An introduction to RStudio

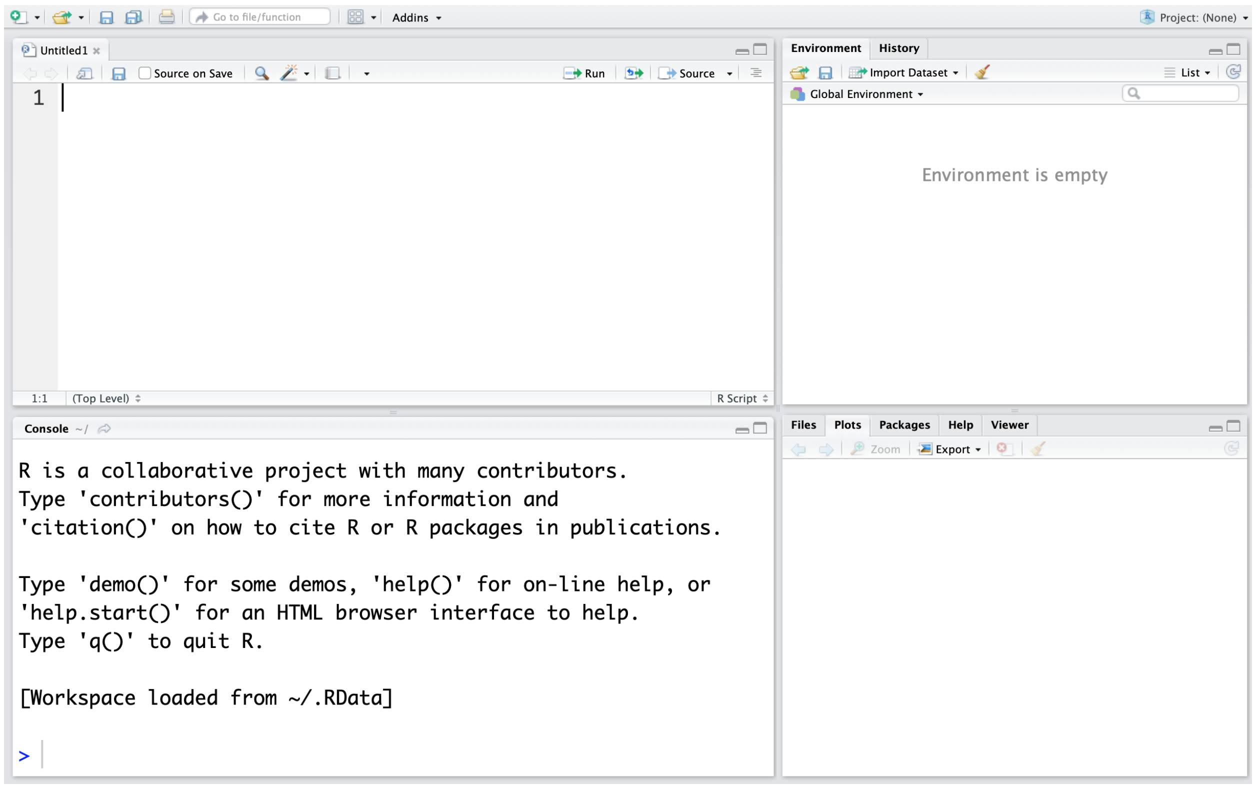
Getting started

The editor in `R` is quite basic and has few limited functions. `RStudio` can provide a more user-friendly coding environment. `RStudio` is an integrated development environment (IDE) for `R`. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

I recommend you to use `RStudio` to edit and execute `R` codes, because it is more user-friendly than `R`. However, please feel free to use `R` if you are more comfortable with `R` interface.

To use `RStudio`, you have to install both `R` (https://cran.rstudio.com) and `RStudio` (https://www.rstudio.com/products/rstudio/download/#download) in your laptop. The free `RStudio` desktop is enough for you.

The left-hand-side panel is the Console, where you can type in commands and see the outputs. The top-right panel can display your current work environment, e.g. the data frames, variables or function outputs, as well as your work history, i.e. the commands you executed so far. The bottom-right panel can show you plots or help information.

