Getting started with R

Hao Wang (haowang.pw)
haowang@asu.edu

January 05, 2018

Contents

1 Why R

- It is a public-domain implementation of the widely regarded S statistical language, and the R/ S platform is a de facto standard among professional statisticians.
- It is comparable, and often superior, in power to commercial products in most of the significant senses: variety of operations available, programmability, graphics, and so on.
- It is available for the Windows, Mac, and Linux operating systems.
- In addition to providing statistical operations, R is a general-purpose programming language, so you can use it to automate analyses and create new functions that extend the existing language features.
- It incorporates features found in object-oriented and functional programming languages.
- The system saves data sets between sessions, so you don't need to reload them each time. It saves your command history too.
- Because R is open source software, it's easy to get help from the user community. Also, a lot of new functions are contributed by users, many of whom are prominent statisticians.

2 Install R and Rstudio

- R can be downloaded from the CRAN prject.
- Rstudio is a commercial IDE (integrated development environment) for R, luckily for individual use it is free. R studio can generate fantastic dynamic decuments with R. For instance, by using rmarkdown, you now say goodbye to copy and paste. This tutorial is written in Rstudio using `rmarkdown`, more intructions can be found here.

3 Use a Script!!

- very, very important, everytime when you start a project with R, keep all the codes in a script. **DO NOT write in console!**

3.1 In Rstudio:

- Using raw script: go to top left, create a new r script
- Using markdown: go to top left, create a new rmarkdown(rmd) file

3.2 In R:

- go to menu bar, *new script*

3.3 Exercise

in Rstudio, create a new r script and a new Rmd file. Name it as "Rworkshop-YOURNAME"

3.4 Embed R codes in .Rmd file

Markdown is a simple language for easy documentation. R codes cannot run directly in a rmd file, you need to specify the coding environment. For example

```
print('hello world')
```

```
## [1] "hello world"
```

The codes for this is

```
 - ```{r chunkname, echo=TRUE, warning=FALSE}
 - print('hello world')
 - ```
```

the `{}` is the chunk environment setting. Mostly we use `r` (which is the programming language setting, you can name this chunk by replacing the `chunkname` part).

For more options of the chunk, please refer to the rmarkdown chunk manual.

4 Basic data types

R treat data in different ways, here I introduce five different types

4.1 Single element

```
a <- 1
a
```

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

```r
# The element can be a string value

b <- "love"
b
```

```
## [1] "love"
```

4.2 Vector

Vector can be presented as a list of elements. In linear algebra, a single row/column of a matrix is called row vector/ column vector.

```r
vector <- c(1, 2, 3, 4, 5)
vector
```

```
## [1] 1 2 3 4 5
```

```r
# vector can be constructed with a list of strings
bvector <- c("happy", "new", "year")
bvector
```

```
## [1] "happy" "new"   "year"
```

4.3 Matrix

A matrix contains multiple vectors, the following provides a simple example.

```r
A <- matrix(seq(from = 1, to = 9, by = 1), nrow = 3, byrow = TRUE)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

4.4 Dataframe

Perhaps the most common data we use in social science is dataframe, often imported from an existing database. For instance, this is an example of Boston housing price from the MASS package

```r
#install.packages("MASS")
library(MASS)
Boston <- Boston
summary(Boston)
```

```
##       crim                zn              indus            chas        
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000  
##  1st Qu.: 0.08204   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000  
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000  
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917  
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000  
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000  
##       nox               rm              age              dis        
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130  
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100  
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207  
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795  
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188  
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127  
##       rad              tax           ptratio          black       
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32  
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38  
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44  
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67  
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23  
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90  
##      lstat            medv      
##  Min.   : 1.73   Min.   : 5.00  
##  1st Qu.: 6.95   1st Qu.:17.02  
##  Median :11.36   Median :21.20  
##  Mean   :12.65   Mean   :22.53  
##  3rd Qu.:16.95   3rd Qu.:25.00  
##  Max.   :37.97   Max.   :50.00
```

```
View(Boston)
```

Difference : often it is hard to distinguish between matrix a dataframe, they look similar. However, the variables (often appears as columns in a dataframe) can consist of different types. General rule is: use data frames if columns (variables) can be expected to be of different types (numeric/character/logical etc.).

Matrices are for data of the same type. Consequently, the choice matrix/data.frame is only problematic if you have data of the same type. The answer depends on what you are going to do with the data in data.frame/matrix. If it is going to be passed to other functions then the expected type of the arguments of these functions determine the choice.

Matrices are a necessity if you plan to do any linear algebra-type of operations.

Data frames are more convenient if you frequently refer to its columns by name (via the compact $ operator).

Data frames are also better for reporting (printing) tabular information as you can apply formatting to each column separately.

We can convert a matrix to a dataframe by function `as.data.frame()`, a dataframe can also be converted into a matrix by function `data.matrix()`

```
# convert matrix A to dataframe
data.A <- as.data.frame(A)

# convert Boston to matrix
matrix.Boston <- data.matrix(Boston)
```

4.5 list

List is somewhat different in R, a list in general can be constructed with ANY thing, you can put a single value, a vector, a matrix, or a dataframe in a list environment.

```
mylist <- list()
mylist$A <- A
mylist$data.A <- data.A
mylist$Boston <- Boston
```

List is very common in the statistical outputs. You can extract information from a list by `$` operator. For example

```
lm <- lm(medv ~ crim + age, data = Boston)
```

We see the new list `lm` is a list of 12 different objects.

