Outline

What is R and why should you invest time in it?

R is, first and foremost, a programming language. That is, just as we use languages to communicate with each other on a daily basis, we use programming languages to tell a computer what to do.

But R is also more than that: it is a language specialized in statistical computing and graphics, which means that it allows you to do everything from making nice graphs to visualize your data to testing statistical differences between conditions, including complex design experiments. Moreover, thanks to its wide use by the community and its customizable nature, R makes it easy to download packages developed by other people to run specific analyses for your research.

Investing time in learning R can make a difference in your career by making you more independent in your research, as you will learn to make high-quality, publication-level figures for your reports and papers, and you will be able to run a sound statistical analysis to check how relevant your results are.

How to install R/RStudio?

To install R, open your web browser and go to the official R website. From there, click on the CRAN link and download the version from the location that is closest to yours. Then download the R version for the operating system you currently use and follow the installation instructions.

The easiest way to learn how to code in R is through RStudio, an Integrated Development Environment (IDE) that allows you to write your R code, execute it and visualize all the associated results and plots in the same place.

To install RStudio, go to the RStudio website. From there, click download R Studio which should take you to a page with the option to download the IDE for your current operating system. From there, follow the installation instructions.

Once you have R and RStudio installed on your system, you are ready to go!

Getting started with R

Upload your data

```
## iris csv dataset downloaded from https://gist.github.com/netj/8836201
input_file <- "your_input_directory/iris_data.csv"
```

Here `input_file` is what is called a "variable", which is just a series of characters that will redirect the machine to the information that is stored in them (in this case, the path to find your data on your computer: `your_input_directory/iris_data.csv`).

As a general rule, it is useful to give your variables telling names, so that you know what you are dealing with later. You could name this variable literally any other way (“x”, “y”, “a”, etc.) but you would easily get lost and forget what it actually represents. Here, we named the path to your data `input_file` because it is the data you will upload to your RStudio environment.
Just replace the example with the path of the file on your computer.

```
## read your input data as a data frame (tabular structure)
df <- read.table(input_file, sep = ",", header = TRUE)
```

Here you are telling RStudio to go find your input data and load it in your environment so you can use it for visualization and analysis. `read.table` is a general function to perform this task, which requires you to specify:

- the file that you want to read (`input_file`),
- `sep`: the symbol used to separate columns in your file (in this case commas, so `","`), and
- `header`: if the first line of your file contains column names, be sure to set this option to `TRUE`; if that is not the case, set it to `FALSE`

A quick way to check whether your data were loaded correctly into R is to display the first few lines of your new dataframe with:

```
# Visualize the first few lines of your uploaded data
head(df)
```

```
> head(df)
  sepal.length sepal.width petal.length petal.width variety
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
```

By default, `head` will show you the first six lines of your table, but you can visualize more by changing the "`n`" option.

Installing packages in R

As mentioned before, one of the main advantages of using R is that you can use packages developed for specific analyses or data visualization. Before we proceed with the data visualization section, we will show some examples of how to install packages in R.

Installing ggplot2

A very useful package to make publication-level graphs is `ggplot2`, which we will use in the data visualization section of this guide.

Here is how you can install it yourself:

```
# Install the ggplot2 package
```

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

By replacing “ggplot2” with the name of another package you can install any other package on CRAN, aka the Comprehensive R Archive Network, whether it is related to data visualization or not.

Once the installation completed, the next thing to do is to load the package to use:

```
# Load the ggplot2 package
library(ggplot2)
```

All done! Now you are ready to use any `ggplot2` function to make some great-looking graphs with ggplot2!

