

R crash course

(Quick intro for high school students. 2024)

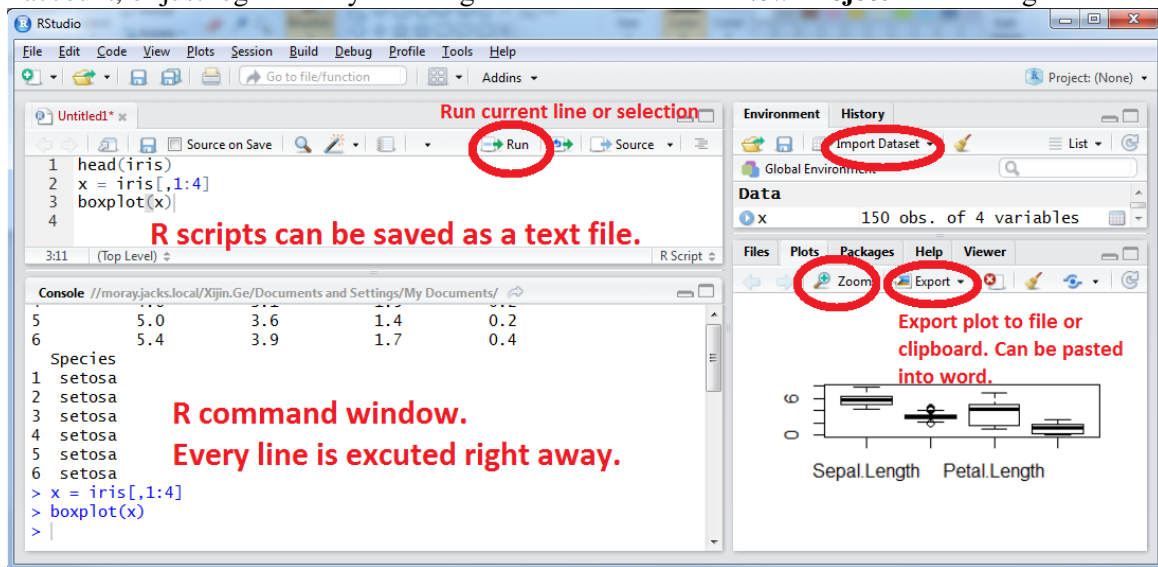
[Dr. Xijin Ge](#), Dept. of Mathematics & Statistics, South Dakota State University

Based on a free book: [Learn R through Examples](https://gexijin.github.io/learnR/) (gexijin.github.io/learnR/)

Coding: give precise commands to stupid but fast machines

Getting started

You can use R and RStudio through a web browser via www.Posit.Cloud website. Sign up to create a free account, or just log-in with your Google account. Click the “New Project” button to get started.



Let the magic begin!

```
head(iris)
str(iris)
summary(iris)
x = iris[, 1:4]
head(x)
boxplot(x)
plot(x[, 3:4])
abline(lm(x[, 4] ~ x[, 3]))
pairs(x)
stars(x)

PL = x$Petal.Length
PL
barplot(PL)
hist(PL)
SP = iris$Species
SP
pie(table(SP))
boxplot(PL ~ SP)
summary(aov(PL ~ SP))
```

Just try all these commands and guess what's going on. If it takes a few months to type these 188 characters, try www.RapidTyping.com.

Free book online: <https://gexijin.github.io/learnR/>



R data types and basic expressions¹

Common **data structures** in R include **scalars**, **vectors**, **factors**, **matrices**, **data frames**, and **lists**. These data structures can contain one or more data elements, namely **numeric** (2.5), **character** ("Jacks"), or **logical** (TRUE or FALSE).

Part 1

1. Expressions

Type anything at the prompt, and R will evaluate it and print the answer.

```
> 1 + 1
```

```
[1] 2
```

There's your result, 2. It's printed on the console right after your entry.

