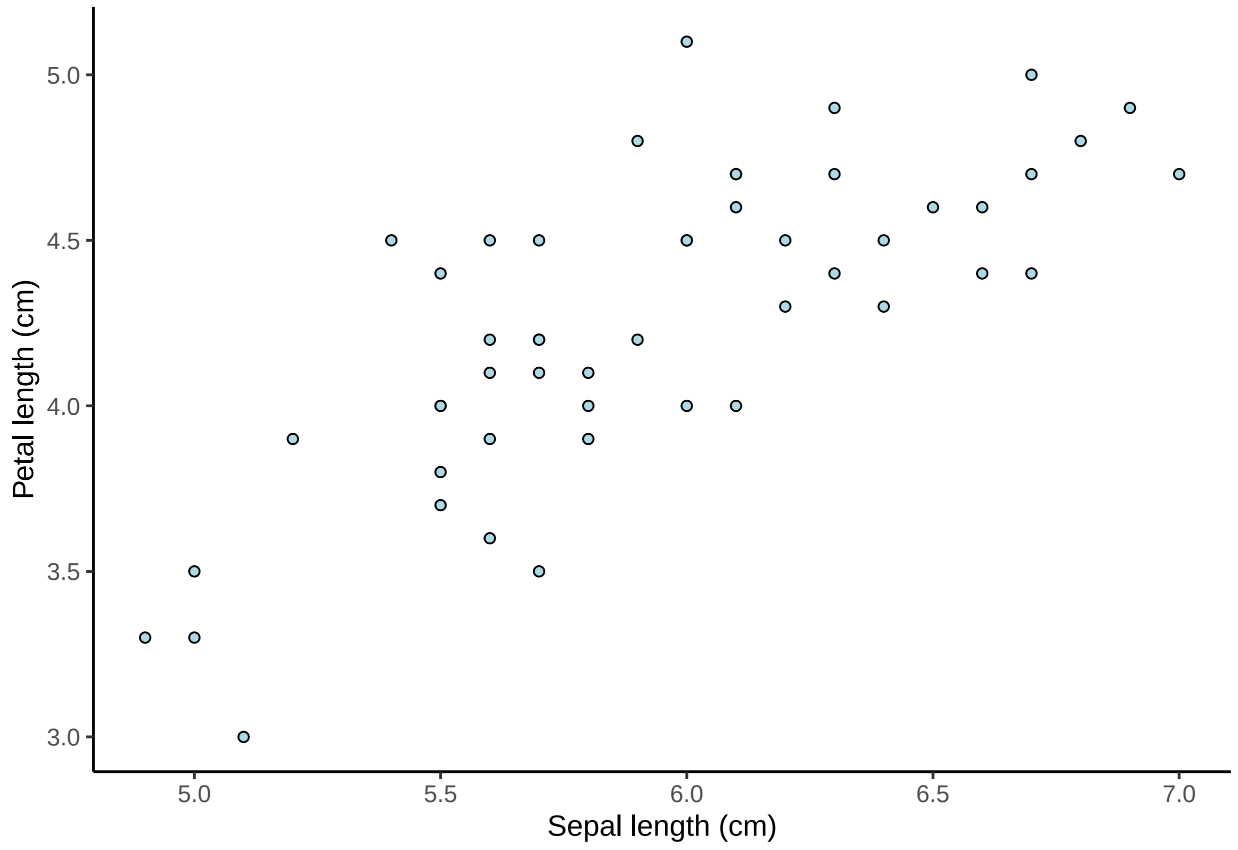
```
cor.test(versicolor$sepal_length, versicolor$petal_length, method = "pearson")
```

```
##
##  Pearson's product-moment correlation
##
## data:  versicolor$sepal_length and versicolor$petal_length
## t = 7.9538, df = 48, p-value = 2.586e-10
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.6020680 0.8532995
## sample estimates:
##      cor
## 0.754049
```

Here we see a significant correlation ($p < 0.05$), with a correlation coefficient of $r = 0.75$. Typically, a value greater than 0.7 is considered to be strong.