Installing packages from bioconductor

Many biology and bioinformatics-specific R packages are found on bioconductor, which allows you to download them from its repository with the `BiocManager` package.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("Package_name") # replace Package_name with the
package you want to install
```

Visualize your data through plots

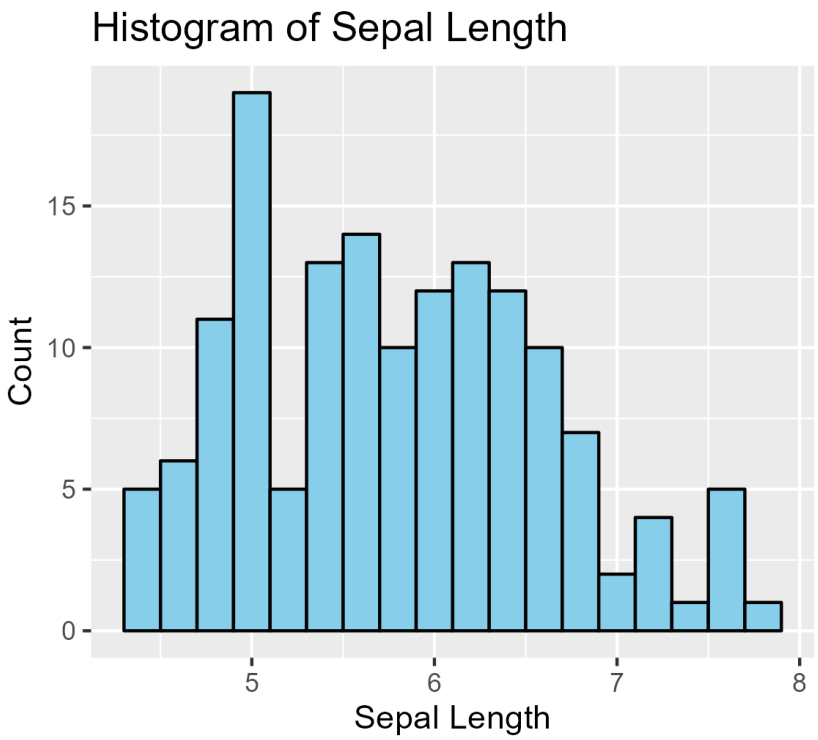
Now that you have uploaded your data and loaded `ggplot2` in your R environment, let’s start making some plots.

Histogram

Histograms are barplots showing the frequency of a certain observation within different numerical ranges, called bins. In general, they are a simple way to get a sense of how your data are distributed, which eventually determines the type of statistical test (see the section dedicated to statistical tests later in this guide) you can use to check if there are differences between e.g. different conditions or species.

As an example, we will plot the histogram of sepal length in the iris dataset:

```
# Histogram showing the distribution of sepal length
ggplot(df, aes(x = sepal.length)) +
  geom_histogram(binwidth = 0.2, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Sepal Length", x = "Sepal Length", y =
"Count")
```



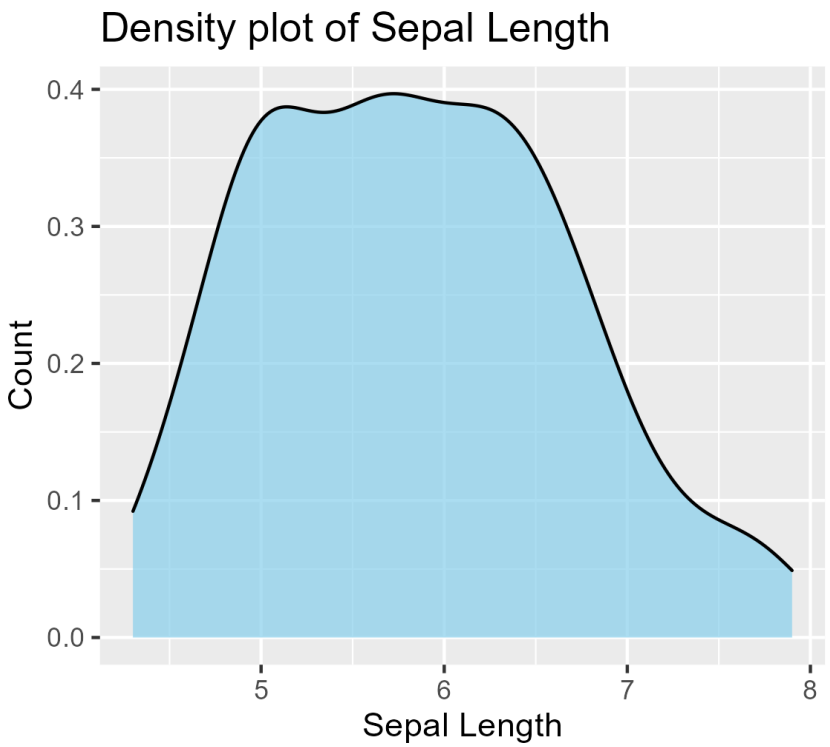
With ggplot2 you can also save your plot in your favorite format, to include it in presentations or papers. We will not include this line in the next examples but you can save them all the same way!