```
coef <- lm$coefficients
coef
```

```
## (Intercept)        crim         age
## 29.80066701 -0.31181577 -0.08955328
```

5 Common operators

This section introduces some common operators in R.

5.1 assign operator

- `<-` : This is the assign operator, which is the most common operator in R, assigning something

- = : This one depends, usually it is an assign operator, sometimes it is an environmental setting. I reconmmend using <- whenever you're creating something new.

5.2 math operators

- +, -, *, /, ^

```
1 + 3
```

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

```
6 - 7
```

```
## [1] -1
```

```
2*34
```

```
## [1] 68
```

```
100/2
```

```
## [1] 50
```

```
3^2
```

```
## [1] 9
```

5.3 relational operators

- ==, <, >, >=, <=

```
3 == 2
```

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

```
print(3 == 2)
```

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

```
3 > 2
```

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

```
print(3 > 2)
```

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

```
3 < 2
```

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

```
3 >= 2
```

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

```
3 <= 2
```

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

6 Linear algebra quick review

6.1 Example

Linear algebra provides a way of compactly representing and operating on sets of linear equations. For example, consider the following two equations:

$$4x_1 - 5x_2 = -13$$
$$-2x_1 + 3x_2 = 9$$

There are two equations with two variables, from the higgh school algebra you can find unique solutions for x_1 and x_2. This problem can be written in a matrix format

$$\mathbf{A}x = \mathbf{b}$$
$$\mathbf{A} = \begin{bmatrix} 4 & -5 \\ -2 & 3 \end{bmatrix}$$
$$\mathbf{b} = \begin{bmatrix} -13 \\ 9 \end{bmatrix}$$

often a lower case letter represents a single vector, an upper case letter represents a matrix. A known real number in linear algebra is called scalar.

6.2 Notation

- vector notation let x be a vector with n elements

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{bmatrix}$$

- matrix notation

let A be a 3*3 matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

6.3 Operation

- matrix addition

$$\begin{bmatrix} 3 & 4 & 1 \\ 6 & 7 & 0 \\ -1 & 3 & 8 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 7 \\ 6 & 5 & 1 \\ -1 & 7 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 8 \\ 12 & 12 & 1 \\ -2 & 10 & 8 \end{bmatrix}$$

- multiplication

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$$\mathbf{AB} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}$$

$$\mathbf{BA} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} = \begin{bmatrix} b_{11}a_{11} + b_{12}a_{21} & b_{11}a_{12} + b_{12}a_{22} & b_{11}a_{13} + b_{12}a_{23} \\ b_{21}a_{11} + b_{22}a_{21} & b_{21}a_{12} + b_{22}a_{22} & b_{21}a_{13} + b_{22}a_{23} \\ b_{31}a_{11} + b_{32}a_{21} & b_{31}a_{12} + b_{32}a_{22} & b_{31}a_{13} + b_{32}a_{23} \end{bmatrix}$$

NOTE: BA and AB are different!

- transpose

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

$$\mathbf{A}' = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}' = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{bmatrix}$$

transpose is also writte as **t(A)**

- inverse

define a n $\times$ n matrix I as

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \vdots & 1 \end{bmatrix}$$

This is called an identity matrix. For a nonsingular matrix A (inverse exists), we have

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

and $\mathbf{A}^{-1}$ is the inverse of matrix **A**.

6.4 matrix notation for linear regression

Consider the following simple linear regression function:

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_i + \epsilon_i \\ y_1 &= \beta_0 + \beta_1 x_1 + \epsilon_1 \\ y_2 &= \beta_0 + \beta_1 x_2 + \epsilon_2 \\ y_3 &= \beta_0 + \beta_1 x_3 + \epsilon_3 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_n + \epsilon_n \end{aligned}$$

which is

$$Y = X\beta + \epsilon$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

$$\epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

and we can solve the OLS estimator β by matrix operation

We want to minimize

$$\sum \epsilon^2 = \epsilon' \epsilon$$

which is

$$(Y - X\beta)'(Y - X\beta)$$

Take derivatives, set to 0

$$\frac{d}{d\beta}(Y - X\beta)'(Y - X\beta) = -2X'(Y - X\beta) = 0$$

Therefore

$$X'Y = X'X\beta$$

and

$$\beta = (X'X)^{-1}X'Y$$

6.5 matrix operators in R

```r
A <- matrix(rpois(9, 1), nrow = 3)
B <- matrix(rnorm(9), nrow = 3)
A
```

```
##      [,1] [,2] [,3]
## [1,]    2    0    2
## [2,]    1    0    1
## [3,]    1    0    0
```

```r
B
```

```
##            [,1]        [,2]      [,3]
## [1,]  0.3467890  0.06844245 1.0458924
## [2,] -0.2757973  1.18574291 0.1265805
## [3,]  0.0354059 -0.99029980 0.7611029
```

```r
# Element-wise product
A*B
```

```
##            [,1] [,2]      [,3]
## [1,]  0.6935781    0 2.0917847
## [2,] -0.2757973    0 0.1265805
## [3,]  0.0354059    0 0.0000000
```

```r
# Addition
A + B
```

```
##           [,1]        [,2]      [,3]
## [1,] 2.3467890  0.06844245 3.0458924
## [2,] 0.7242027  1.18574291 1.1265805
## [3,] 1.0354059 -0.99029980 0.7611029
```

```r
# Subtraction
A - B
```

```
##           [,1]        [,2]       [,3]
## [1,] 1.6532110 -0.06844245  0.9541076
## [2,] 1.2757973 -1.18574291  0.8734195
## [3,] 0.9645941  0.99029980 -0.7611029
```

```r
# Matrix mutiplication
A %*% B
```

```
##           [,1]        [,2]     [,3]
## [1,] 0.7643899 -1.84371470 3.613991
## [2,] 0.3821949 -0.92185735 1.806995
## [3,] 0.3467890  0.06844245 1.045892
```

```r
# Matrix transpose
t(A)
```

```
##      [,1] [,2] [,3]
## [1,]    2    1    1
## [2,]    0    0    0
## [3,]    2    1    0
```

```r
# Matrix inverse
solve(B)
```

```
##           [,1]       [,2]       [,3]
## [1,] 1.6770981 -1.7750222 -2.0094272
## [2,] 0.3498223  0.3702504 -0.5422959
## [3,] 0.3771498  0.5643195  0.7017581
```

```r
# Generalized inverse
require(MASS)
ginv(B)
```

```
##           [,1]       [,2]       [,3]
## [1,] 1.6770981 -1.7750222 -2.0094272
## [2,] 0.3498223  0.3702504 -0.5422959
## [3,] 0.3771498  0.5643195  0.7017581
```

7 Write your own function

R is very powerful at creating your own customized functions. Sometimes we are not satisfied by the can soluations. Here I provide an example of calculating means

7.1 basic setting