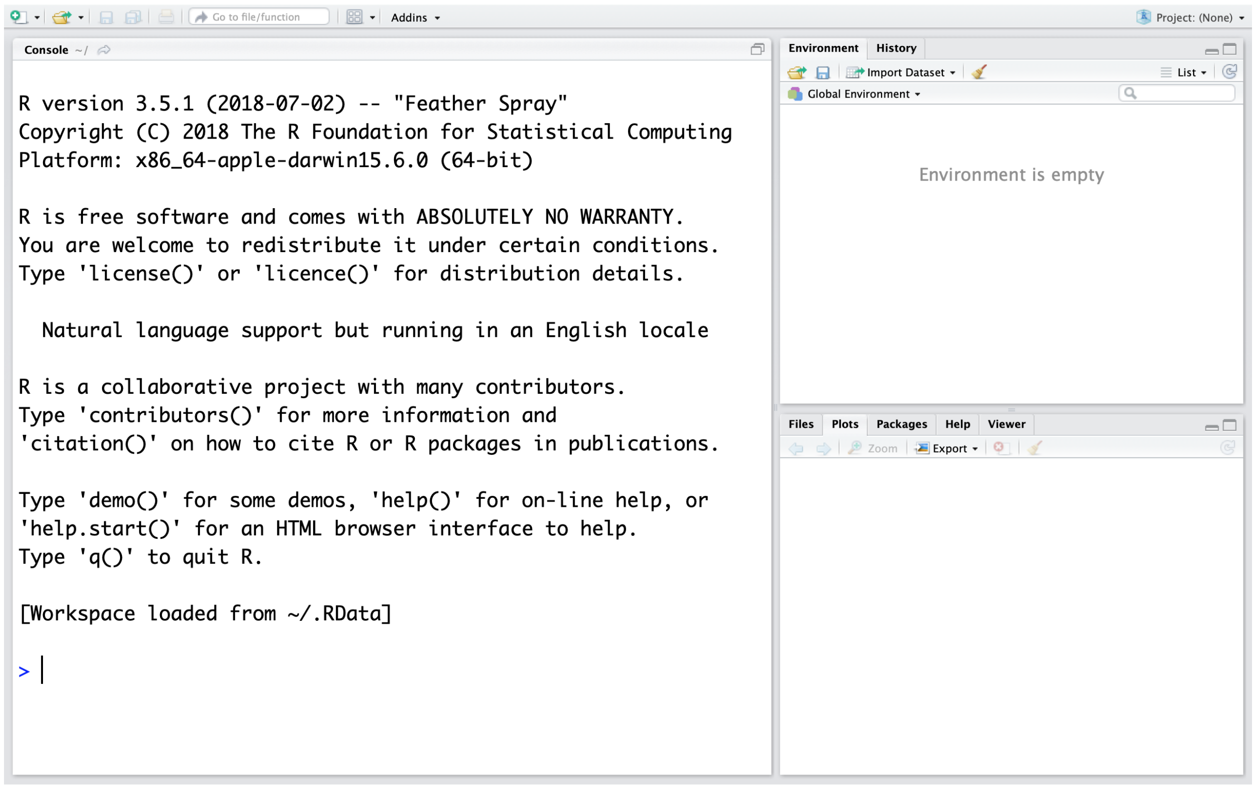


```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

To use the `RStudio` editor, click the green plus button on the top-left conner and choose `R` Script, or click File ⇒ New File ⇒ `R` Script. Then you can find a new window of `R` script editor on the top-left panel.

R script

Instead of typing commands in the Console directly, we can write down the commands in the `R` script editor. An `R` script file contains `R` commands and has extension `.r` or `.R`.



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> 
```

Figure 1: `RStudio` screen.

To execute the commands in the `R` script, highlight the lines you'd like to execute and click the `Run` button with a right green arrow on the top of the editor window. If you just want to execute one line, you can also put your mouse on the line and click `Run`. Then you can see the outputs in the Console window.