```
# Save plot to output file
ggsave("your_output_directory/sepal_length_histogram.png")
```

Kernel density

Another way to visualize the distribution of your data is through kernel density plots, which are a smoothed version of the histogram.

```
# Density plot to visualize data distribution
ggplot(df, aes(x = sepal.length)) +
  geom_density(fill = "skyblue", color = "black", alpha=.7) +
  labs(title = "Density plot of Sepal Length", x = "Sepal Length", y =
"Count")
```

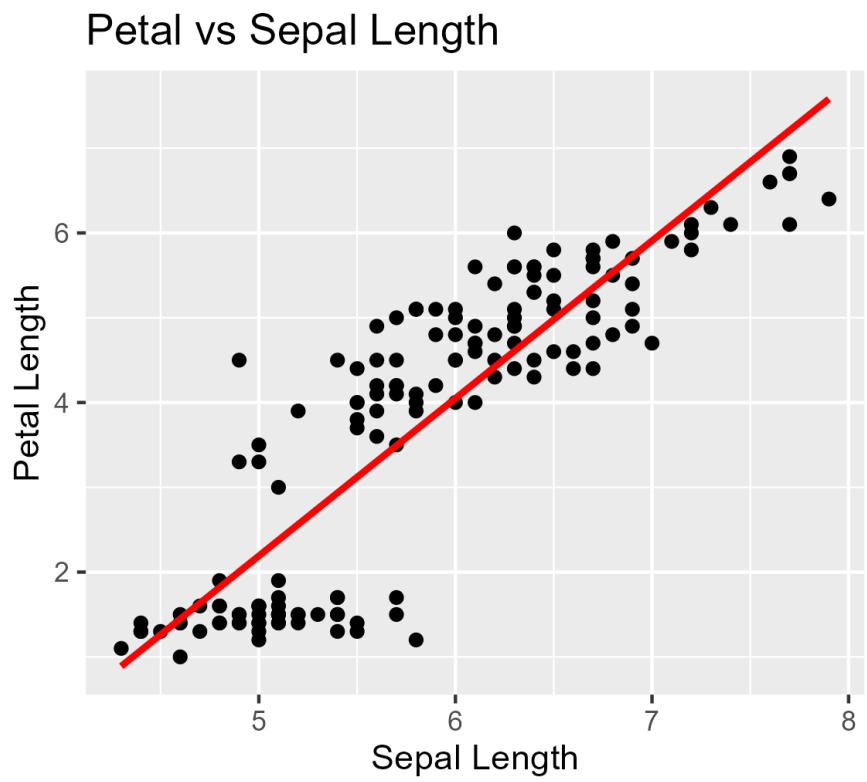


Scatterplot

When you have two continuous variables, you can plot them one against the other to reveal any relationship (e.g. correlation) between them.

If we take sepal and petal length in the iris dataset, for example, and we plot them one against the other, this is what we obtain:

```
# Scatter plot with linear regression line of petal against sepal length
ggplot(df, aes(x = sepal.length, y = petal.length)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  labs(title = "Sepal Width vs Sepal Length", x = "Sepal Length", y =
"Sepal Width")
```



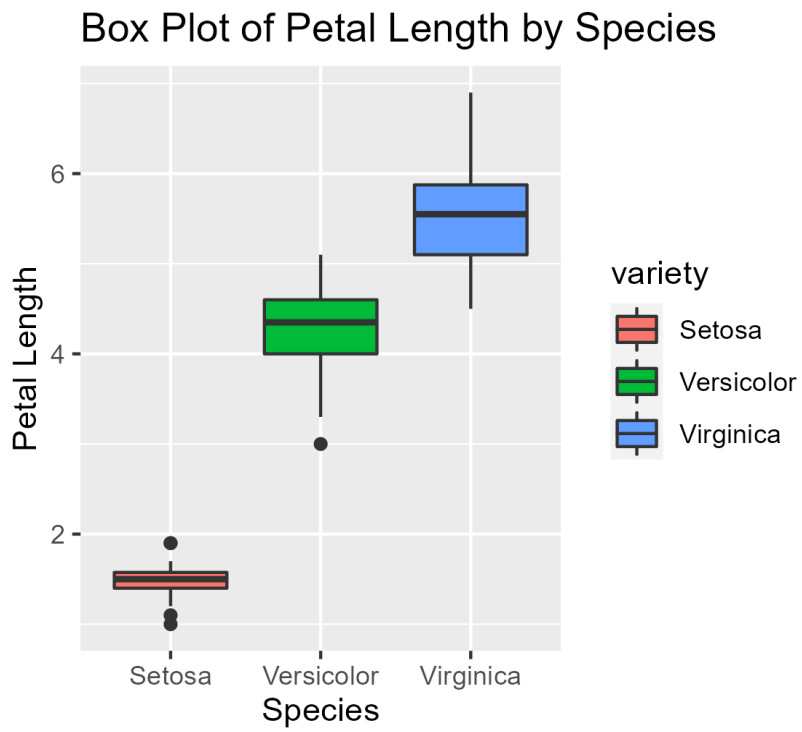
It looks like the two measurements are correlated, i.e. flowers with longer sepals tend to have longer petals, and vice versa. However, in order to establish the magnitude and validity of this correlation, you will have to run a statistical test (see the Correlation test section).

Boxplot