```r
myfunction.mean <- function(x){
  temp <- sum(x)/length(x)
  return(temp)
}

a <- seq(1, 100, by = 1)
myfunction.mean(a)
```

```
## [1] 50.5
```

```r
# compare with default function mean
mean(a)
```

```
## [1] 50.5
```

7.2 function for linear regression

We can write our own function for linear regression coefficients based on matrix natotions

```r
myols <- function(y, x){
# add intercept
  int <- rep(1, nrow(x))
  x <- cbind(int, x) #combine orginal x and intercept
  beta <- solve(t(x) %*% x) %*% t(x) %*% y
  return(beta)
}

# generating randam numbers
set.seed(1)
y <- rnorm(100)
x1 <- rpois(100, 1)
x2 <- rnorm(100)
x <- cbind(x1, x2)

myols(y, x)
```

```
##             [,1]
## int  0.05709859
## x1   0.06472452
## x2  -0.08484081
```

```r
# compare with the default function
lm(y~x)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Coefficients:
## (Intercept)          xx1          xx2
##     0.05710      0.06472     -0.08484
```

7.3 a simple loop

Sometimes we need to perform the same function for multiple times. For instance, we want to know the frequency of "1" in a list of random numbers

```
x <- rpois(20, 1)
x
```

```
##  [1] 1 1 1 1 1 3 1 2 2 1 2 1 0 1 1 0 1 1 0 1
```

```
count1 <- function(x){
  k <- 0 #initial value
  for (i in 1:length(x)) {
  if (x[i] == 1)
  k <- k + 1
  }
  return(k)
}
count1(x)
```

```
## [1] 13
```

8 Workspace and importing files

The workspace in R contains data and other objects. User defined objects created in a session will persist until R is closed. If the workspace is saved before quitting R, the objects created during the session will be saved.

8.1 working directory

It is convenient to create a folder or directory with a short path name to store data and codes.

```
setwd() this is command to set up your working directory

in Windows
setwd("C:/Users/User Name/Documents/FOLDER")

in Macs
setwd("/Users/User Name/ Documents/FOLDER")
```

The working directory has to be a folder. You can either type in, or copy and pasted the appropriate pathway which you would like to work in.

8.2 install and using packages

R packaages are reprodducible and reusable R-codes. They contain lots of convenient `can` functions, often packages published on CRAN are tested by R users.

To use certain functions in a R package, you need to install and run the packages first. The commands are

```
install.packages("package names")
library(package names)
```

for example

```
install.packages("car")
library(car)
```

some useful packages

- dplyr: data management
- ggplot2: graphing
- stargazer: produce tables
- plm: time-series cross-sectional
- glm: generalized linear models
- foreign: can read multiple types of files into R

8.3 import raw datafiles

R often has multiple ways to load an existing data into working memory. Here are some examples. First we need to setup a working directory.

```
setwd("D:/git/POS603-Lab/2018-R-Workshop")
#setwd("D:/git/POS603-Lab/2018-R-Workshop")
getwd()
```

```
## [1] "D:/git/POS603-Lab/2018-R-Workshop"
```

1. Read csv (comma separated values) file: use read.csv function

```
#R can read files directly from the website
mydata.csv <- read.csv('http://www.cyclismo.org/tutorial/R/_static/simple.csv', header =
write.csv(mydata.csv, "mydata.csv")
#header option means taking the first row as variable names.
#sep option is determined by the way the data file is separated
```

2. Read txt

We create a txt data first and then import the txt file to R.

```
require(car)
```

```
## Loading required package: car
```

```
write.table(cars,"cars.txt", sep = "\t")
mydata.txt <- read.table("cars.txt")
```

3. Read dta(stata)

- For stata 12 and earlier (package 'foreign')
- For stata 13 and later (package'readstata13', this also works for earlier versions)

```
#install.packages(c('readstata13', 'foreign'))
require(readstata13)
```

```
## Loading required package: readstata13
```

```
require(foreign)
```

```
## Loading required package: foreign
```

```
stata14 <- read.dta13('stata14.dta')
stata12 <- read.dta('stata12.dta')
```

9 Simple data management

9.1 Create new variables

```
library(car)
mydata <- cars

#creat new variable, which is the square of original variable
mydata$speed2 <- (mydata$speed)^2

#This line create dummny variables based on speed
mydata$speed3 <- ifelse(mydata$speed > 10,
c("slow"), c("quick"))
mydata$speed3
```

```
##  [1] "quick" "quick" "quick" "quick" "quick" "quick" "quick" "quick"
##  [9] "quick" "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"
## [17] "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"
## [25] "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"
## [33] "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"
## [41] "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"  "slow"
```

```
## [49] "slow"  "slow"
```

```
# another example: create 3 categories
mydata$speed4[mydata$speed < 10] <- "slow"
mydata$speed4[mydata$speed >= 10 & mydata$speed < 18] <- "middle"
mydata$speed4[mydata$speed >= 18] <- "quick"
mydata$speed4
```

```
##  [1] "slow"   "slow"   "slow"   "slow"   "slow"   "slow"   "middle"
##  [8] "middle" "middle" "middle" "middle" "middle" "middle" "middle"
## [15] "middle" "middle" "middle" "middle" "middle" "middle" "middle"
## [22] "middle" "middle" "middle" "middle" "middle" "middle" "middle"
## [29] "middle" "middle" "middle" "quick"  "quick"  "quick"  "quick"
## [36] "quick"  "quick"  "quick"  "quick"  "quick"  "quick"  "quick"
## [43] "quick"  "quick"  "quick"  "quick"  "quick"  "quick"  "quick"
## [50] "quick"
```

9.2 Subsetting data

```
require(car)
mydata <- cars
#selecting by observation values
mydata.sub1 <- subset(mydata, subset = speed >= 10)

#selecting by columns
mydata.sub2 <- subset(mydata, select = c(speed))

#reverse selecting
mydata.sub3 <- subset(mydata, select = -c(speed))
```

10 Working Example: Baylor's American Religious Survey

```
library(ggplot2)
library(MASS)
library(readstata13)
```

The following R chunk load data from Baylor Regilious Survey. Baylor Religious Study is a comprehensive analysis on religious beliefs in the United States. For detailed explanation please refer to http://www.thearda.com/Archive/Files/Descriptions/BAYLORW2.asp. Total sample size is 1648 adults, with 318 variables, survey was conducted by Gallup.