Let's plot only *I. versicolor*, using **dplyr::filter()**:

```
ggplot2::ggplot(data = versicolor,
                aes(x=sepal_length, y=petal_length)) +
  geom_point(shape = 21, fill = "lightblue") +
  xlab("Sepal length (cm)") +
  ylab("Petal length (cm)") +
  theme_classic()
```



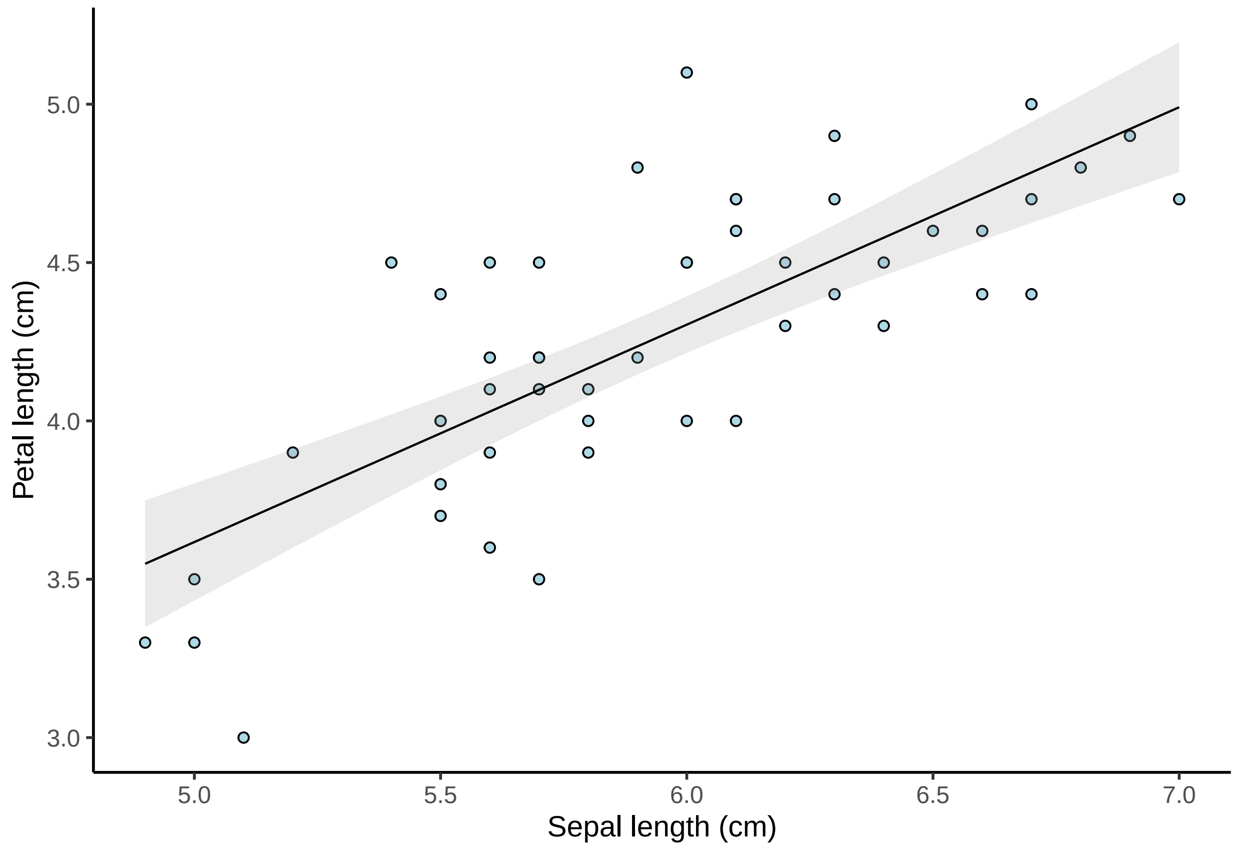
Let's run a quick linear model, where we check whether there is a relationship between petal and sepal length:

```r
versicolor_lm = lm(data = versicolor, petal_length~sepal_length)
summary(versicolor_lm)
```

```
##
## Call:
## lm(formula = petal_length ~ sepal_length, data = versicolor)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.68611 -0.22827 -0.04123  0.19458  0.79607
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.18512    0.51421   0.360     0.72
## sepal_length  0.68647    0.08631   7.954 2.59e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3118 on 48 degrees of freedom
## Multiple R-squared:  0.5686, Adjusted R-squared:  0.5596
## F-statistic: 63.26 on 1 and 48 DF,  p-value: 2.586e-10
```

If we plot the trend line for this linear regression, the intercept = 0.19, and the gradient = 0.69. The R^2 value is 0.56 ($p < 0.05$). Let's add this line to our dot plot, with a confidence interval band:

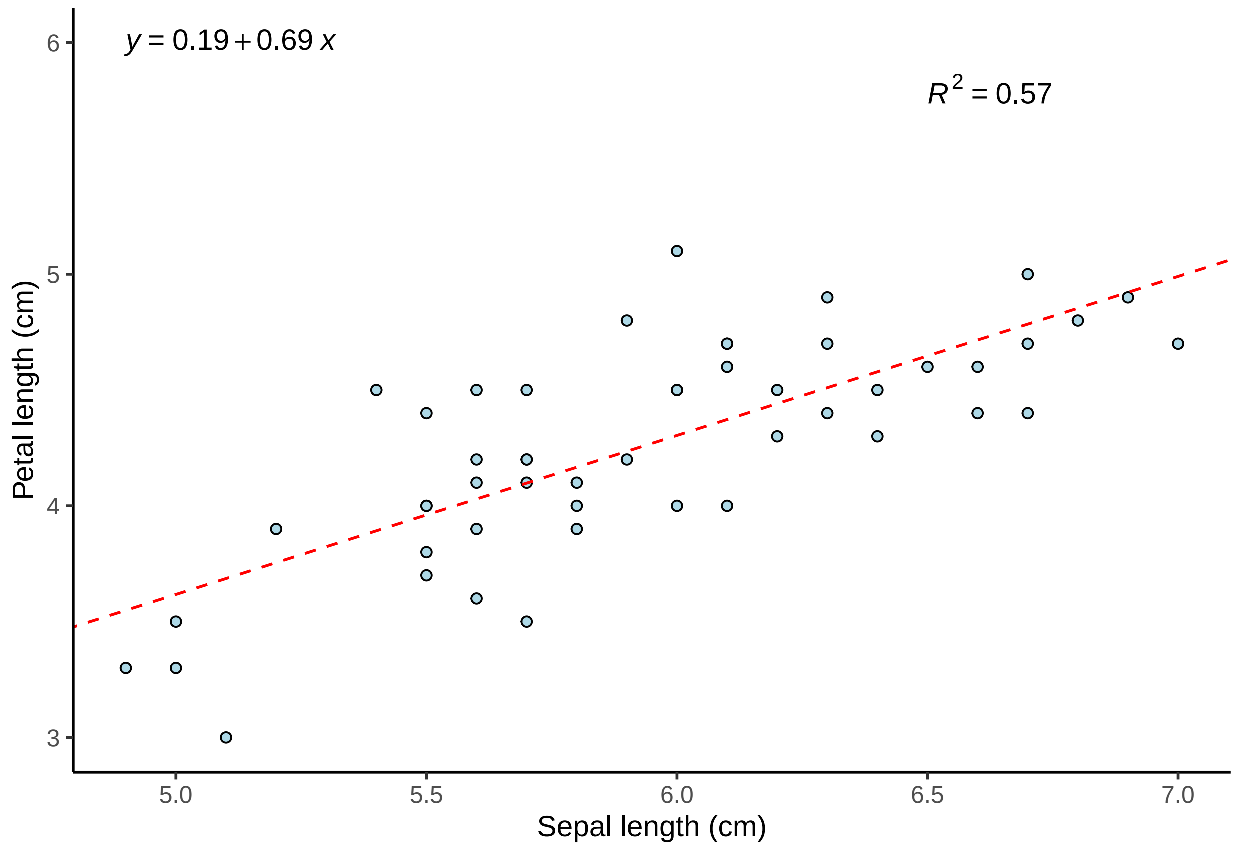
```r
ggplot2::ggplot(data = versicolor,
                aes(x=sepal_length, y=petal_length)) +
  geom_point(shape = 21, fill = "lightblue") +
  xlab("Sepal length (cm)") +
  ylab("Petal length (cm)") +
  theme_classic() +
  stat_smooth(method = "lm", formula = y~x, geom = "smooth",
              alpha = 0.2, color = "black", linewidth = 0.4, se = TRUE)
```



```r
# set se = FALSE to remove the confidence interval band
```

We can also add this line using the **geom_abline()** function:

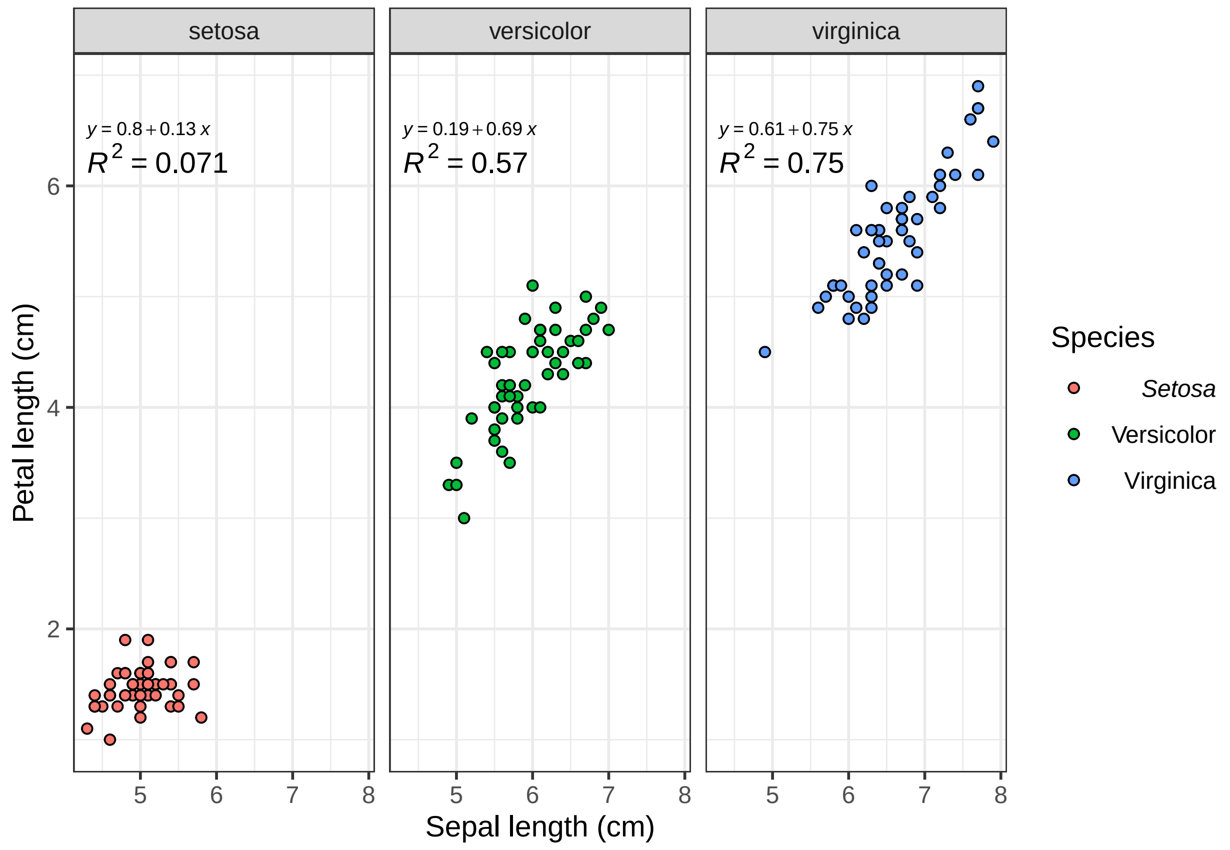
```r
coeff = coefficients(versicolor_lm)
intercept = coeff[1]
gradient = coeff[2]

ggplot2::ggplot(data = versicolor,
                aes(x=sepal_length, y=petal_length)) +
  geom_point(shape = 21, fill = "lightblue") +
  # add the abline
  geom_abline(intercept = intercept, slope = gradient, linewidth = 0.5,
              color = "red", linetype = "dashed") +
  xlab("Sepal length (cm)") +
  ylab("Petal length (cm)") +
  theme_classic() +
  # the stat_regline_equation() function in the ggpubr package adds the equation as text to the plot
  ggpubr::stat_regline_equation(label.y = 6, aes(label = after_stat(c(eq.label))) ) +
  # the stat_regline_equation() function in the ggpubr package adds the R-squared value as text to the
  ggpubr::stat_regline_equation(label.y = 5.8, label.x = 6.5,
                                aes(label = after_stat(c(rr.label))) )
```