The advantage of typing commands in the editor is that we can save the `R` script and use it later if we'd like to repeat the commands or edit the script. We usually create an `R` script for a specific task, e.g. fit a linear model for a dataset (with commands to load data, fit a model and draw plots etc.), so we could repeat/modify the task in the future.

To save the `R` script, click File ⇒ Save As and choose the file folder where you'd like to save the script.

Set working directory

`R` looks for files in its current working directory if we don't specify the complete route of the file. In `RStudio`, click Session ⇒ Set Working Directory ⇒ Choose Directory, and choose a file folder. `R` will look for files or save files in this file folder as default if we don't explicitly specify another directory. For example, if you'd like to save the `R` script, click File ⇒ Save, `R` will save the script to your current working directory.

I recommend you to do this as your first step after opening `RStudio`. You can put the datasets and related `R` scripts in the same file folder and set this file folder as your working directory. In this way, you can read data and save files or plots easily without specifying complete routes.

Variables, vectors, factors, matrices and lists

Variables

We can store numerical (or other) data in {variables}, using `<-` in `R`, or simply `=` in `RStudio`. If you want to store something in a variable, please make sure that there is no space in `<-`: `x <- 10` means we store 10 in `x`, however, `x < - 10` is a logical command which means whether `x` is less than -10.

```
x <- 10
x
```

```
## [1] 10
```

```
x < - 10
```

```
## [1] FALSE
```

```
x
```

```
## [1] 10
```

```
x=10
x
```

```
## [1] 10
```

Vectors

We can create vectors by using `c()`.

```
balance = c(1000,1375,345,2867,12)
Income =c(25480,27648,48735,19035)
```

Some calculations based on vectors:

```
Income/1000
```

```
## [1] 25.480 27.648 48.735 19.035
```

```
log(Income)
```

```
## [1] 10.145649 10.227309 10.794153  9.854035
```

```
mean(balance)
```

```
## [1] 1119.8
```

```
sd(Income)
```

```
## [1] 12871.05
```

```
summary(balance)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      12     345    1000    1120    1375    2867
```

```
Income>20000
```

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

`>` in `Income>20000` is a logical operator, which produces outputs of `TRUE` or `FALSE`. Try other logical operators `<`, `>=`, `==`, `!=`, etc.

Note that `==` is different from `=`.

```
x=5
x==2
```

```
## [1] FALSE
```

```
x=2
x
```

```
## [1] 2
```

To find how many entries in a vector satisfying a criterion, we can use logical operators:

```
set.seed(12)
x=rnorm(30)
sum(x<0.1)
```

```
## [1] 19
```

```
x=c(0,0,1,1,0)
y=c(1,0,1,0,0)
sum(x!=y)
```

```
## [1] 2
```

```
sum(x==y)
```

```
## [1] 3
```

Think about why do we obtain 19, 2 and 3 in the above codes.

If you'd like to know what a function/command does, e.g. `sum`, type

```
?sum
help(sum)
```

and you can find the help information of `sum` in the bottom-right panel.

Factors

We can store qualitative/categorical values in factors. To create a factor, we can first create a character vector and then convert it to a factor using `factor`.

```
student=rep(c("No","Yes"),c(3,2))
student
```

```
## [1] "No"  "No"  "No"  "Yes" "Yes"
```

```
student_fact=factor(student)
student_fact
```

```
## [1] No  No  No  Yes Yes
## Levels: No Yes
```

`student` is a character vector, which stores character values indicating whether one observation is a student or not. However, `student_fact` is a factor that has two levels `No` and `Yes`.

Type `?rep` to see how to use `rep`.

Use `levels` to see the levels in a factor

```
levels(student_fact)
```

```
## [1] "No"  "Yes"
```

Matrices

We can create a matrix using `matrix`.

```r
x = matrix(c(1,2,8,3,4,7,10,8,5,7,4,1), nrow=4, ncol=3)
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    5
## [2,]    2    7    7
## [3,]    8   10    4
## [4,]    3    8    1
```

```r
?matrix
```

Type `?matrix` to see why we have this matrix with the inputs.