```
#The first step is to load Byalor Religious Survey 2005 (I)
mydata <- read.dta13("http://www.thearda.com/download/download.aspx?file=Baylor%20Religi
```

10.1 Subsetting Data

Because the raw file includes some missing points, I create a new dataset of potential interesting variables without missing data.

```
plotdata <- mydata[, c('relgious', 'attend', 'gaymarr', 'gayborn',
                       'votefem', 'partyid')]
#religious: how religious you are
#attend:    how often do you attend church
#gaymarr:   gay marriage should be legal
#gayborn:   people are born either homo or heterosexual
#vetefem:   would you vote for a female cadidate nominated by your party
#partyid:   democract or republican
plotdata <- na.omit(plotdata)
```

10.2 Mean attitudes towards gay marriage

```
summary(plotdata$gaymarr)
```

```
## Strongly disagree          Disagree             Agree    Strongly agree
##               507               311               223               291
##         Undecided
##               159
```

```
mean(plotdata$gaymarr)
```

```
## Warning in mean.default(plotdata$gaymarr): argument is not numeric or
## logical: returning NA
```

```
## [1] NA
```

What happened? we got an error message since all the survey items are coded in the categorical way.

- Convert categorical data into numeric

```
plotdata2 <- as.data.frame(sapply(plotdata, as.numeric))#sapply function returns matrix
#legnth, and in the same time converted factors into numeric numbers by as.numeric comma
table(plotdata2$gaymarr)
```

```
##
##   1   2   3   4   5
```

```
## 507 311 223 291 159
```

```
mean(plotdata2$gaymarr)
```

```
## [1] 2.519785
```

- Question: How to interpret the mean value of plotdata2? Is it correct?
- Recode variables

```
plotdata2$gaymarr1 <- plotdata2$gaymarr
plotdata2$gaymarr1[plotdata2$gaymarr1 == 5] <- NA
mean(plotdata2$gaymarr1, na.rm = TRUE)
```

```
## [1] 2.223724
```

```
median(plotdata2$gaymarr1, na.rm = TRUE)
```

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

- Get an overview by the summary() command

```
summary(plotdata)
```

```
##                  relgious                     attend
##  Not at all religious:165   Never                :327
##  Not too religious   :213   Weekly               :318
##  Somewhat religious  :617   Several times a year :168
##  Very religious      :483   Once or twice a year :155
##  Don't know          : 13   Several times a week :137
##                             Less than once a year:122
##                             (Other)              :264
##               gaymarr                  gayborn    votefem
##  Strongly disagree:507   Strongly disagree:254   Yes:1190
##  Disagree         :311   Disagree         :259   No : 301
##  Agree            :223   Agree            :420
##  Strongly agree   :291   Strongly agree   :281
##  Undecided        :159   Undecided        :277
##
##
##                partyid
##  Independent        :311
##  Moderate Democrat  :281
##  Moderate Republican:261
##  Strong Democrat    :196
##  Strong Republican  :174
##  Leaning Republican :125
##  (Other)            :143
```

- Frenquency Table