Add facet_wrap() to separate the plots by species:

```
ggplot2::ggplot(data = iris_data, aes(x=sepal_length, y=petal_length, fill=species)) +
  geom_point(shape = 21) +
  scale_fill_manual(values = c("yellow", "lightblue", "red")) +
  xlab("Sepal length (cm)") +
  ylab("Petal length (cm)") +
  theme_bw() +
  facet_wrap(~species) +
  guides(fill=guide_legend(title="Species")) +
  scale_fill_discrete(labels=c(expression(italic('Setosa')), 'Versicolor', "Virginica")) +
  ggpubr::stat_regline_equation(label.y = 6.5, size = 2.5,
                                aes(label = after_stat(c(eq.label))) ) +
  ggpubr::stat_regline_equation(label.y = 6.25,
                                aes(label = after_stat(c(rr.label))) )
```

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

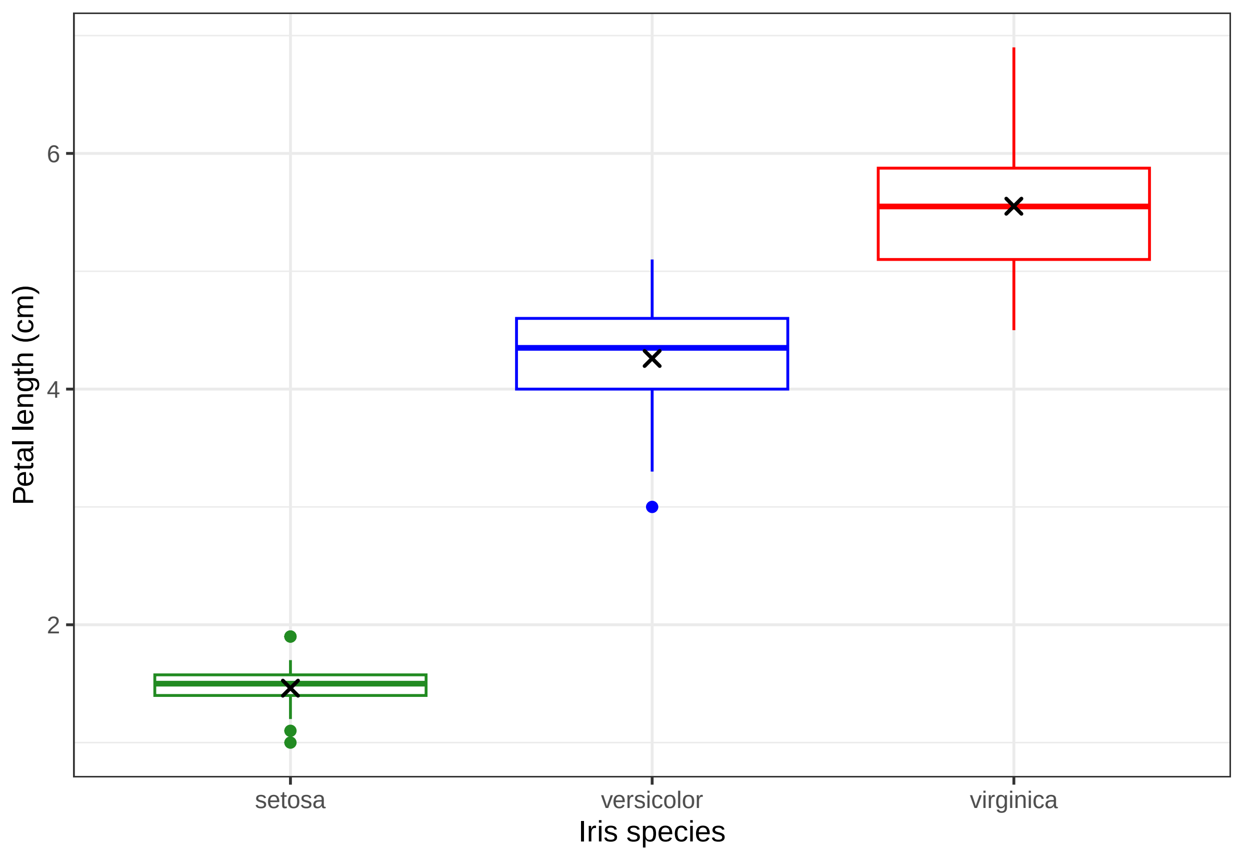


**Using geom_boxplot()**

The **geom_boxplot()** function creates box-and-whisker summary plots for your data:

```
ggplot2::ggplot(data = iris_data, aes(x=species, y=petal_length, color=species)) +
  geom_boxplot() +
  scale_color_manual(values = c("forestgreen", "blue", "red")) +
  xlab("Iris species") +
  ylab("Petal length (cm)") +
  theme_bw() +
  theme(legend.position="none") +
  # add points to denote means
  stat_summary(fun = "mean", colour = "black", shape = 4)
```