2. Type the string "Go Jacks". (Don't forget the quotes!)

```
> "Go Jacks"
```

```
[1] "Go Jacks"
```

3. **Challenge 1:** Now try multiplying 45.6 by 78.9

4. Logical Values

Some expressions return a "logical value": **TRUE** or **FALSE**. Many programming languages refer to these as "Boolean" values. Let's try typing an expression that gives us a logical value:

```
> 3 < 4
```

```
[1] TRUE
```

5. And another logical value (note that you need a double-equals sign to check whether two values are equal - a single-equals sign won't work):

```
> 2^3 == 5
```

```
[1] FALSE
```

6. Variables

As in other programming languages, you can store a value into a variable to access it later. Type `x = 42` to store a value in `x`. `x` is a **scalar**, containing only one data element.

```
> x = 42
```

```
[1] 42
```

You can also use the following. This is a conventional, safer way to assign values.

```
> x <- 42
```

7. `x` can now be used in expressions in place of the original result. Try dividing `x` by 2 (`/` is the division operator).

```
> x / 2
```

```
[1] 21
```

8. You can re-assign any value to a variable at any time. Try assigning "Go Jacks!" to `x`.

```
> y = "Go Jacks!"
```

9. You can print the value of a variable at any time just by typing its name in the console. Try printing the current value of `x`.

```
> y
```


```
[1] "Go Jacks!"
```

10. Now try assigning the **TRUE** logical value to `x`.

¹ This material is based on <http://tryr.codeschool.com/>

```
> z = TRUE
```

These variables are now created in the memory, shown in the **upper right** in the RStudio interface under **Global Environment**. You can store multiple values in a variable. That is called a **vector**, which is explained below. An object can also contain a table with rows and columns, like an Excel spreadsheet, as a **matrix**, or **data frame**.

I recommend that you start a R script file by clicking  and select “R script”. And then type your commands in the R script file. The commands can be run line by line, by placing the cursor in the line and click on the “Run” button. You can also select several lines to run them. Remember to save these R scripts that took so much time to write.

11. Functions

You call a function by typing its name, followed by one or more arguments to that function in parenthesis. Most of your R commands are functional calls. Let's try using the `sum` function, to add up a few numbers. Enter:

```
> sum(1, 3, 5)
[1] 9
```

12. Some arguments have names. For example, to repeat a value 3 times, you would call the `rep` function and provide its `times` argument:

```
> rep("Yo ho!", times = 30)
[1] "Yo ho!" "Yo ho!" "Yo ho!"
```

13. Looking for Help and Example Code

```
> ? sum
```

A web page will pop up. This is the official help information for this function. At the bottom of the page is some example code. The quickest way to learn an R function is to run the example codes and see the input and output. You can easily copy, paste, and twist the example code to do your analysis.

14. `example()` brings up examples of usage for the given function. Try displaying examples for the `boxplot` function:

```
> example(boxplot) # bring example of boxplot
```

Example commands and plots will show up automatically by typing Return in RStudio.

In R, you need to click on the plots.

15. **Challenge 2:** When you read/write data files without specifying paths, R uses the default **working directory**. Try to find and run the function that returns the current working directory.

16. **Challenge 3:** Try to find and run the function that lists all the files in the current folder.

17. It is important to add comments to your code. Everything after the “#” will be ignored by R when running. We often recycle and repurpose our codes.

```
> max(1, 3, 5) # return the maximum value of a set of numbers (vectors)
```

Part 2.

Motivating problem: Calculate $\sqrt{1} + \sqrt{2} + \dots + \sqrt{1000}$

1. Vectors

Once upon a time, Tom, Jerry, and Mickey went fishing and they caught 7, 3, and 9 fishes, respectively. This information can be stored in a **vector**, like this:

```
> c(7,3,9)
[1] 7 3 9
```

The `c` function (`c` is short for “combine”) creates a vector by combining 3 integers. If we want to continue to use the vector, we **hold it in an object and give it a name**:

```
> fish = c(7,3,9)
```

You will notice that “fishes” appear on the top right of the RStudio interface under Environment.

```
> fish
[1] 7 3 9
```

2. **fish** is a vector with 3 elements. There are many functions that operate on vectors. You can plot the vector:

```
> barplot(fish) # figure 1A
```

3. You can compute the total:

```
> sum(fish)
[1] 19
```

4. We can access the individual elements by indices, starting at 1:

```
> fish[3]
[1] 9
```

5. **Challenge 4:** Does Mickey caught more fishes than Tom and Jerry combined? Write R code to verify this statement using the **fish** vector and return a TRUE or FALSE value.