```
table(plotdata$gaymarr)
```

```
##
## Strongly disagree          Disagree             Agree    Strongly agree
##               507               311               223               291
##         Undecided
##               159
```

10.3 Using ggplot2

The ggplot2 package, created by Hadley Wickham, offers a powerful graphics language for creating elegant and complex plots. Its popularity in the R community has exploded in recent years. Origianlly based on Leland Wilkinson's The Grammar of Graphics, ggplot2 allows you to create graphs that represent both univariate and multivariate numerical and categorical data in a straightforward manner. Grouping can be represented by color, symbol, size, and transparency. The creation of trellis plots (i.e., conditioning) is relatively simple.

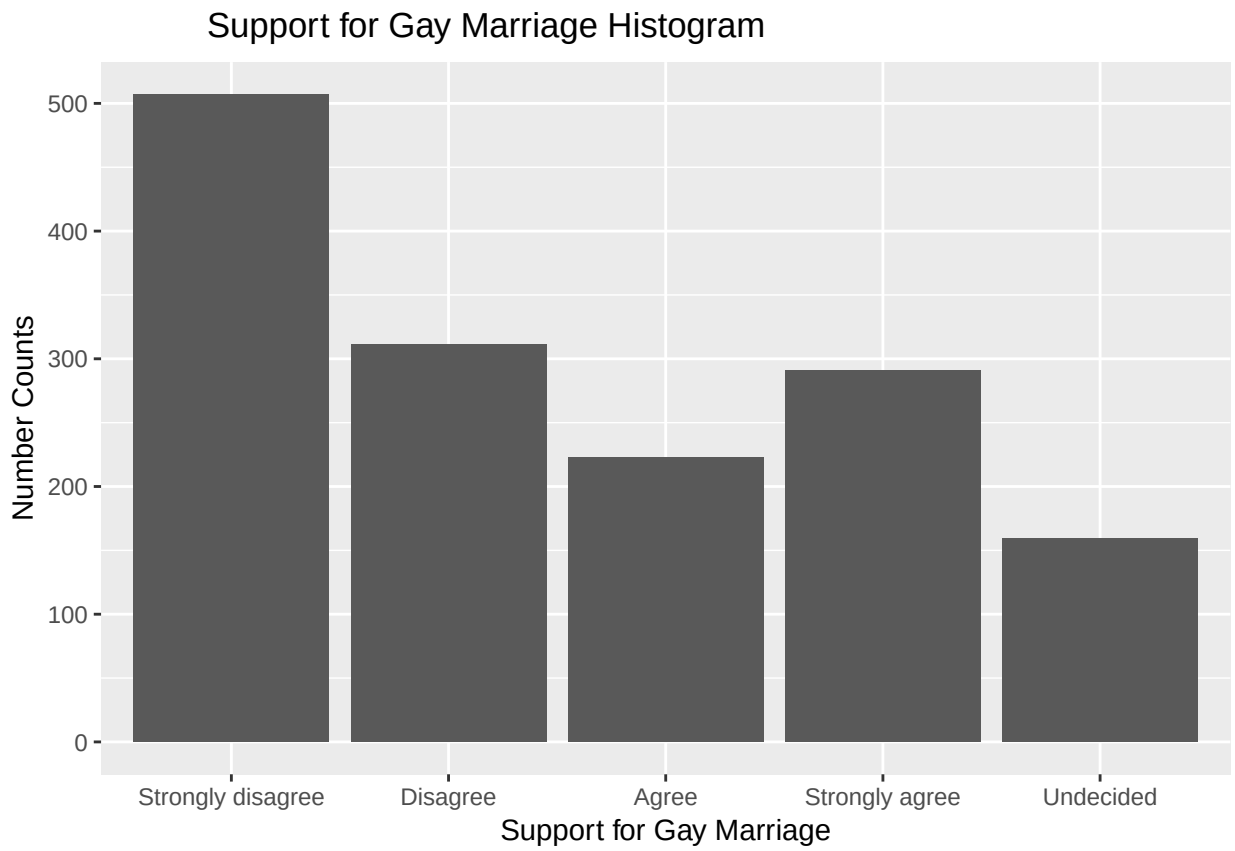
ggplot 2 reference guide: http://docs.ggplot2.org/current/index.html#

ggplot 2 cheatsheet: https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

10.3.1 Histogram

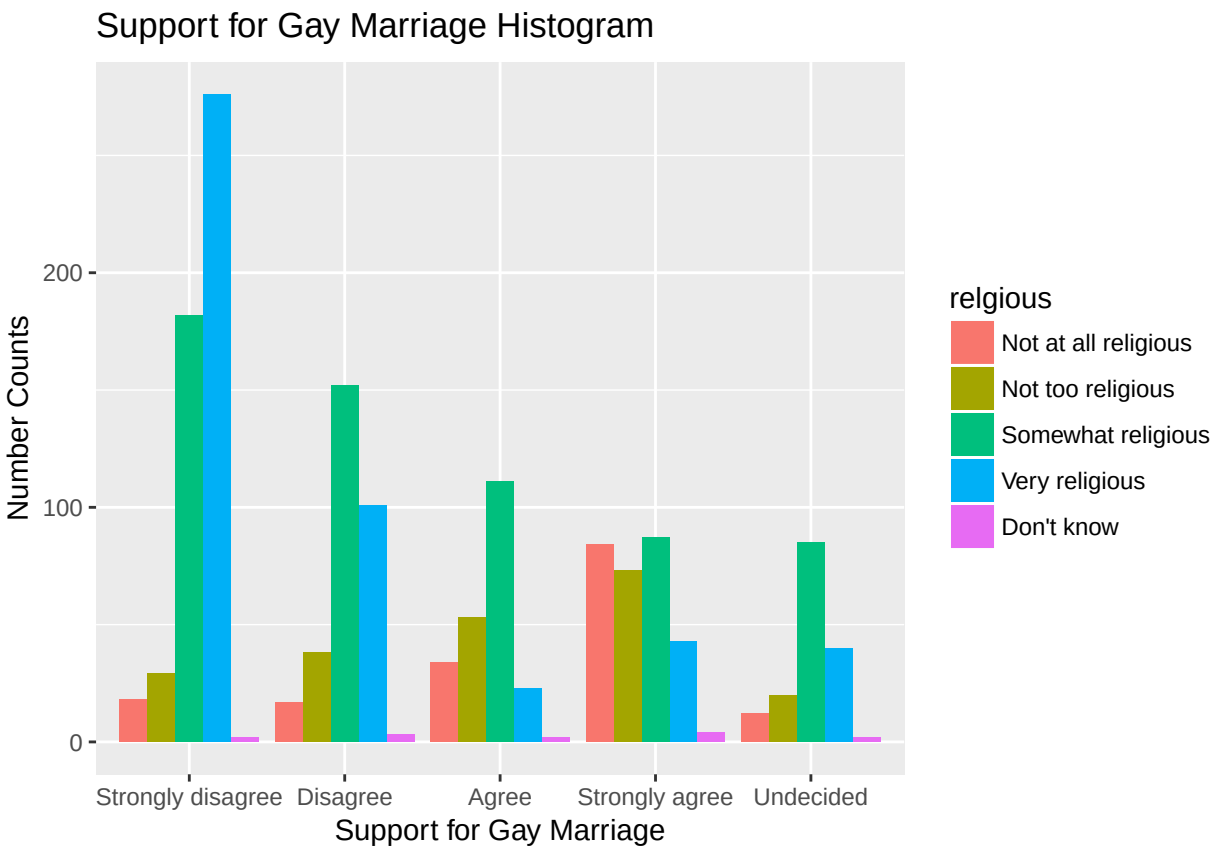
This show the histogram of Attitudes on gay marriage