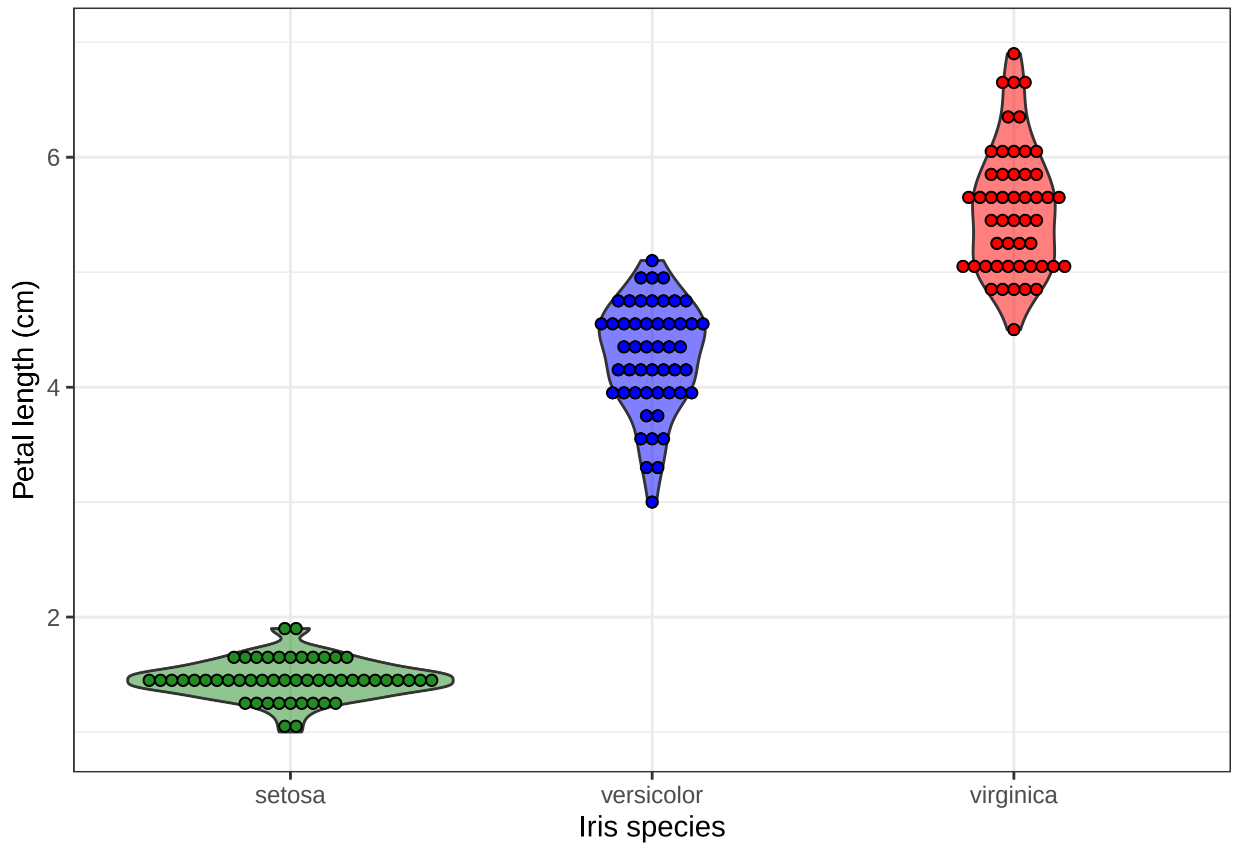
```
## Warning: Removed 3 rows containing missing values (`geom_segment()`).
```



**Using geom_violin()**

```
ggplot2::ggplot(data = iris_data, aes(x=species, y=petal_length, fill=species)) +
  geom_violin(alpha=0.5) +
  geom_dotplot(binaxis= "y",
               stackdir = "center",
               dotsize = 0.5) +
  scale_fill_manual(values = c("forestgreen", "blue", "red")) +
  xlab("Iris species") +
  ylab("Petal length (cm)") +
  theme_bw() +
  theme(legend.position="none")
```