6. Jerry protested that the ¼ inch long fish he caught and released per fishing rules was not counted properly. We can change the values in the 2nd element directly by:

```
> fish[2] <- fish[2] + 1
```

On the right side, we take the current value of the 2nd element, which is 3, and add 1 to it. The result (4) **is assigned back to** the 2nd element itself. As a result, the 2nd element is increased by 1. We can also directly overwrite the values.

```
> fish[2] <- 4
```

```
> fish
```

```
Tom Jerry Mickey
7    4    9
```

7. They started a camp fire, and each ate 1 fish for dinner. Now the fishes left:

```
> fish2 <- fish - 1
> fish2
[1] 6 3 8
```

Most arithmetic operations work just as well on vectors as they do on single values. R subtracts 1 from each individual element. If you add a scalar (a single value) to a vector, the scalar will be added to each value in the vector, returning a new vector with the results.

8. While they were sleeping in their camping site, a fox stole 3 fishes from Jerry’s bucket, and 4 fishes from Mickey’s bucket. How many left?

```
> stolen = c(0, 3, 4) # a new vector
> fish2 - stolen
```

```
[1] 1 3 4
```

If you add or subtract two vectors of the same length, R will take the corresponding values from each vector and add or subtract them. The 0 is necessary to keep the vector length the same.

9. Proud of himself, Mickey wanted to make a 5ft x 5ft poster to show he is the best fisherman. Knowing that **a picture is worth a thousand words**, he learned R and started plotting. He absolutely needs his names on the plots. The data elements in a vector can have names or labels.

```
> names(fish) =  
c("Tom", "Jerry", "Mickey")
```

The right side is a vector, holding 3 character values. These values are assigned as the names of the 3 elements in the **fish** vector. **names** is a built-in function. Our vector looks like:

```
> fish  
Tom Jerry Mickey  
7 4 9
```

```
> barplot(fish) # see figure 1B
```

10. Assigning names for a vector also enables us to use labels to access each element. Try getting the value for Jerry:

```
> fish["Jerry"]  
Jerry  
4
```

11. Now see if you can set the value for the **Tom** to something other than 5 using the name rather than the index.

12. Tom proposes that their goal for next fishing trip is to double their catches.

```
> 2 * fish  
Tom Jerry Mickey  
14 8 18
```

Challenge 6: Create a vector representing the prices of groceries, bread \$2.5, milk \$3.1, jam \$5.3, beer \$9.1. And create a bar plot to represent this information.

13. Vectors cannot hold values with different modes (types). Try mixing modes and see what happens:

```
> c(1, TRUE, "three")  
[1] "1" "TRUE" "three"
```

All the values were converted to a single mode (characters) so that the vector can hold them all. To hold diverse types of values, you will need a **list**.

14. Sequence Vectors

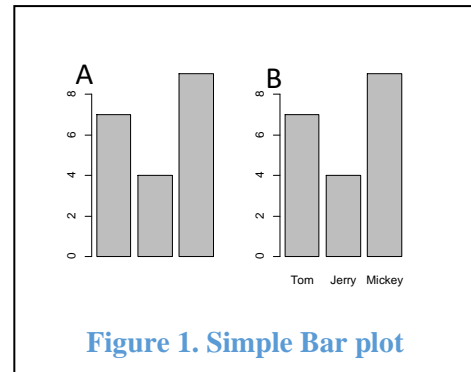
If you need a vector with a sequence of numbers you can create it with `start:end` notation. This is often used in loops and operations on the indices of vectors etc. Let's make a vector with values from 5 through 9:

```
> 5:9  
[1] 5 6 7 8 9
```

15. **Challenge 7:** Compute $1+2+3+\dots+1000$ with one line of R code. Hint: examine the example code for `sum()` function in the R help document.

16. **Challenge 8:** Compute $1+2+3+\dots+1000000$ with one line of R code.

17. **Challenge 9:** Compute $\sqrt{1} + \sqrt{2} + \dots + \sqrt{1000}$



18. Scatter Plots

The `plot` function takes two vectors, one for X values and one for Y values, and draws a graph of them.

```
> x = 1:100
> y = sqrt(x)
```

19. Then simply call `plot` with your two vectors:

```
> plot(x,y)
```

Great job! Notice on the graph that values from the first argument (`x`) are used for the horizontal axis, and values from the second (`y`) for the vertical.

20. Challenge 10: Create a vector with 21 integers from -10 to 10 and store it in vector `x`. Then create a scatterplot of x^2 against `x`.