Boxplots are a nice way of visualizing the distribution of your data based on some key values: median, minimum and maximum value, first and third quartile.

In the next example, we will use the iris dataset to visualize petal length of all the measured plants, based on the species they belong to:

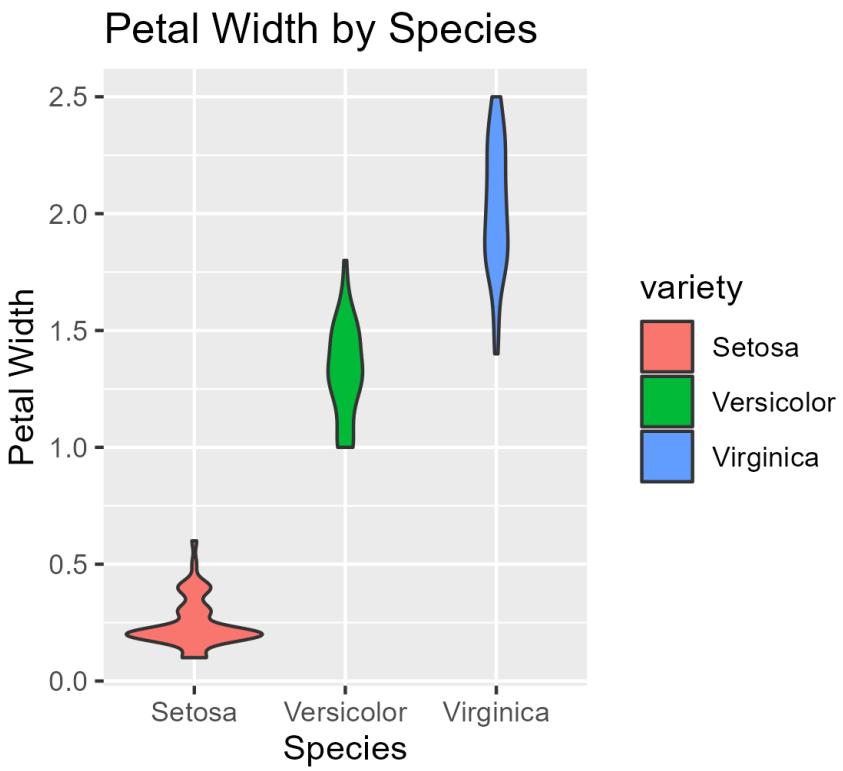
```
# Box plot of petal length by species
ggplot(df, aes(x = variety, y = petal.length, fill=variety)) +
  geom_boxplot() +
  labs(title = "Box Plot of Petal Length by Species", x = "Species", y =
"Petal Length")
```



Violin plots

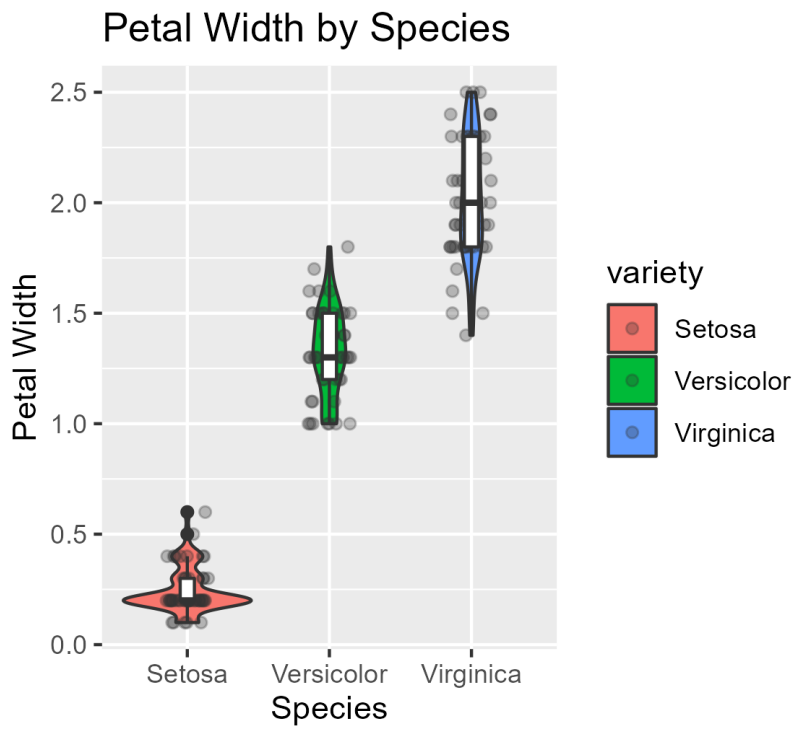
Violin plots are another way to visualize the distribution of your data, which is a hybrid of a box plot and a kernel density plot.

