R Intro - Lecture 1

Mikael Durling

2023-10-06

Outline

- R history and philosophy
- Getting started
- R fundamentals
- R as statistics tables
- Basic data import
- Basic plot
- Packages
- Keeping track of your work

In the beginning, there was ‘S’

History

S was developed in '70s at Bell labs as an alternate and interactive language and environment - enable exploratory data analysis (as advocated by Tukey *et al*). (In contrast to the more static systems already available at that time.)

R was developed by Ross Ihaka and Robert Gentleman at University Auckland. A programming language to learn statistics. R is the letter following S!

Open source software!

One of the major preferred languages for data science "big data".

History

R was designed from the beginning to be extendable. Users are encouraged to create packages that cater for different uses.

Re-usable code is encouraged. If you write code to solve a problem once, don't re-invent the wheel the next time!

Now base R is extended with a rich ecosystem of packages that add end extend functionality. The Comprehensive R Archive Network (CRAN), BioConductor etc

Getting started

There are several ways to run R

- Plain R on the command line (Unix/Linux systems, MacOS, Windows)
- R for MacOS and Windows with slightly more Graphical user interfaces and provisions to edit R code within the program interface.
- RStudio - An integrated development environment for R (and Python) where the user has many tools to be able to work smoothly when writing R code.

'Vanilla' R

```
Mikaels-MacBook-Pro:~ mikael$ R

R version 4.3.0 (2023-04-21) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computi
Platform: x86_64-apple-darwin20 (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.

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.
```

R / R Studio

- How to launch
- Where am I?

R Fundamentals

R is a programming language! However, it is mostly used in an interactive manner where you explore the data.

As with all languages data is stored in variables. In R, there are four major kinds of variables:

- Vectors, stores one or more values of the same kind
- Lists, stores one or more values of possibly different kinds
- Array, an N-dimensional array. (Matrix is the two-dimensional special case)
- Dataframes, stores one or more (possibly) named vectors of equal length. This is similar to a table with named columns. The dataframe and it's derivates is the major way to store data to be analysed.

Basic evaluation

R can be used as your basic calculator. Unless you store the result somewhere, R prints it directly:

```
> 1 + 2 * sqrt(9)
[1] 6
>
```

Assignment is done with -> or <- (don't use = to avoid mistakes):

```
1 + 1 -> variable.one
# same as
variable.two <- 1 + 1
variable.one == variable.two
```

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

Vectors

Remember that a vector is the basic type. Any operator applies to all elements:

```
my.vector <- c(1, 2, 3)
my.vector * 3
```

```
## [1] 3 6 9
```

But that is usefull too:

```
# Normalise by mean:
vector.a <- 1:10
vector.b <- vector.a / mean(vector.a)
vector.b
```

```
##  [1] 0.1818182 0.3636364 0.5454545 0.7272727 0.9090909 1
##  [8] 1.4545455 1.6363636 1.8181818
```

Coersion

Sometimes a vector ends up not beeing the kind you want.

```
vector.a <- c('1', '2', '3', '3', '3')
vector.a
```

```
## [1] "1" "2" "3" "3" "3"
```

```
as.numeric(vector.a)
```

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

There are many more “as.something” functions available.

Factorial data

Let's revisit the vector of characters

```
vector.a
```

```
## [1] "1" "2" "3" "3" "3"
```

```
# Convert to factorial data
as.factor(vector.a)
```

```
## [1] 1 2 3 3 3
## Levels: 1 2 3
```

```
# Levels can be any string
as.factor( c( "apple", "apple", "pineapple", "lemon"))
```

```
## [1] apple     apple     pineapple lemon
## Levels: apple lemon pineapple
```

Indexing vectors

Indexing (picking a):

```
vector.a <- 1:5 * 2
vector.a
```

```
## [1]  2  4  6  8 10
```

```
# Indices start on 1. Get one element
vector.a[2]
```

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

```
# You can pick more than one
vector.a[c(1,3,2)]
```

```
## [1] 2 6 4
```

Indexing vectors

```
# Or filter by setting TRUE/FALSE for values to keep
vector.a[c(TRUE, TRUE, FALSE, TRUE, FALSE)]
```

```
## [1] 2 4 8
```

```
# Can be used to filter
vector.a[vector.a > 5]
```

```
## [1]  6  8 10
```

Functions

What would mathematics be without functions?