Part 3

1. “Data frames” have rows and columns: The Iris flower dataset

In 1936, Dr. Edgar Anderson collected data to quantify the geographic variation of *Iris* flowers. The dataset consists of 50 samples from each of three sub-species (*Iris setosa*, *Iris virginica* and *Iris versicolor*). Four features were measured from each sample: the lengths and the widths of sepals and petals, in centimeters (cm). This data is included in R base software. Go to the Wikipedia page for this data set (yes, it is famous!). Have a quick look at the data there, think about what distinguishes the three species? If we have a flower with sepals of 6.5cm long and 3.0cm wide, petals of 6.2cm long, and 2.2cm wide, which species does it most likely belong to? Think (!) for a few minutes while eyeballing the data at Wikipedia.

To answer these questions, let’s visualize and analyze the data with R. Type these commands in **bold**, without the prompt “>” or the comments after “#”.

```
> iris           #This will print the whole dataset, which is included with R
> dim(iris)     # show the numbers of rows and columns
[1] 150 5
> head(iris)    # show the first few rows; useful for bigger datasets.
```



Figure 2. Iris flower. Photo from Wikipedia.

So, the first 4 columns contain numeric values. The last one is species information as **character** values. This is an important distinction, as we cannot add or subtract character values. This object is a **data frame**, with both numeric and character columns. A **matrix** only contains one type of values, often just numbers.

Individual values in a data frame can be accessed using row and column indices.

```
> iris[3, 4]    # shows the value in 3rd row, 4th column. It is 0.2.
> iris[3, ]     # shows all of row 3
> iris[, 4]     # shows all of column 4
> iris[3, 1:4]  # shows row 3, columns 1 to 4.
```

Challenge 1a: display data in rows 1 to 10 from columns 2 to 5.

```
> colnames(iris) # Column names.
```

```
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

Remember these column names, as we are going to use them in our analysis.

```
> iris$Petal.Length # entire column
```

R is case-sensitive. “petal.length” will not be recognized.

```
> iris$petal.length # does not work. Gives “NULL”
```

	Column indices				
	1	2	3	4	5
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5			
2	4.9	3.0			
3	4.7	3.2			setosa
4	4.6	3.1			setosa
5	5.0	3.6			setosa
6			1.7	0.4	setosa

Figure 3. Example of a data frame.

> `iris$petal_length` # does not work. Column name must match exactly

2. Analyzing one set of numbers (vector)

> `PL = iris$Petal.Length` # I am just lazy and don't want to type "iris\$Petal.Length", repeatedly.

> `PL` # it is a vector holding 150 numbers.

> `mean(PL)`

> `summary(PL)`

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
1.000 1.600 4.350 3.758 5.100 6.900
```

- The minimum petal length is 1.0, and the maximum is 6.9.
- **Average** petal length is 3.758.
- The mid-point or **median** is 5.35, as about half of the numbers is smaller than 5.35. Why the median is different from the mean?
- The 3rd quartile, or 75th percentile is 5.1, which is bigger than 75% of the numbers.
- The 1st quartile, or 25th percentile is 1.6.

These summary statistics are graphically represented as a boxplot in the Figure 1A. Boxplots are more useful when multiple sets of numbers are compared.

> `boxplot(PL)` # Figure 4A. It graphically represents the spread of the data.

> `boxplot(iris[, 1:4])` # boxplot of several columns at the same time Figure 4B.

In RStudio, you can copy a plot to clipboard using the **Export** button on top of the plot area. Or you can click **zoom**, right click on the popup plot and select "**Copy Image**". Then you can paste the plot into Word.

Challenge 2a: What can we tell from this boxplot (Figure 4B)? Summarize your observations in PLAIN English. Note the differences in median and the spread (tall boxes). What is the most variable feature?

To quantify the variance, we can compute the **standard deviation** σ :

$$\sigma = \sqrt{\frac{1}{N} [(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_N - \mu)^2]}, \text{ where } \mu = \frac{1}{N}(x_1 + \dots + x_N),$$

If all the measurements are close to the mean (μ), then standard deviation should be small.

> `sd(PL)` #sd() is a function for standard deviation

```
[1] 1.765298
```

> `sd(iris$Sepal.Width)`

```
[1] 0.4358663
```

As we can see, these flowers have similar sepal width. They differ widely in petal length. This is consistent with the boxplot above. Perhaps changes in petal length led to better survival in different habitats.

With R it is very easy to generate graphs.