Matrix transpose `t()`:

```r
t(x)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    8    3
## [2,]    4    7   10    8
## [3,]    5    7    4    1
```

To access a row, a column or an element of a matrix:

```r
x[1,]
```

```
## [1] 1 4 5
```

```r
x[,2]
```

```
## [1]  4  7 10  8
```

```r
x[3,2]
```

```
## [1] 10
```

Matrix multiplication `%*%`:

```r
y=c(1,3,0,2)
t(x)%*%y   #matrix multiplication
```

```
##      [,1]
## [1,]   13
## [2,]   41
## [3,]   28
```

```r
x*y  #element-wise multiplication
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    5
## [2,]    6   21   21
## [3,]    0    0    0
## [4,]    6   16    2
```

Lists

Lists can be used to store data with different types. To take out elements from a list, we can use `$name` or `[[]]`.

```
Income=c(20395,34755,27585,16745,18325)
list1=list(Student=student_fact,Income=Income)
list1$Student
```

```
## [1] No  No  No  Yes Yes
## Levels: No Yes
```

```
list1$Income
```

```
## [1] 20395 34755 27585 16745 18325
```

```
list1[[1]]
```

```
## [1] No  No  No  Yes Yes
## Levels: No Yes
```

```
list1[[2]]
```

```
## [1] 20395 34755 27585 16745 18325
```

Some functions output their results in lists. Here is an example of linear regression `lm()`.

```
data(mtcars)
fit=lm(mpg~cyl,data=mtcars)
names(fit)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
fit$coefficients
```

```
## (Intercept)         cyl
##    37.88458    -2.87579
```

```
fit$df.residual
```

```
## [1] 30
```

```
fit[[1]]
```

```
## (Intercept)         cyl
##    37.88458    -2.87579
```

Import data

Import data

Download `Advertising.csv` and `Default.txt` from the Moodle page and save them in the same file folder. Set this file folder as your working directory.

Read `.csv` data `read.csv`:

```
Advertising=read.csv("Advertising.csv",header=TRUE)
Default=read.table("Default.txt",header=TRUE)
```

```
?read.csv
?read.table
```

Type `?read.csv` and `?read.table` to see the meaning of `header=TRUE` and other input options.

Data frames

The above two datasets have a two-way structure: columns store variables/features and rows store objects/instances. In `R`, these datasets are usually stored as data frames. A data frame is similar to a matrix: both of them have a two-way structure. However, a data frame can store different types of data (similarly to a list).

We can view the data frames by typing their names in Console:

```
Advertising
```

To access the variables, objects or elements in a data frame:

```
second_object=Default[2,]
first_variable=Default[,1]
first_variabe=Default$default
Default[2,3]
```

```
## [1] 817.1804
```

```
Default$default[3]
```

```
## [1] No
## Levels: No Yes
```

```
Default$balance[5]
```