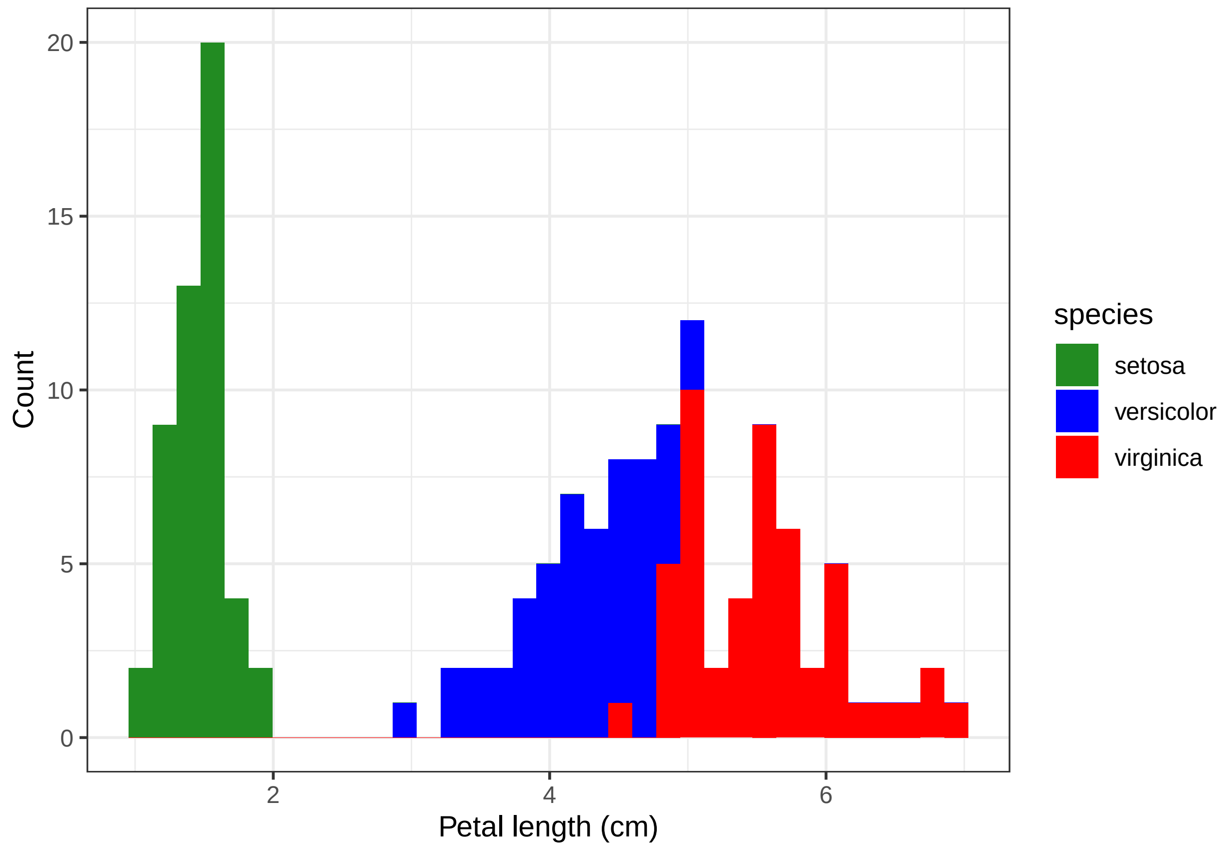
```
## Bin width defaults to 1/30 of the range of the data. Pick better value with
## `binwidth`.
```