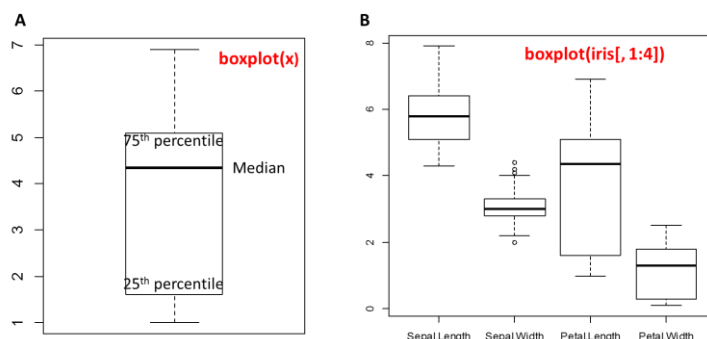


Figure 4. Boxplot of petal length (A) and of all 4 columns (B).

> **barplot(PL)**

> **plot(PL)** # Run sequence plot Figure 5

As we can see, the first 50 flowers (*Iris setosa*) have much shorter petals than the other two species. The last 50 flowers (*Iris virginica*) have slightly longer petal than the middle (*iris versicolor*).

> **hist(PL)** # histogram

Histogram shows the distribution of data. The histogram top right of Figure 5 shows that there are more flowers with Petal Length between 1 and 1.5. It also shows that the data does not show a bell-curved distribution.

> **qqnorm(PL)** #Q-Q plot for normal distribution

> **qqline(PL)** #add a line

Q-Q plot can help check if data follows a Gaussian distribution, which is widely observed in many situations. Also referred to as normal distribution, it is the pre-requisite for many statistical methods. See Figure 6 for an example of normal distribution. Quantiles of the data is compared against those in a normal distribution. If the normal Q-Q plot is close to the reference line produced by qqline(), then the data has a normal distribution.

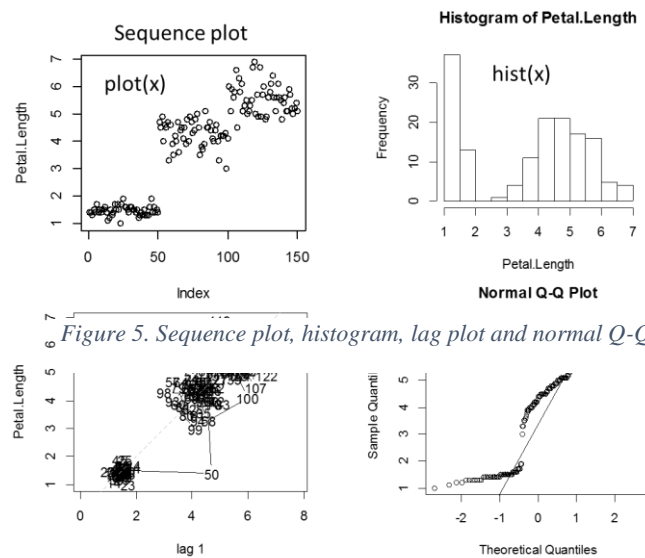


Figure 5. Sequence plot, histogram, lag plot and normal Q-Q plot.

Challenge 2b: Investigate Sepal Length distribution using these techniques and summarize your observations in PLAIN English. Hint: assign **x** with the values in the sepal length column (**SL = iris\$Sepal.Length**), and re-run all the code in this section.

3. Student's t-test

In hypothesis testing, we evaluate how likely the observed data can be generated if we assume a certain hypothesis is true. If this probability (P value) is very small (<0.05, typically), we reject that hypothesis.

Are the petal lengths of *iris setosa* significantly different from these of *iris versicolor*?

> **PL1= PL[1:50]** # the first 50 values of Sepal.Length are for *iris setosa*

> **PL2 = PL[51:100]** # the next 50 values of Sepal.Length are for *iris versicolor*

> **t.test(PL1, PL2)** # t.test() is an R function for student's t-test

Welch Two Sample t-test

data: x and y

t = -39.493, df = 62.14, **p-value < 2.2e-16**

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

-2.939618 -2.656382

sample estimates:

mean of x mean of y

1.462 4.260

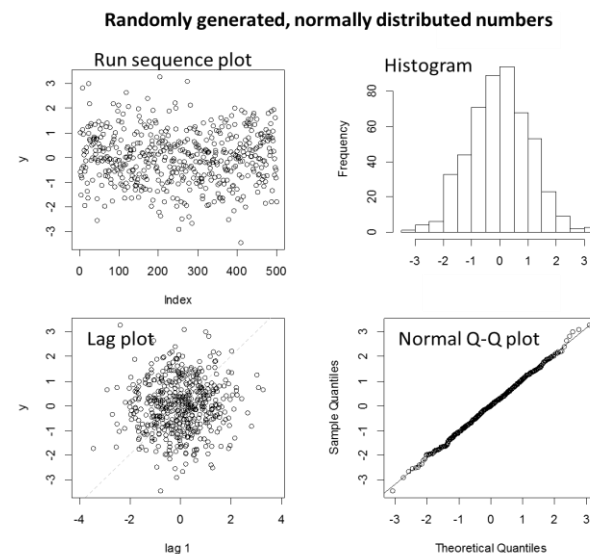


Figure 6. Plots for random generated numbers following a normal distribution. This is for reference.

In this student's t-test, our hypothesis is that the mean petal length is the same for *I. versicolor* and *I. virginica*. A small P-value of 2.2×10^{-16} indicates under this hypothesis, it is extremely unlikely to observe the difference of 1.292cm through random sampling. Hence, we reject that hypothesis and conclude that the true mean is different.

Challenge 3a: Are *iris setosa* and *iris versicolor* significantly different in sepal width? Hint: Replace PL = Petal.Length with something else and re-run above code.

We can also do t-test on one set of numbers. This is a one-sample T-test of mean:

```
> t.test(iris$Sepal.Length, mu=5.8)
```

One Sample t-test

data: Sepal.Length

t = 0.64092, df = 149, p-value = 0.5226

alternative hypothesis: true mean is not equal to 5.8

95 percent confidence interval:

5.709732 5.976934

sample estimates: mean of x: 5.843333

In this case, our hypothesis is that the true average of sepal length for all iris flowers is 5.8. Since P value is quite big, we accept this hypothesis. This function also tells us the 95% confidence interval on the mean. Based on our sample of 150 iris flowers, we are 95% confident that the true mean is between 5.71 and 5.98.

Challenge 3b: Compute 95% confidence interval of petal length for *iris setosa*.

4. Test for normal distribution

We can perform hypothesis testing on whether a set of numbers derived from normal distribution. The null hypothesis is that the data is from a normal distribution.

```
> shapiro.test(PL)
```

Shapiro-Wilk normality test

W = 0.87627, p-value = 7.412e-10

If petal length is normally distributed, there is only 7.412×10^{-10} chance of getting a test statistic of 0.87627, which is observed in our sample of 150 flowers. In other words, it is highly unlikely that petal length follows a normal distribution. We reject the normal distribution hypothesis.

Challenge 4a: Is sepal width normally distributed? Run Shapiro's test and also generate histogram and normal Q-Q plot.

5. Analyzing a column of categorical values

In the iris dataset, the last column contains the species information. These are "string" values or categorical values.

```
> counts = table(iris$Species) # tabulate the frequencies
```

```
> counts
```

setosa	versicolor	virginica
50	50	50

```
> pie(counts) #See Figure 7 7A.
```

Pie charts are very effective in showing proportions.

```
> barplot(counts) #See Figure 7B
```

We can see that the three species are each represented with 50 observations.

Part 4

6. Analyzing the relationship between two columns of numbers

Scatter plot is a very effective in visualizing correlation between two columns of numbers.

```
> PL = iris$Petal.Length
```

```
> PW = iris$Petal.Width
```

```
> SP = iris$Species
```

```
> plot(PW, PL) # scatterplot, refined version in Figure 8
```

Figure 8 shows that there is a positive correlation between petal length and petal width. In other words, flowers with longer petals are often wider. So, the petals are getting bigger substantially, when both dimensions increase.

Another unusual feature is that there seems to be two clusters of points. Do the points in the small cluster represent one particular species of Iris? We need to further investigate this. The following will produce a plot with the species information color-coded. The resultant Fig.2 clearly show that indeed one species, *I. setosa* constitutes the smaller cluster in the low left. The other two species also show difference in this plot, even though they are not easily separated. This is a very important insight into this dataset.

```
> plot(PW, PL, col=rainbow(3)[SP]) # change colors based on another column (Species).
```

```
> legend("topleft", levels(SP), fill=rainbow(3)) # add legends on topleft.
```