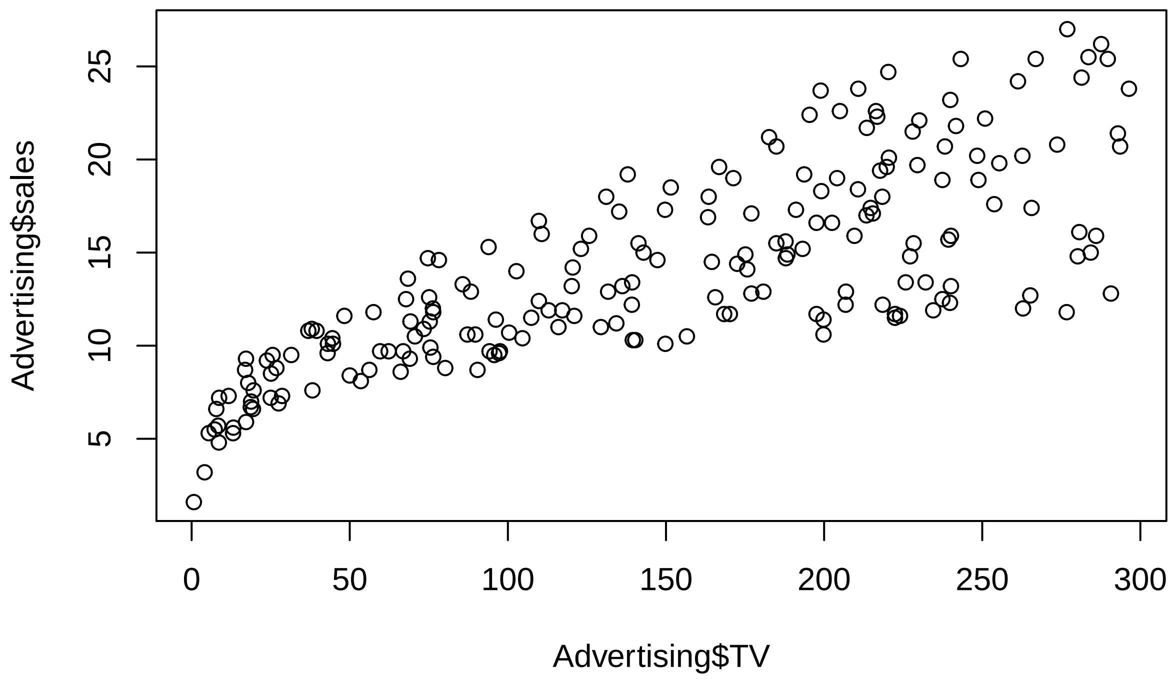
```
## [1] 785.6559
```

Plots

Draw a scatter plot

We can draw scatter plots by using `plot()`

```
plot(Advertising$TV,Advertising$sales)
```

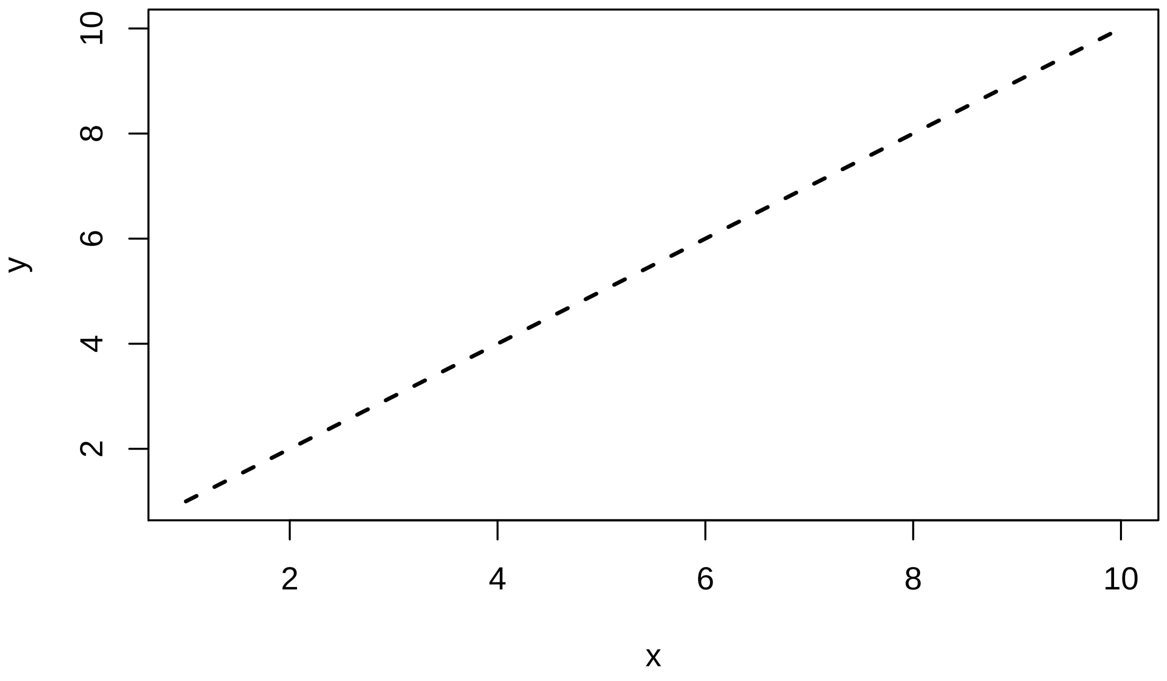


Now you can see a plot in the plot window. Tpye `?plot` to explore options that can change the looking of your plot, e.g. `col`, `pch`, ...