```
# Violin plot of petal width by species
ggplot(df, aes(x = variety, y = petal.width, fill = variety)) +
  geom_violin() +
  labs(title = "Petal Width by Species", x = "Species", y = "Petal Width")
```



In ggplot, you can overlay different types of plots one on top of the other to visualize more details of your data. For example, you can make a violin plot with a boxplot on top of it, and where you plot all the measured values at once:

```
# Violin plot with single data points and boxplot of petal width by species
ggplot(df, aes(x = variety, y = petal.width, fill = variety)) +
  geom_violin() +
  geom_jitter(width=0.15,height=0, col="gray20",alpha=0.3)+ # add data
points; alpha defines transparency
  geom_boxplot(fill="white",width=0.1)+ # add boxplot over violins
  labs(title = "Petal Width by Species", x = "Species", y = "Petal Width")
```



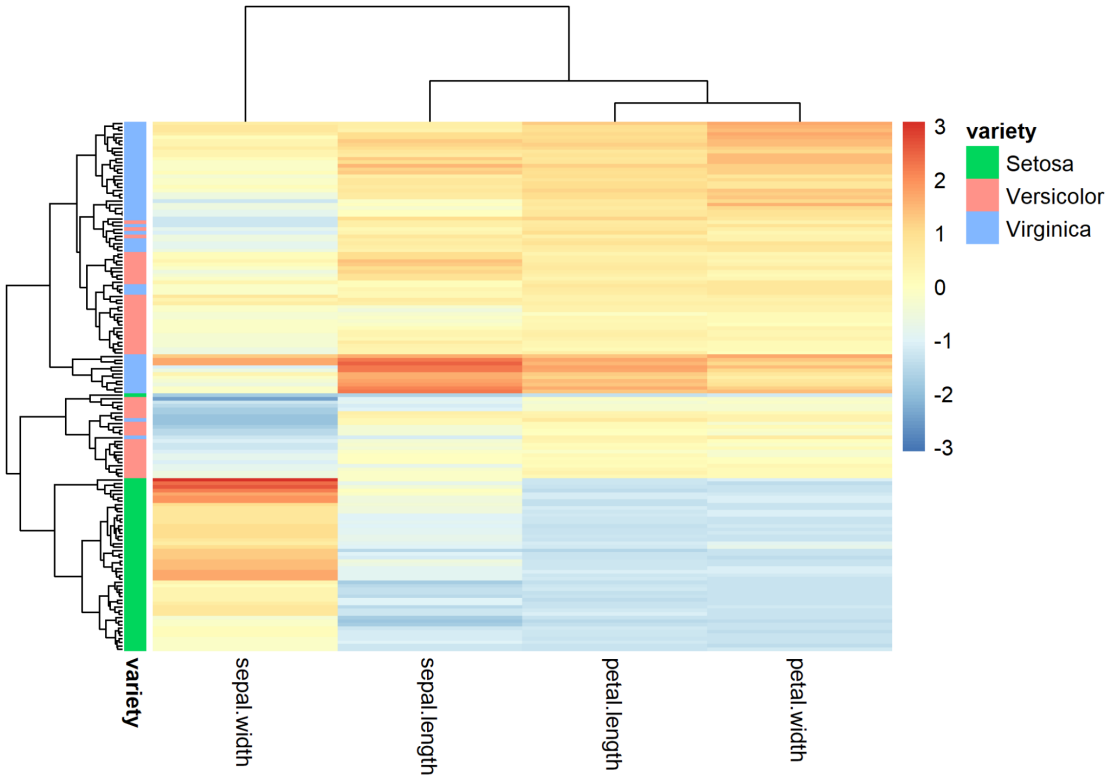
Heatmap

Heatmap is a graphical representation of data points which is often used to find specific patterns in the dataset. Together with clustering, heatmap is a commonly used approach to identify a subset of a dataset following a particular pattern.

```
#Load the "pheatmap" package
library("pheatmap") # you might need to install the package
#to install use: install.packages("pheatmap")

#Data processing for Heatmap
Heatmap_data <- df[,1:4] # only keep the first 4 columns of your df
head(Heatmap_data) # visualize the beginning of your new df ('data')
row.names(Heatmap_data) <- row.names(df)


# Plot your heatmap
pheatmap(Heatmap_data,
         scale = "column",
         annotation_row = df[, 5, drop = FALSE], # add species as color bar
         show_rownames = FALSE)
```



Statistical analysis

When you are done visualizing your data, you may be wondering whether the differences you see between experimental conditions (in our example different species) are observed because of chance or because of an underlying biological reason.
This section will present some statistical tests typically used in (plant) biology.

T-Test

In the next example, we will test whether the mean length of sepals in Versicolor species is truly different from that of Virginica.
In a T-test the so-called null hypothesis (H0) is that the mean of the two conditions (in our case of sepal length in the two species) is the same, while the so-called alternative hypothesis is that the mean is different between the two conditions. A *P*-value below 0.05 means that if we reject the null hypothesis there is a 5% chance that we are wrong.