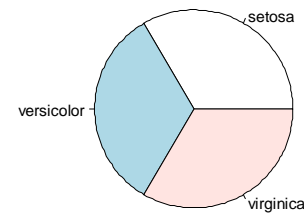
The rainbow() function generates 3 colors and Species information is used to choose colors. Note that Species column is a **factor**, which is a good way to encode columns with multiple levels. Internally, it is coded as 1, 2, 3.

```
> str(iris) # show the structure of data object
```

```
'data.frame':    150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1
 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5
 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1
 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...
 1 1 1 1 1 1 1 1 1 ...
```

Perhaps due to adaption to environment, change in petal length lead to better survival. With the smallest petals, *Iris Setosa* is found in Arctic regions. *Iris versicolor* is often found in the Eastern United States and Eastern Canada. *Iris virginica* “is common along the coastal plain from Florida to Georgia in the Southeastern United States [Wikipedia].” It appears the iris flowers in warmer places are much larger than those in colder ones. With R, it is very easy to generate lots of graphics. But we still have to do the thinking. It requires put the plots in context.

A



B

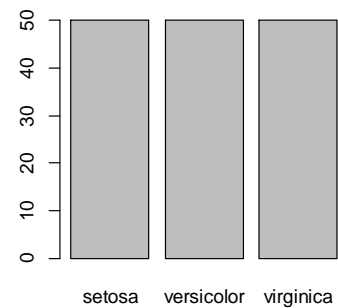


Figure 7. Frequencies of categorical values visualized by Pie chart (A) and bar chart (B).

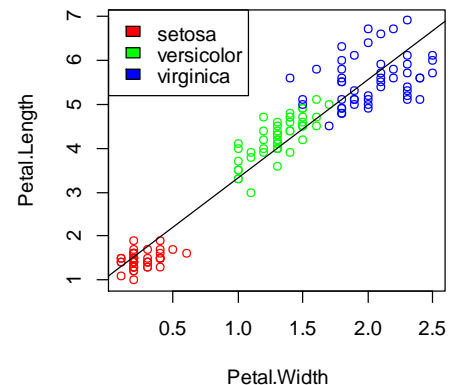


Figure 8. Scatter plot shows the correlation of petal width and petal length.

Quantitative analyses of correlation

We can quantitatively characterize the strength of the correlation using several types of correlation coefficients, such as Pearson's correlation coefficient, r . It ranges from -1 to 1.

```
> cor(PW, PL)
```

```
[1] 0.9628654
```

This means the petal width and petal length are strongly and positively correlated.

```
> cor.test(PW, PL)
```

Pearson's product-moment correlation

data: Petal.Width and Petal.Length

$t = 43.387$, $df = 148$, **p-value < 2.2e-16**

alternative hypothesis: true correlation is not equal to 0

95 percent confidence interval:

0.9490525 0.9729853

sample estimates: cor 0.9628654

Through **hypothesis testing** of the correlation, we reject the null hypothesis that the true correlation is zero. That means the correlation is statistically significant.

Note that Pearson's correlation coefficient is not robust against outliers and other methods such as Spearman's exists. See help info:

```
> ? cor #show help info on cor ()
```

We can also determine the equation that links petal width and petal width. This is so called regression analysis. We assume

$$\text{Petal.Length} = \alpha \times \text{Petal.Width} + c + e,$$

where α is the slope parameter, c is a constant, and e is some random error. This linear model can be determined by a method that minimizes the least squared-error:

```
> model = lm(PL ~ PW) # Linear Model (lm): petal length as a function of petal width
```

```
> summary(model) # shows the details
```

Call: `lm(formula = Petal.Length ~ Petal.Width)`

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.08356	0.07297	14.85	<2e-16 ***
Petal.Width	2.22994	0.05140	43.39	<2e-16 ***

As we can see, we estimated that $\alpha=2.22944$ and $c=1.08356$. Both parameters are significantly different from zero as the P values are $<2 \times 10^{-16}$ in both cases. In other words, we can reliably predict

$\text{Petal.Length} = 2.22944 \times \text{Petal.Width} + 1.08356$. This model can be put on the scatter plot as a line.

```
> abline(model) # add regression line to existing scatter plot.
```

```
> text(1.5, 2, "R=0.96") # add annotation at position x=1.5, y= 2.
```

Sometimes, we use this type of regression analysis to investigate whether variables are correlated.

Challenge 6b: Investigate the relationship between sepal length and sepal width using scatter plots, correlation coefficients, test of correlation, and linear regression. Again, interpret all your results in PLAIN and proper English.