To save the plot, simply click the `Export` button on the plot window.

Draw a line

```
# Generate a sequence of values, from 1 to 10 with step size 0.01
x=seq(1,10,by=0.01)
y=x
plot(x,y,type="l",lty=2,lwd=2)
```



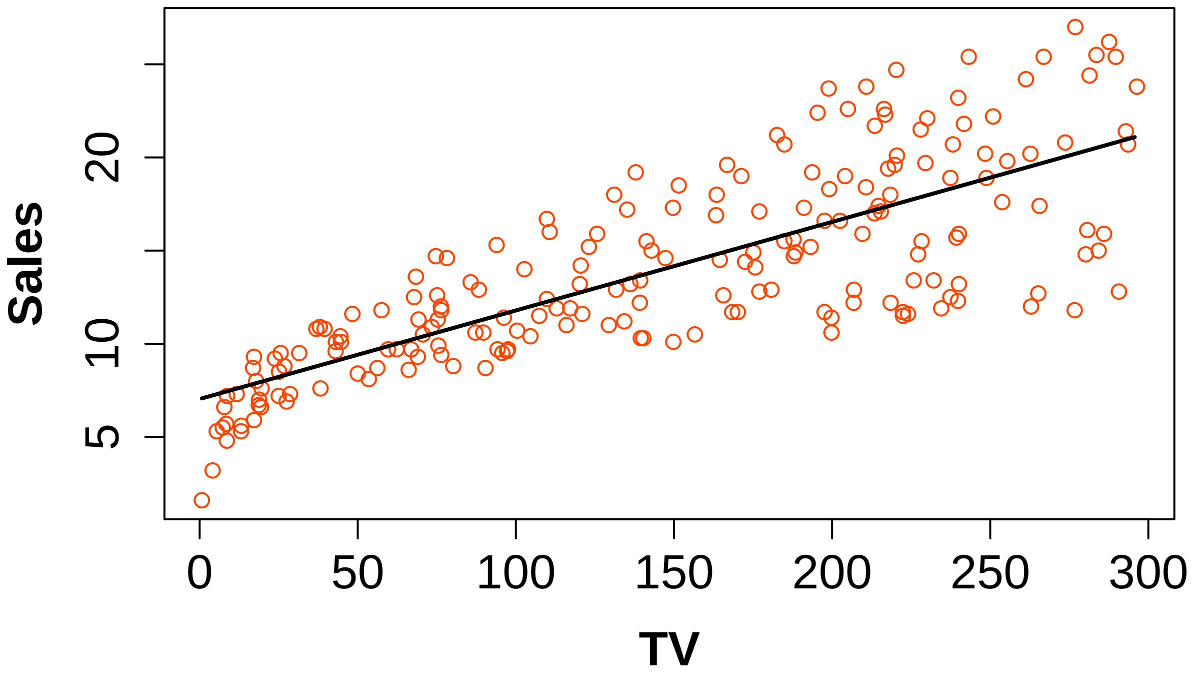
Use

help to understand the meaning of `lty` and `lwd`. Explore options to change the line type, colour or width.

Add a regression line to a scatter plot

Open a new `R` script in the editor window and copy paste the following commands in it. Save it as `advertising-plot.R` in your current working directory.

```
# Read the Advertising data and call the data frame Advertising.
Advertising=read.csv("Advertising.csv",header=TRUE)
# Fit a simple linear regression line, sales~TV.
fit1=lm(sales~TV,data=Advertising)
# Get the variables, sales and TV, from the data frame
sales=Advertising$sales; TV=Advertising$TV;
# Generate new predictors, i.e. new TV values, for the regression line.
TVlims =range(TV)
TV.grid=seq (from=TVlims [1], to=TVlims [2])
# Predict new sales values based on the fitted model and new TV values.
preds=predict(fit1 ,newdata =list(TV=TV.grid))
# Scatter plot of sales against TV.
plot(TV ,sales ,xlim=TVlims ,cex.lab=1.5,
     cex.axis=1.5,  font.lab=2 ,col ="orangered",xlab="TV",ylab="Sales")
# Add the regression line using new TV values and predicted sales values
# to the current plot.
lines(TV.grid ,preds ,lwd =2, col ="black")
```