Worth to note:

- R supports both ordered and named arguments
- Many cases of both

Functions

```
# Define simple function
my.fun <- function(a, b) ( a * b ) / (b * a)

# Call the function
my.fun( 13, 5.5 )
```

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

```
# Slightly more elaborate
my.fun <- function(a, b) {
  foo <- a *b
  foo <- foo / (a*b)
  foo
}
my.fun(13, 5.5)
```

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

Distributions in R

R provides function to work with most distribution. Density, distribution function, quantile function and random number generators are provided. (d-, p-, q- and r- function). Ie for the normal distribution:

- `dnorm` density function
- `pnorm` distribution function
- `qnorm` quantile function
- `rnorm` random number generator function

R for Critial value lookup

What is the probability that 2 is drawn from a normal distribution with mean 0 and sd 1?

```
1 - pnorm(2, mean=0, sd=1)
```

```
## [1] 0.02275013
```

Similarly, drawing 10000 random numbers from the normal distrinution, how many will be in the 5% upper tail?

```
sum(pnorm(rnorm(10000)) < 0.05)
```

```
## [1] 498
```

Distribution functions

Some other distributions aviable:

- chisq (Chi-squared)
- student (Student's t)
- pois (Poisson)
- binom (Binomial)

Basic testing

All basic statistical tests are available. Eg.

```
vector.a <- rnorm(20, mean=4)
vector.b <- rnorm(20, mean=5)
t.test( vector.a, vector.b )
```

```
##
##  Welch Two Sample t-test
##
## data:  vector.a and vector.b
## t = -4.8964, df = 37.898, p-value = 1.845e-05
## alternative hypothesis: true difference in means is not
## 95 percent confidence interval:
##  -1.7080191 -0.7087334
## sample estimates:
## mean of x mean of y
##  4.038993  5.247370
```

Getting help

Help is always near

```
help(rbinom)
help.search("uniform")
```

will provide standardise help text. Terribly usful when you forget the parameter names.

There is also ‘help.search(“whatever”)

Dataframes

A data.frame is how tabular data is stored in R. It is basically a collection of named vectors. When modelling, having the data in a properly formatted data.frame is pivotal. Let us create a dummy data.frame:

```
my.data <- data.frame(a=1:10, b=rnorm(10), d=as.factor( rpo
                      e=c('apple', 'pear')) # Example of re-use
summary(my.data)
```

```
##        a                b              d          e
##  Min.   : 1.00   Min.   :-1.28178   0:1   Length:10
##  1st Qu.: 3.25   1st Qu.:-0.46667   1:4   Class :charact
##  Median : 5.50   Median : 0.29364   2:1   Mode  :charact
##  Mean   : 5.50   Mean   : 0.07032   3:3
##  3rd Qu.: 7.75   3rd Qu.: 0.76585   5:1
##  Max.   :10.00   Max.   : 1.12252
```

Data frames

Columns can accessed by name or index (first index is column, second is row)

- `my.data$age` will give the age column as a vector
- `my.data[2]` will do the same
- `my.data[2,]` will get the second line as a new 1-row dataframe
- Similarly `my.data[c(1,3,5),]` will extract a dataframe consisting of lines 1, 3 and 5.
- Indexing can be used for filtering in the same fashon as shown earlier for vectors!

Data frames

```
summary(my.data)
```

```
##        a                b              d          e
##  Min.   : 1.00   Min.   :-1.28178   0:1   Length:10
##  1st Qu.: 3.25   1st Qu.:-0.46667   1:4   Class :charact
##  Median : 5.50   Median : 0.29364   2:1   Mode  :charact
##  Mean   : 5.50   Mean   : 0.07032   3:3
##  3rd Qu.: 7.75   3rd Qu.: 0.76585   5:1
##  Max.   :10.00   Max.   : 1.12252
```

```
my.data$e
```

```
##  [1] "apple" "pear"  "apple" "pear"  "apple" "pear"  "ap
## [10] "pear"
```

Not expected?

Dataframes

Let's fix it. It should be a factor:

```
my.data$e <- as.factor(my.data$e)
summary(my.data)
```

```
##        a                b               d          e
##  Min.   : 1.00   Min.   :-1.28178   0:1   apple:5
##  1st Qu.: 3.25   1st Qu.:-0.46667   1:4   pear :5
##  Median : 5.50   Median : 0.29364   2:1
##  Mean   : 5.50   Mean   : 0.07032   3:3
##  3rd Qu.: 7.75   3rd Qu.: 0.76585   5:1
##  Max.   :10.00   Max.   : 1.12252
```

Data import

Importing data to a data frame

```
# Read data from file or URL
my.data <- read.csv("https://raw.githubusercontent.com/miko
summary(my.data)
```