```
library(ggplot2)
hist.gay <- ggplot(plotdata, aes(x = gaymarr)) +
            geom_histogram(stat="count", alpha =1) +
            labs(x="Support for Gay Marriage", y="Number Counts") +
            ggtitle("           Support for Gay Marriage Histogram")
hist.gay
```



10.3.2 Histogram according to religious degrees.

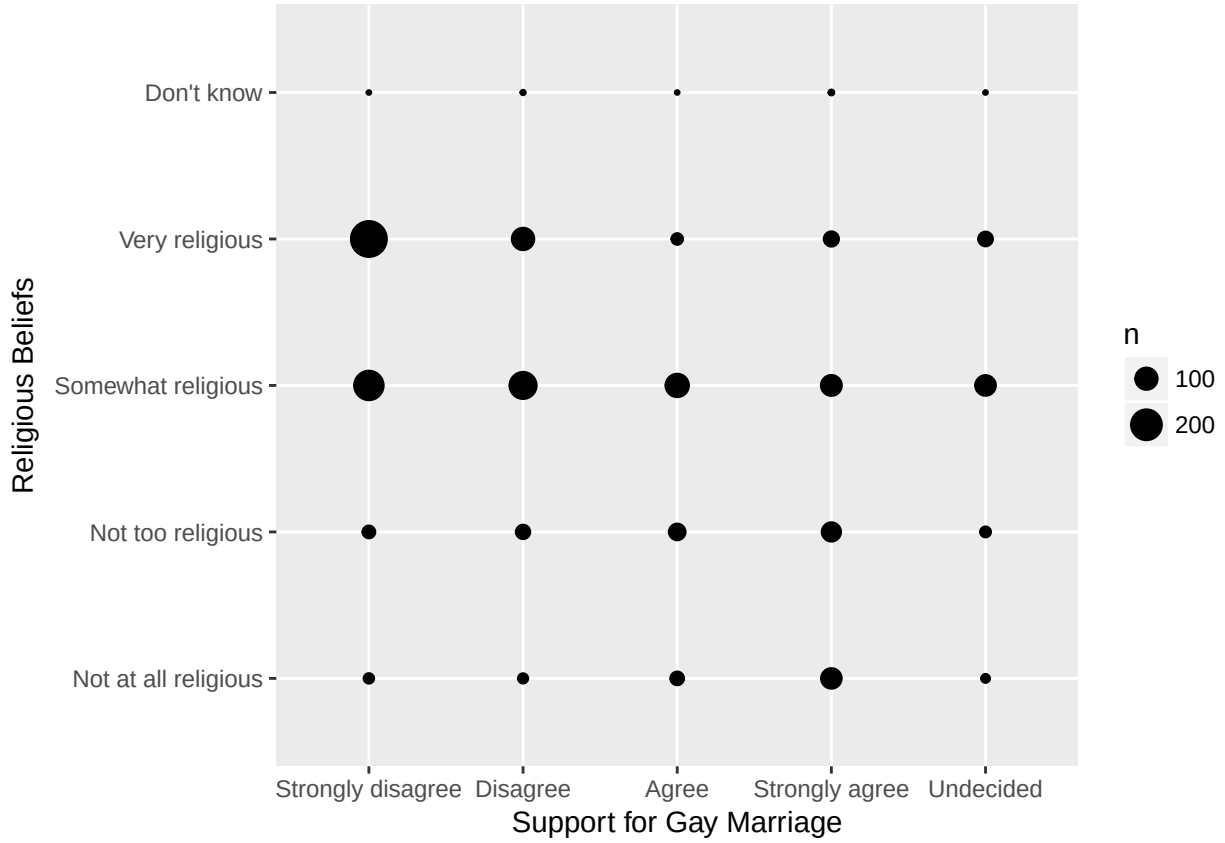
```
hist.relig.gay <- ggplot(plotdata, aes(x =gaymarr, fill=relgious)) +
                  geom_histogram(stat="count",alpha =1, position = "dodge")+
                  ylab("Number Counts") +
                  xlab("Support for Gay Marriage")+
                  ggtitle("Support for Gay Marriage Histogram")
hist.relig.gay
```



10.3.3 Scatter Plot

In scatter plot we write two parameters in the aes() option. I use additional option geom_count() here to illustrate the size.

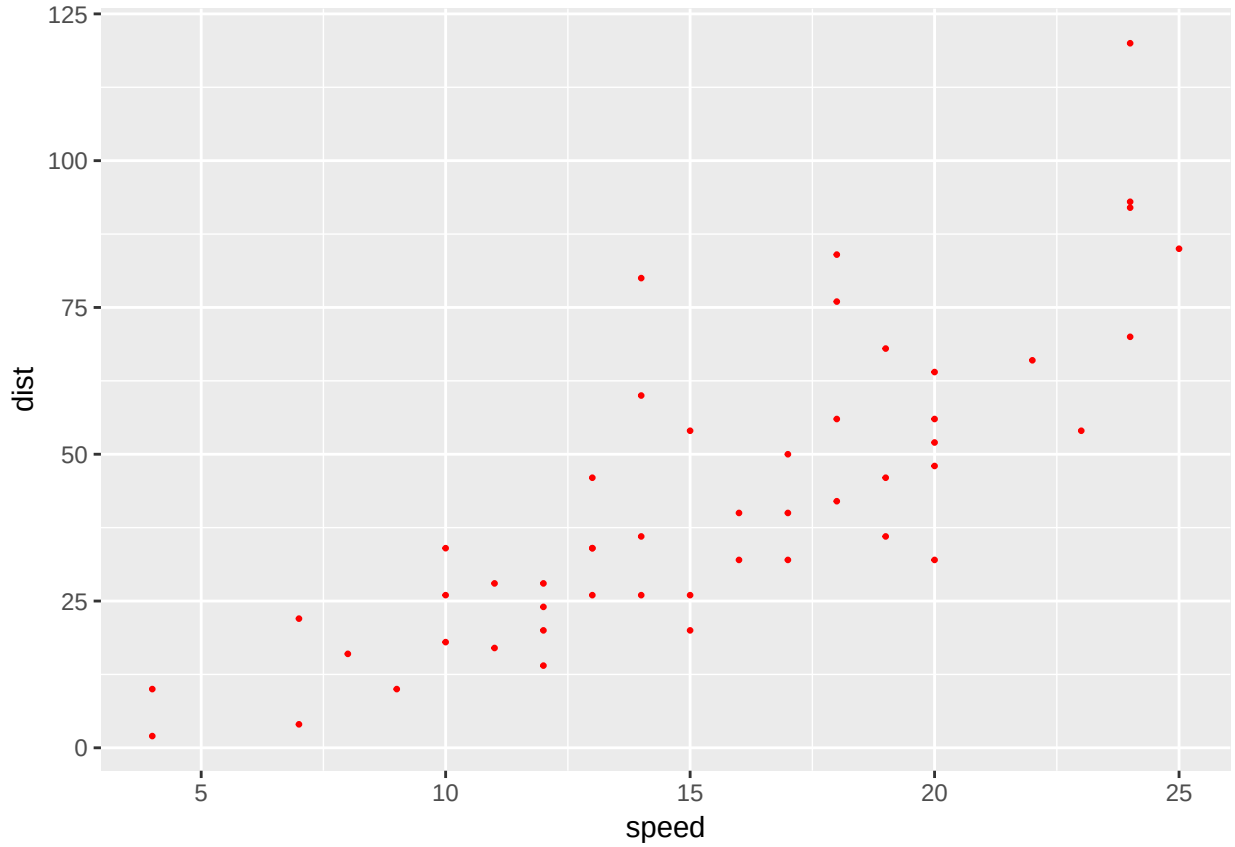
```
relig.gaymarr <- ggplot(data=plotdata, aes(y = relgious, x = gaymarr)) +
                 geom_count() +
                 scale_size_area() +
                 ylab("Religious Beliefs") +
                 xlab("Support for Gay Marriage")
relig.gaymarr
```



10.3.4 Adding lines

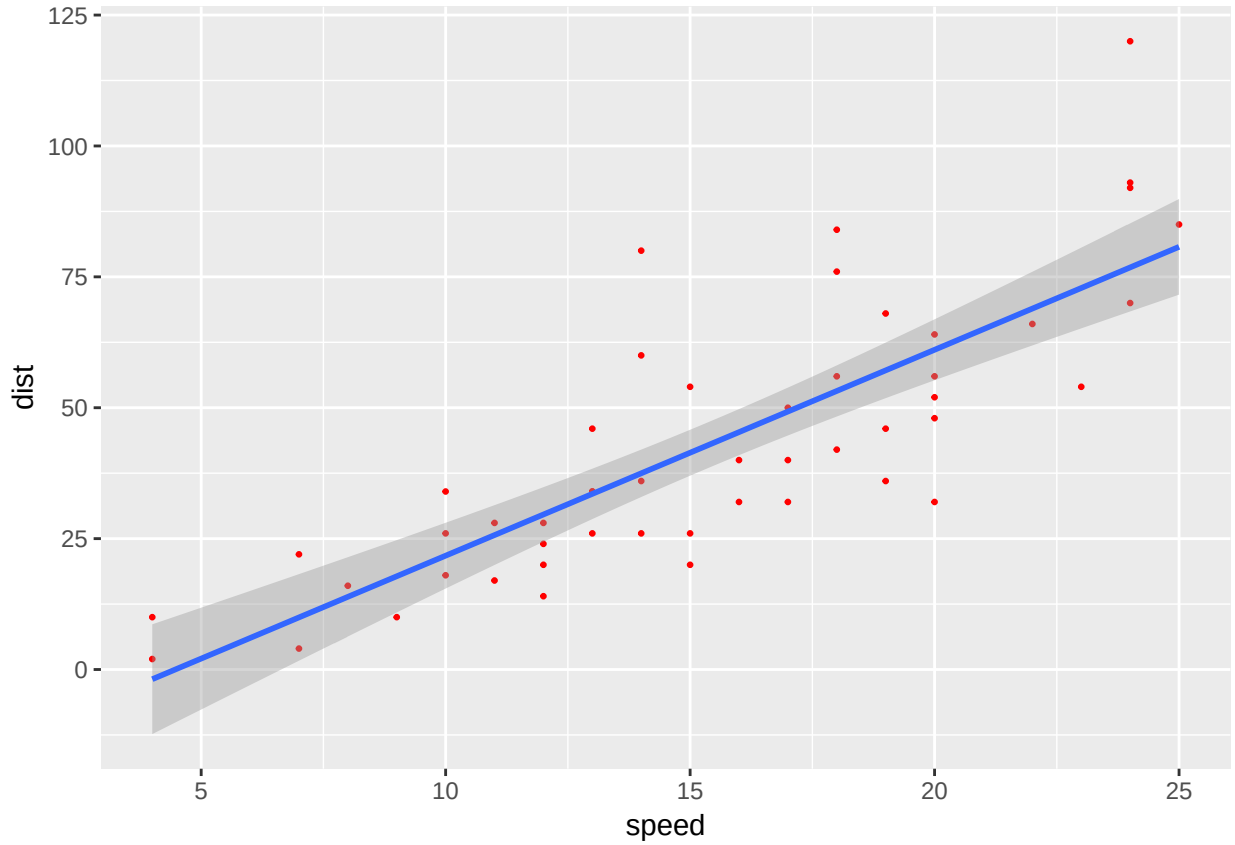
Let's use the default data from the car package here. It measures the relation between car speed and stop distance.