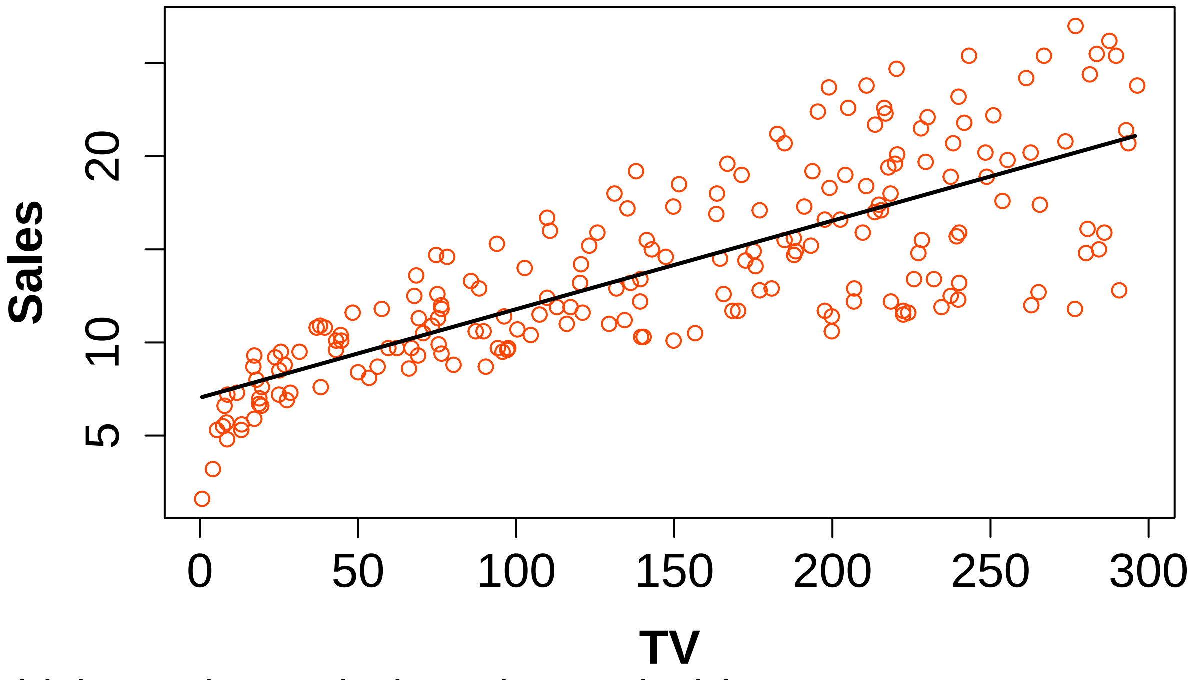


Use help to understand everything in this script.

# starts a comment line. You have to include sufficient comments in your scripts, to make sure that your codes are readable to someone else. It is also good for yourself because after a long time of creating the script, you may forget what you'd like to do with it. It is easier to pick up if you have sufficient comments.

Instead of running the script line by line, you can use

```r
source("advertising-plot.r")
```



or click the `source` button on the editor window to run the whole script.

When you have an error

First, don't panic! Everyone has errors. Read the error message carefully. If you can understand the error message, revise the codes to see if they work. If the codes are related to a specific function, e.g. `mean`, `plot`, type `?function_name` to see help from `R`.

Usually, other people may have similar errors or questions like you, so search on Google to find an answer, e.g. R plot, R add a regression line to a plot, R plot pch, R line cex.lab, etc. or simply R+copy of the error message. Stackoverflow (https://stackoverflow.com) is also a reliable cite where people ask R/statistics/machine learning related questions.