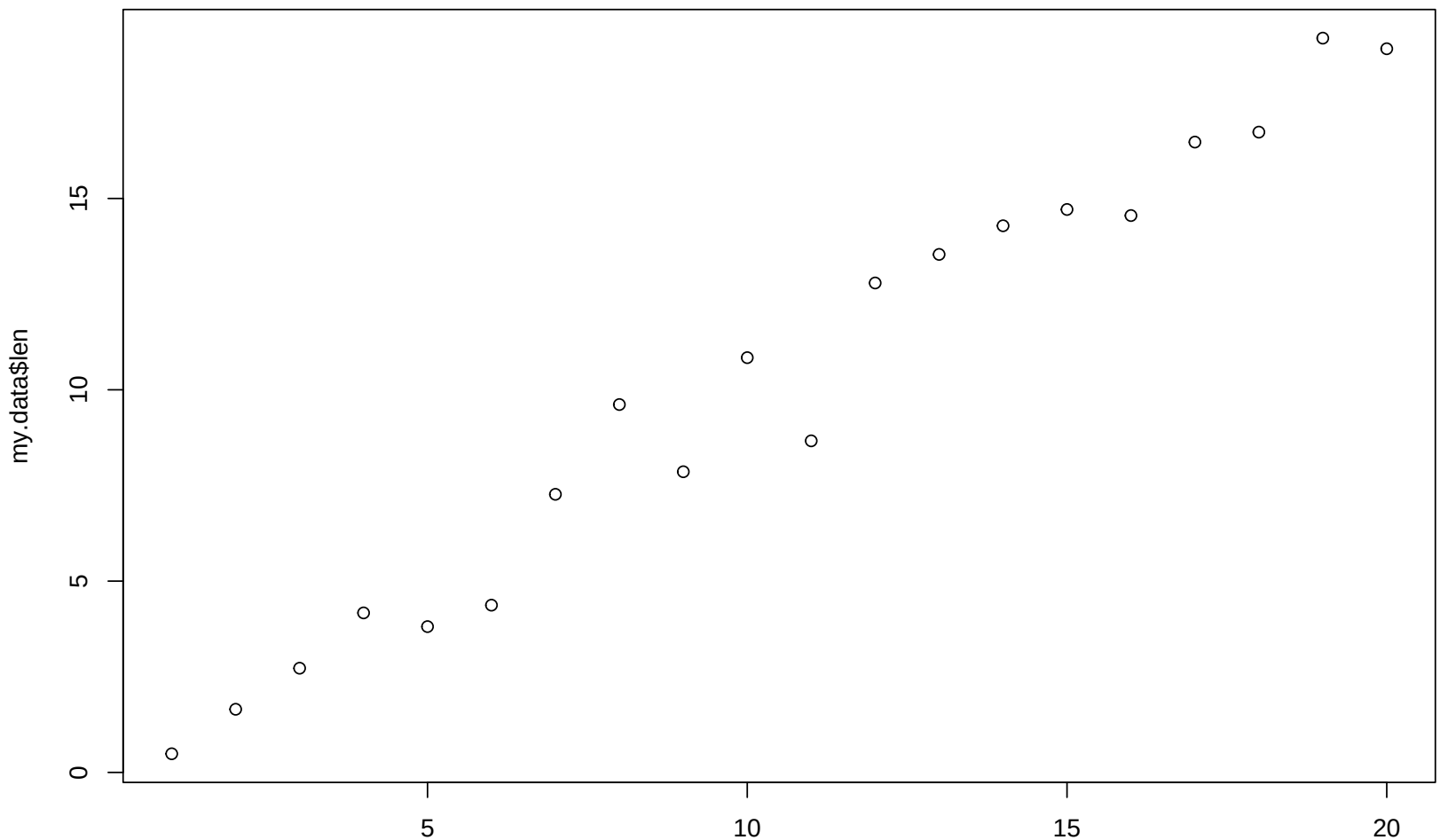
```
##        age              len
##  Min.   : 1.00   Min.   : 0.4872
##  1st Qu.: 5.75   1st Qu.: 4.3207
##  Median :10.50   Median :10.2271
##  Mean   :10.50   Mean   :10.1326
##  3rd Qu.:15.25   3rd Qu.:14.5935
##  Max.   :20.00   Max.   :19.1905
```

In *tidyverse* there are neat functions to read sheets from Excel workbooks into dataframes! Use those rather than exporting to CSV and then import.