```
# Perform the t-test for different levels of the categorical variable
Species
t_test <- t.test(Sepal.Length ~ Species, data = df,
                         Species %in% c("versicolor", "virginica"))
print(t_test) # p-value = 1.866e-07
```

In this example, the p-value is very small, which means that the mean sepal length of Versicolor flowers is statistically different from that of Virginica flowers.

Correlation test

If you want to determine the strength and relationship of two variables, you can use a correlation test to show the statistical association or correlation between the variables.
In R, you can do a correlation test with the function `cor.test`, which gives you a correlation coefficient, p-value and confidence intervals

```
# Perform the correlation test
cor_test <- cor.test(df$Sepal.Length, df$Petal.Length)

# Print the correlation value, p-values and confidence interval
print(cor_test)
```

```
> cor_test<-cor.test(iris$Sepal.Length, iris$Petal.Length)
> print(cor_test)

        Pearson's product-moment correlation

data:  iris$Sepal.Length and iris$Petal.Length
t = 21.646, df = 148, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.8270363 0.9055080
sample estimates:
      cor
0.8717538
```

The results from this correlation test show that there is a strong positive correlation (0.87) between sepal length and petal length, with a significant p-value (2.2e-16).

ANOVA

An Analysis of Variance (ANOVA) is commonly used to compare the means of different groups or treatments. ANOVA is helpful to determine if there are any statistically significant differences between the means of groups. However, its limitation is that you cannot distinguish which groups differ from each other.

```
# Perform an analysis of variance (ANOVA)
anova_analysis <- aov(Petal.Length ~ Species, data = df)
```

```
# Summarize the ANOVA results
summary(anova_analysis)
```

```
> summary(anova_analysis)
             Df Sum Sq Mean Sq F value Pr(>F)
Species       2  437.1  218.55    1180 <2e-16 ***
Residuals   147   27.2    0.19
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The results of the ANOVA analysis indicate that there are variations in petal lengths across the three species of irises. In other words, the mean petal lengths are not the same for all the species. However, we cannot determine exactly how the petal lengths differ between each species or which species exhibits a greater difference in petal length compared to the others.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a powerful statistical approach used to find out the main axes of variance within a dataset and identify the key variables in the dataset. PCA is also used to detect outliers in a dataset.

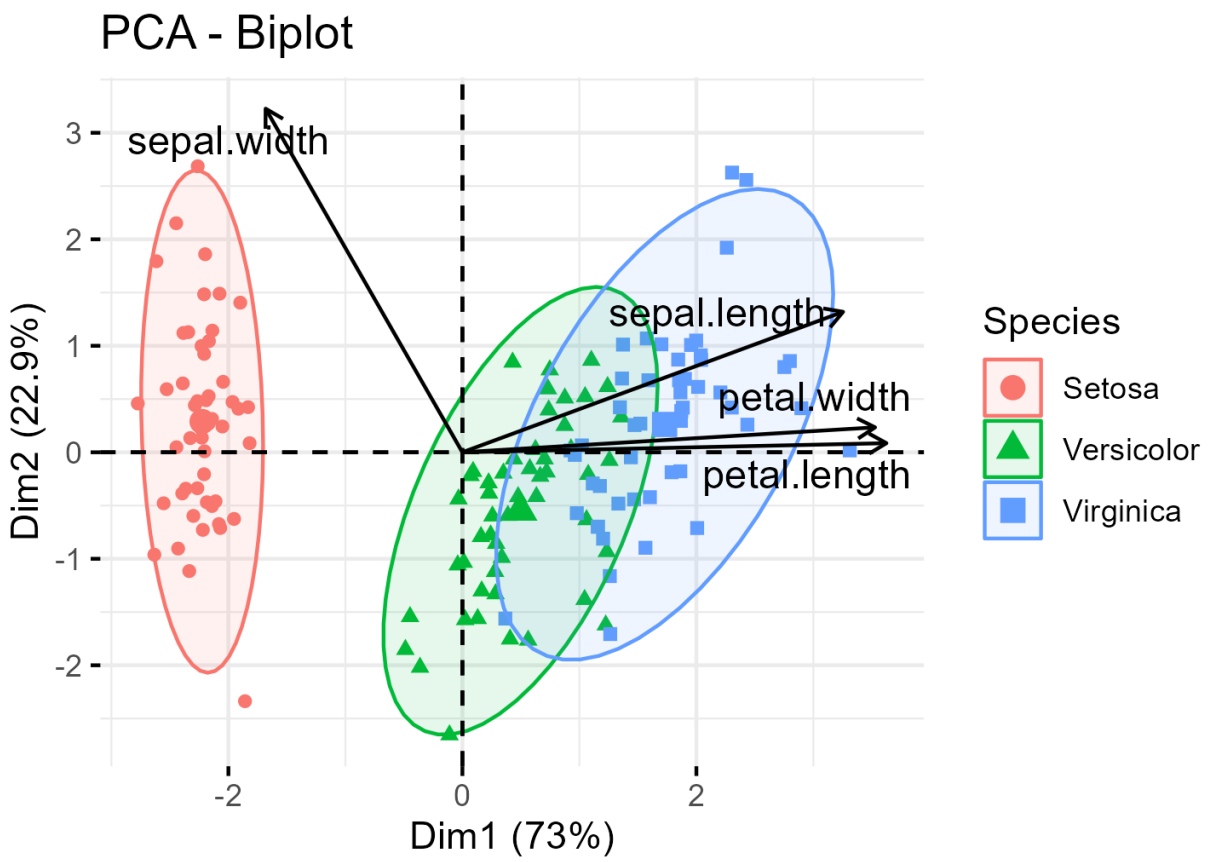
```
# Install and load packages for PCA
install.packages("FactoMineR") # for PCA analysis
install.packages("factoextra") # for PCA visualization

library(FactoMineR)
library(factoextra)

# Run PCA

pca <- PCA(df[1:4], graph = FALSE)


# Visualize PCA results
fviz_pca_biplot(pca,
                col.ind = df$variety,
                addEllipses = TRUE, label = "var",
                col.var = "black", repel = TRUE,
                legend.title = "Species")
```