Scatter plot matrix

We can generate a matrix of scatter plots simply by:

```
> pairs(iris[, 1:4])
```

```
> pairs(iris[, 1:4], col=rainbow(3)[ SP] ) #Figure 9
```

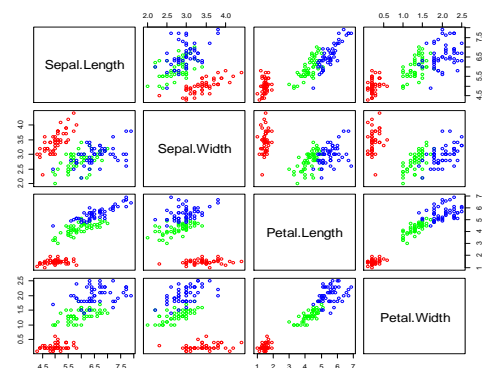


Figure 9. Scatter plot matrix.

Challenge 6c: Look at this large plot for a moment. Provide interpretation of these scatter plots.

7. Test the difference among multiple groups (Analysis of Variance: ANOVA)

As indicated by Fig 10, sepal width has small variation, even across 3 species. We want to know if the mean sepal width is the same across 3 species. This is done through Analysis of Variance (ANOVA).

```
> SW = iris$Sepal.Width
> boxplot(SW ~ SP)      #Figure 10
> result = aov(SW ~ SP) # ANOVA
> summary(result)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Species	2	11.35	5.672	49.16	<2e-16 ***
Residuals	147	16.96	0.115		

Since P-value is much smaller than 0.05, we reject the null hypothesis. The mean sepal width is not the same for 3 species. This is the only thing we can conclude from this. The boxplot in Figure 10 seems to indicate that *I. Setosa* has wider sepals.

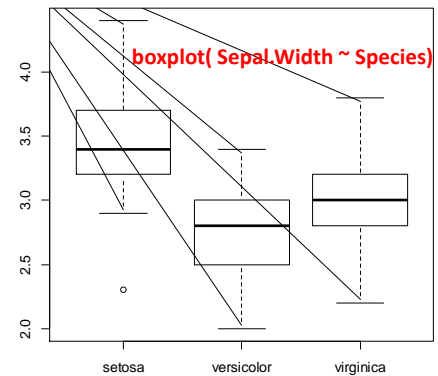


Figure 10. Boxplot of sepal width across 3 species.

8. Create plots step by step using ggplot2. [Chapter 2.4](#)

```
install.packages("ggplot2")
library(ggplot2) # load the ggplot2 package
```

```
install.packages("ggplot2")
library(ggplot2) # load the ggplot2 package
```

```
# map data to x and y coordinates
ggplot(data = iris) +
  aes(x = Petal.Length, y = Petal.Width)
```

```
# add data points
ggplot(data = iris) +
  aes(x = Petal.Length, y = Petal.Width) +
  geom_point()
```

```
# change color & symbol type
ggplot(data = iris) +
  aes(x = Petal.Length, y = Petal.Width) +
  geom_point(aes(color = Species, shape = Species))
```

```
# add trend line
ggplot(data = iris) +
  aes(x = Petal.Length, y = Petal.Width) +
  geom_point(aes(color = Species, shape = Species)) +
  geom_smooth(method = lm)
```

9. Interactivity with plotly

```
# store plot in an object
p <- ggplot(data = iris) +
  aes(x = Petal.Length, y = Petal.Width) +
  geom_point(aes(color = Species, shape = Species)) +
  geom_smooth(method = lm)
```

```
install.packages("plotly")
```

```
library(plotly)
ggplotly(p)
```

10. It's insanely easy to create Shiny web apps. [Chapter 7](#)

- Select **File -> New File -> Shiny Web App** from the RStudio main menu.
- Click on **Run App** on the top-right of the script window
- Change the color to "red".
- Change title to "Iris data"
- Change data to iris[, "Sepal.Length"]
- Add a control widget to select columns
selectInput("cid", "Column", choices = colnames(iris)),
- Use the widget to swap data.
- Grand challenge. If species is selected, show a pie chart instead.

11. [Project workflow](#)

- Find and download data
- Examine in Excel
- Upload to Posit.cloud
- Read the file
- Clean the data
- Exploratory analysis (distributions, correlations, etc)
- Ask a question. Write code. Try and refine iteratively.
- Summarize and present

Free R textbook: Learn R through Examples

<https://gexijin.github.io/learnR/>

R Cheat sheets:

<https://posit.co/resources/cheatsheets/>