Basic plotting - scatter

Basic plotting is done with `plot` function:

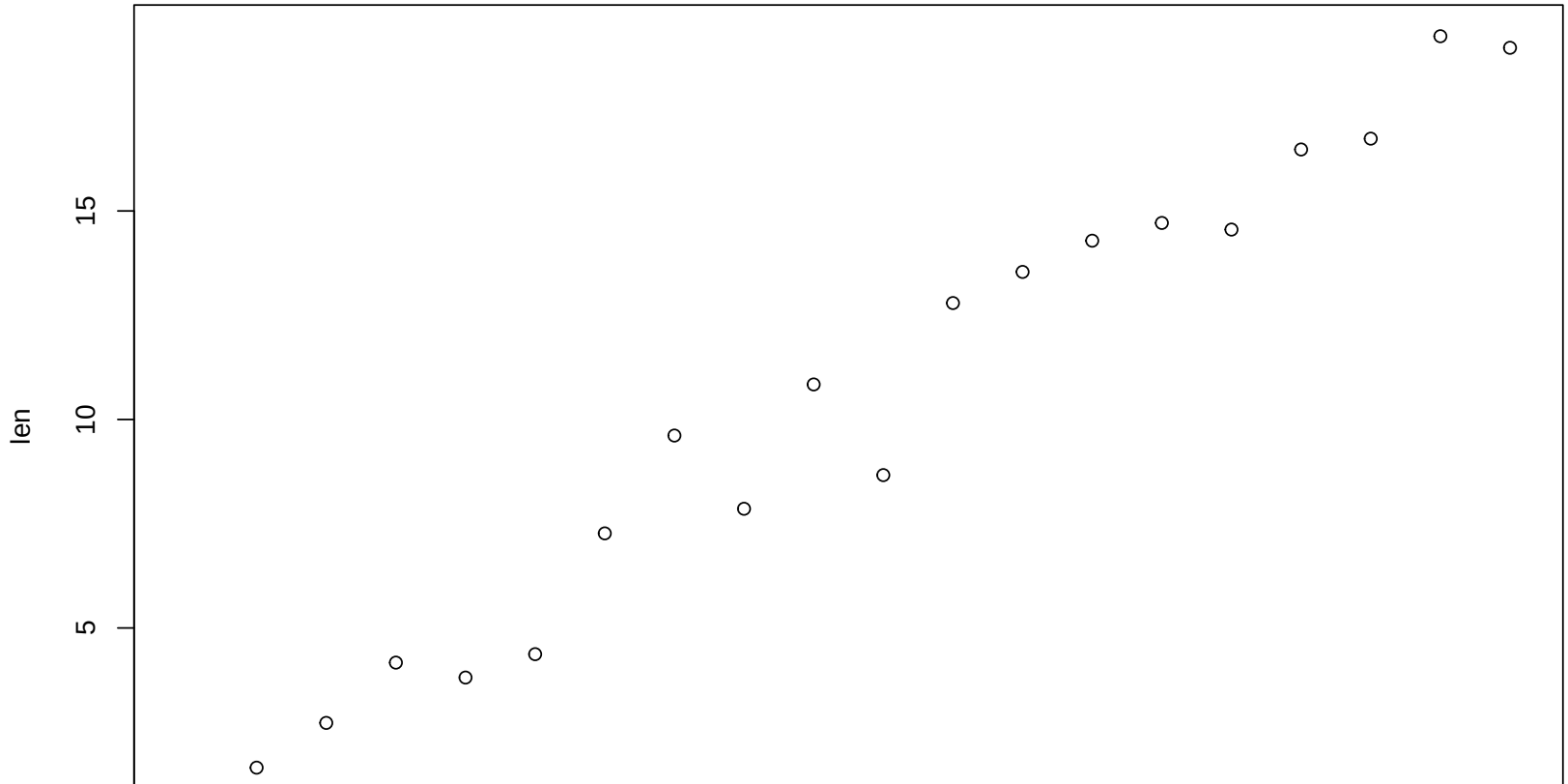
```
plot(my.data$age, my.data$len)
```



Basic plotting, model syntax

In R, the syntax to show one parameter as dependent on another is to use the ~, *ie.* `y ~ x`. This can be used when specifying what to plot too:

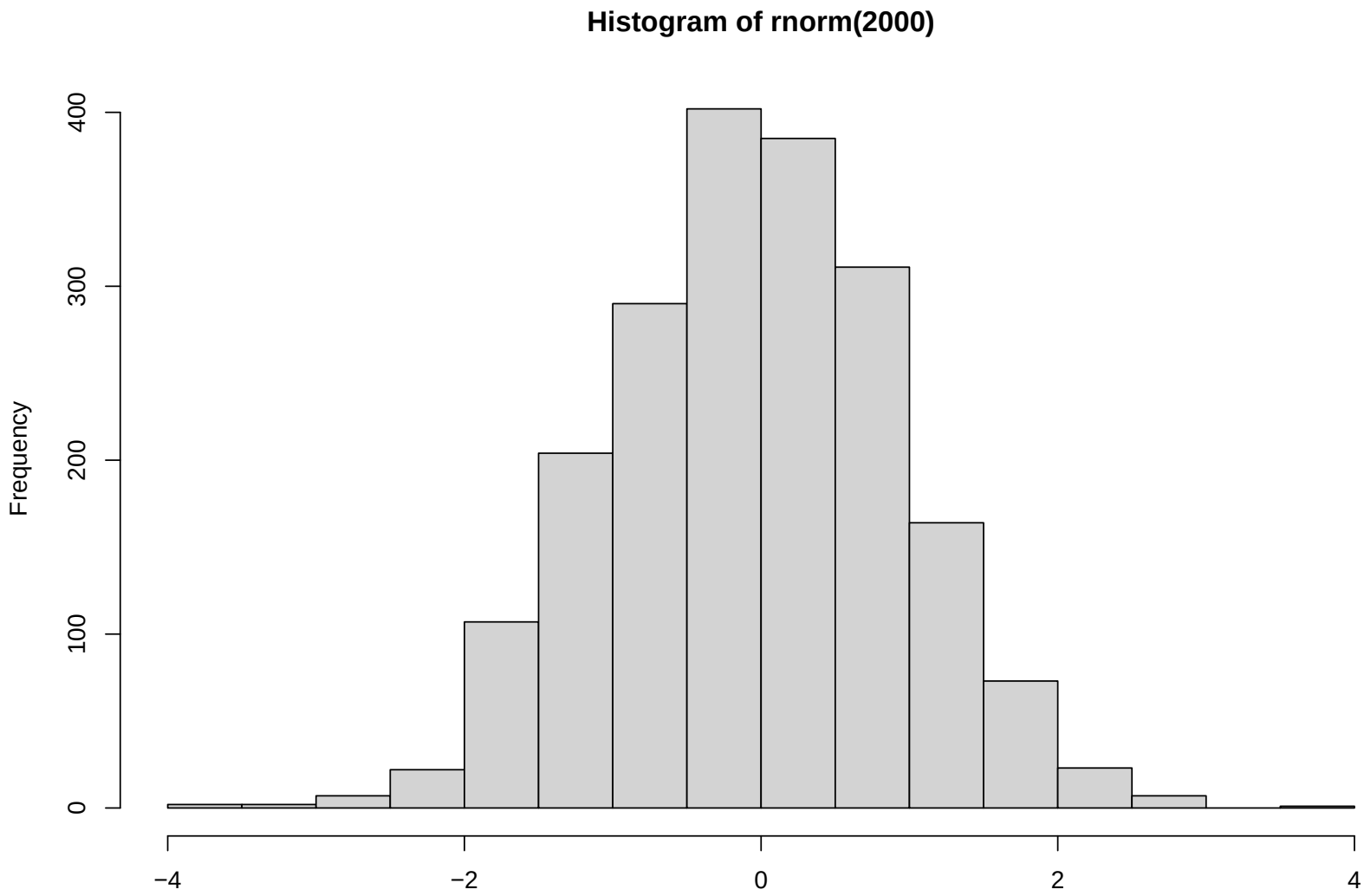
```
plot(len ~ age, data = my.data)
```



Basic plotting - histogram

Similarly, we have the `hist` function to create histograms:

```
hist(rnorm(2000))
```



Installing packages

Base R is nothing without packages. Package installation is trivial.

- Use `install.packages("package name")` function, or
- Use the package installation menu or RStudio/R

Once the package is installed, it has to be enabled with the using `library()`. The full workflow would look something like:

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

Keeping track of your work

It is important to always keep notes on what you have done.
RStudio provides a very handy tool for this with the R Notebook.
In an R notebook, you can intermix R code with running text to continuously explain what you have done.

Exercises

Create an R Notebook where describe how you do the following steps:

- Create one vector with 20 random numbers from a uniform distribution. (Use help to search for the function)
- Create another vector that is a function of the first vector with some added gaussian noise.
- Create a third factorial vector with 20 values
- Create a dataframe consisting of these three vectors
- Plot the two first vectors against each other