Here, PCA biplot shows two main axes of variances of the dataset where the first axis explained 73% of the variance and the second axis explained 22.9% of the variance.

Additional useful examples

Reading different data files in R

Often scientific data is stored in different file formats. Here we discussed the different functions available in R to read various data files:

```
#Reading a csv file
read.csv("Path_to_your_file", sep=",", header=T)
# If the file doesn't have a header then add header=FALSE or F

#Reading a tsv file
read.csv("Path_to_your_file", sep=" ", header=T)

#Reading a table and creating a data frame
read.table("Path_to_your_file", sep=" ", header=T)
```

```
read.delim("Path_to_your_file", sep=" ", header=T)

#Reading an excel, xlsx,xls file
library(readxl)
read_excel("Path_to_your_file")
read_xlsx("Path_to_your_file")
read_xls("Path_to_your_file")
```

Export the data frame as an output

Once you get your expected output, here is the command to use to export the data frame in different formats:

```
write.csv(dataframe, "Path_to_your_output_file.csv") #export as csv
write.table(dataframe, "Path_to_your_output_file") #export as a table with an extension of choice
```

Tips and tricks

Last but not least, we wanted to provide some general tips and tricks (and what to avoid)

Figures

1. Before you make any figures, just like before you start planning an experiment, ask yourself: what are you trying to test? What piece of information are you trying to highlight in your data?
2. Tip that is always valid in programming: Google is your friend!
3. Avoid cramming your figures with information. If your figure looks very crowded, it probably means that you can split it into multiple figures.
4. Use color-blind-friendly palettes to increase the accessibility of your figures to a broader audience. A useful tool to find color-blind-friendly colors is colorbrewer

Inspiration

If you are unsure of what statistical analyses you need to run on your data, you can use resources such as this one to help you.

Take inspiration from publications in your field to represent your data. You can also use websites such as the R graph gallery to see examples of how different discrete and continuous data can be visualized

Feedback

Ask for feedback! Show your figure to your peers and colleagues: what is their reaction?

More examples

Bonus: https://github.com/cxli233/FriendsDontLetFriends

Code availability

Code available at: https://github.com/TurchiL/Plantae_Decoding_code_R

Sources

https://www.r-project.org/
https://posit.co/download/rstudio-desktop/
https://github.com/cxli233/FriendsDontLetFriends
https://colorbrewer2.org/
https://r-graph-gallery.com/index.html
https://stats.oarc.ucla.edu/other/mult-pkg/whatstat/