**Using geom_histogram()**

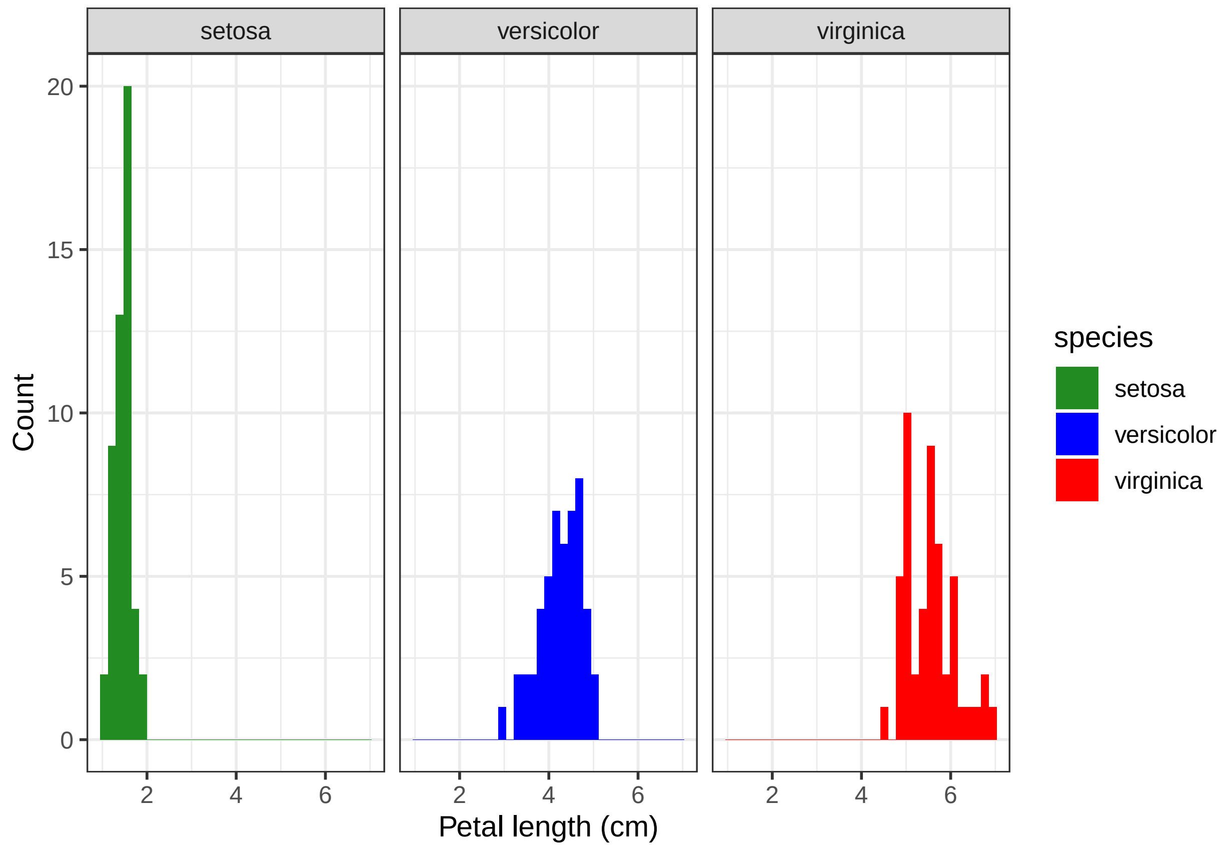
Use **geom_histogram** to generate histogram plots. Set an appropriate bin value:

```
ggplot2::ggplot(data = iris_data, aes(x=petal_length, fill=species)) +
  geom_histogram(bins = 35) +
  scale_fill_manual(values = c("forestgreen", "blue", "red")) +
  xlab("Petal length (cm)") +
  ylab("Count") +
  theme_bw()
```



Use facet_wrap() here:

```
ggplot2::ggplot(data = iris_data, aes(x=petal_length, fill=species)) +
  geom_histogram(bins = 35) +
  scale_fill_manual(values = c("forestgreen", "blue", "red")) +
  xlab("Petal length (cm)") +
  ylab("Count") +
  theme_bw() +
  facet_wrap(~species)
```



**Saving your plots as .PNG, .JPG, .PDF, or .SVG**

To save a plot, first assign it a name. We will call the last plot we made **petal_length_hist**:

```
petal_length_hist = ggplot2::ggplot(data = iris_data,
                                    aes(x=petal_length, fill=species)) +
  geom_histogram(bins = 35) +
  scale_fill_manual(values = c("forestgreen", "blue", "red")) +
  xlab("Petal length (cm)") +
  ylab("Count") +
  theme_bw() +
  facet_wrap(~species)
```

Now we can use the ggsave() function to save it in any format we like. Play around with the dimensions to get an ideal size:

```
# save as PNG
ggplot2::ggsave("outputs/petal_length.png", plot = petal_length_hist,
                dpi = 350, width = 8, height = 4)
# save as PDF
ggplot2::ggsave("outputs/petal_length.pdf", plot = petal_length_hist,
                dpi = 350, width = 8, height = 4)
# save as JPG
ggplot2::ggsave("outputs/petal_length.jpg", plot = petal_length_hist,
                dpi = 350, width = 8, height = 4)
```