```
mydata <- cars
car <- ggplot(data = mydata, aes(x = speed, y = dist)) +
       geom_point(size = 0.5, color = "red")
car
```



1. Method 1, use the default geom_smooth()

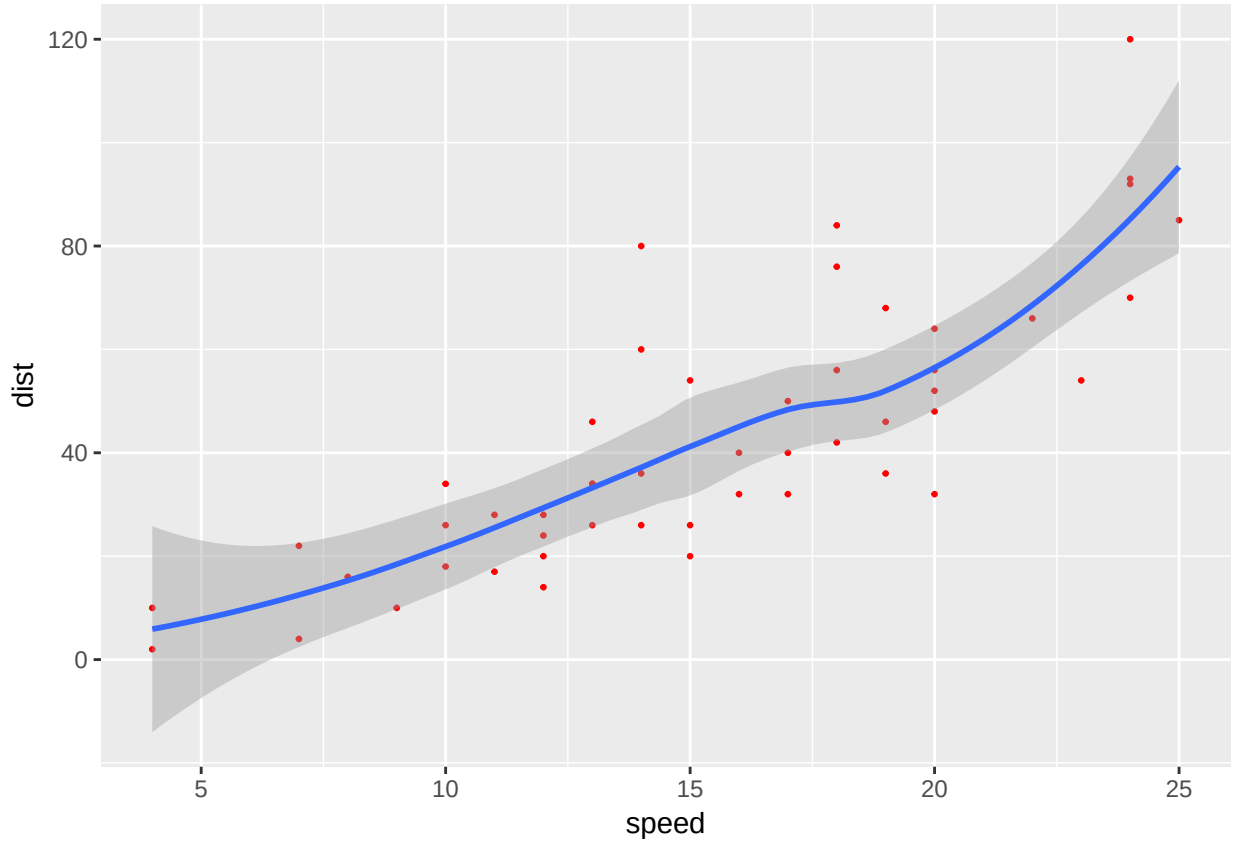
```
car <- ggplot(data = mydata, aes(x = speed, y = dist)) +
       geom_point(size=0.5, color = "red") +
       geom_smooth(method = 'lm', se=TRUE)
car
```



2. You can try other options of geom_smooth()

LOESS is a nonparametric method that combine multiple regression models in a k-nearest-neighbor-based modeling.

```
car <- ggplot(data = mydata, aes(x = speed, y = dist)) +
       geom_point(size=0.5, color = "red") +
       geom_smooth(method = 'loess', se=TRUE)
car
```



3. If you want full control over your line In this condition you need to calculate all the parameters of the line. Let's try with linear models

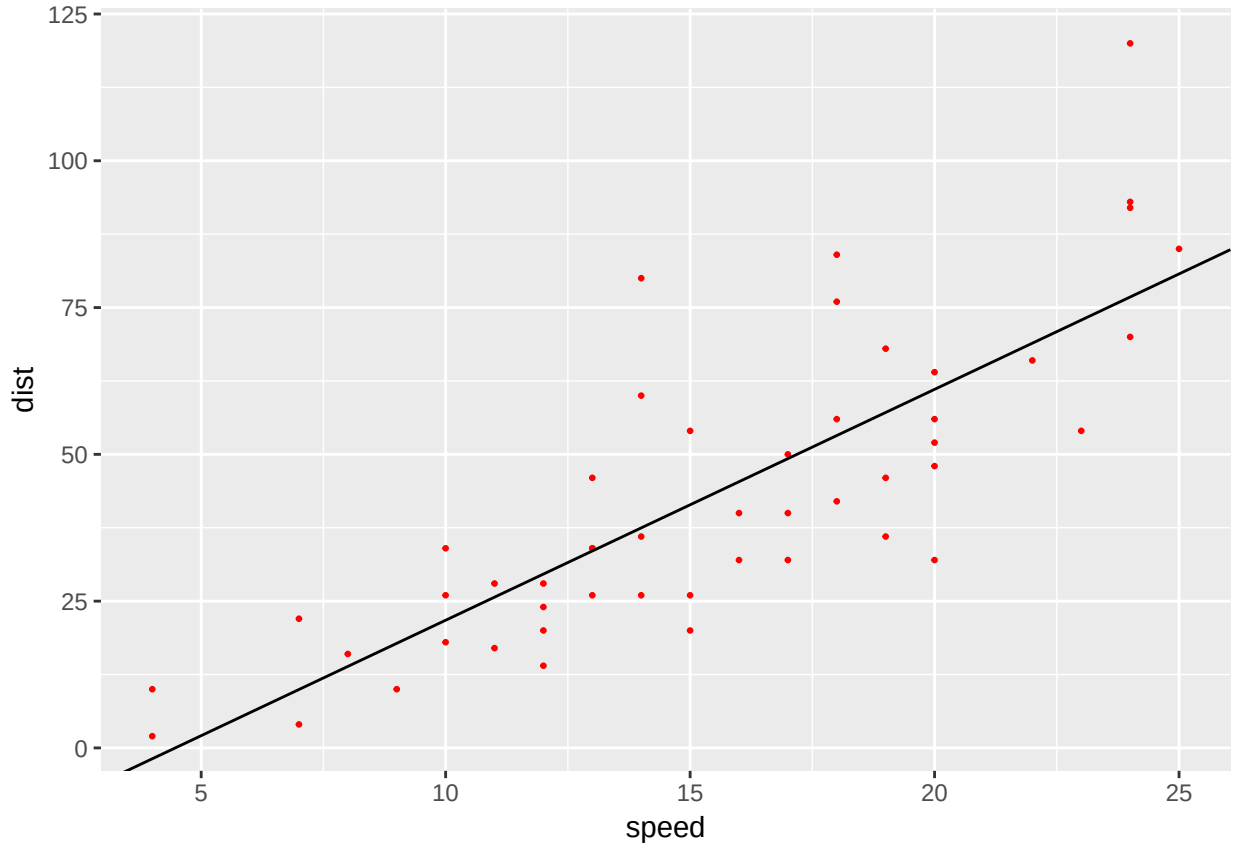
```
lm <- lm(dist ~ speed, data = mydata)
summary(lm)
```

```
##
## Call:
## lm(formula = dist ~ speed, data = mydata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## speed         3.9324     0.4155   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.38 on 48 degrees of freedom
```

```
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

```
intercept <- summary(lm)$coefficients[1,1]
slope <- summary(lm)$coefficients[2,1]

car <- ggplot(data = mydata, aes(x = speed, y = dist)) +
       geom_point(size = 0.5, color = "red") +
       geom_abline(intercept = intercept, slope = slope)
car
```



The summary(lm) command returns coefficients of the regression. In this case we need to extract intercept in row 1, column 1; and slope in row 2, column 1.

11 MISC: install stan packages for Bayesian analysis

Please follow the guide on this page to install rethinking package. http://xcelab.net/rm/software/

You need to run the following code in R:

```
install.packages(c('rstan',' devtools','coda','mvtnorm','loo'))
```

```
library(devtools)
install_github("rmcelreath/ rethinking")
library(rethinking)
```