Lab: Introduction to R

Processing Big Data - 7CS516

Gkratsia Tantilian

04/03/2022

Basic Commands - Variables

- Assigning a value to a variable

```
x <- 5
```

- Getting the value of a variable

```
print(x)
```

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

- We can also assign a value using `=` instead of `<-`
- We can use the *Up Arrow* in R-console to display previously used commands

Basic Commands - Functions

- *R* uses functions to perform operations
- To run a function: `funcname(input1, input2)`
- A useful built-in function to *concatenate* values into one vector: `c()`

Example:

```
# Assign a 1d vector to the variable x
x <- c(1,3,2,5)
```

- To get more information about a function: `?funcname`
- Defining a new function

```
mySquare <- function(x) {
  return(x*x)
}
# Call the function
mySquare(2)
```

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

Basic Commands - Vectors

- Create two vectors

```
x <- c(1,3,2,5)
y <- c(3,1,2,0)
```

- We can add the vector elements but they must be of the same *length*

```
length(x) == length(y)
```

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

```
x + y
```

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

Basic Commands - Matrices

- For details see `?matrix`
- Creating a matrix

```
x <- matrix(data=c(1,2,3,4), nrow=2, ncol=2)
```

- Accessing the matrix

```
print(x)
```

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

```
print(x[1, 2])
```

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

Basic Commands - Matrices

- Accessing sub-matrix

```
x <- matrix(data=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16), nrow=4, ncol=4)
print(x)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

```
print(x[2:3, 2:4])
```

```
##      [,1] [,2] [,3]
## [1,]    6   10   14
## [2,]    7   11   15
```

Basic Commands - Matrices

- Accessing sub-matrix

```
x <- matrix(data=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16), nrow=4, ncol=4)
print(x)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

```
print(x[, 2:4])
```

```
##      [,1] [,2] [,3]
## [1,]    5    9   13
## [2,]    6   10   14
## [3,]    7   11   15
## [4,]    8   12   16
```

Basic Commands - Matrices

- Applying a function on a matrix

```
sqrt(x)
```

```
##          [,1]     [,2]     [,3]     [,4]
## [1,] 1.000000 2.236068 3.000000 3.605551
## [2,] 1.414214 2.449490 3.162278 3.741657
## [3,] 1.732051 2.645751 3.316625 3.872983
## [4,] 2.000000 2.828427 3.464102 4.000000
```

- Square all the elements

```
x^2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   25   81  169
## [2,]    4   36  100  196
## [3,]    9   49  121  225
## [4,]   16   64  144  256
```

Basic Commands - Matrices

- Dimensions of a matrix

```
dim(x)
```

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

- Question: What is `byrow` for?

```
matrix(data=c(1,2,3,4), nrow=2, ncol=2, byrow=TRUE)
```

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

```
matrix(data=c(1,2,3,4), nrow=2, ncol=2)
```

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

Basic Commands - Memory

- Check what variables are in memory

```
ls()
```

```
## [1] "mySquare" "x"        "y"
```

- What is the size of each object

```
object.size(x)
```

```
## 344 bytes
```

- Free memory by deleting variables

```
rm(x, y, mySquare)
ls()
```

```
## character(0)
```

Basic Commands - Random Numbers

- Generating random numbers from a normal distribution

```
x <- rnorm(50)
```

- Creating correlated variable based on x

```
y <- x + rnorm(50, mean = 50, sd = 0.1)
```

- Finding the correlation between x and y

```
cor(x, y)
```

```
## [1] 0.9958071
```

- Seeding the random generator to get the same pseudo-random numbers

```
set.seed(2022)
x <- rnorm(50)
```

Basic Commands - Random Numbers

- Statistical measures of a vector of numbers

```
# Mean Value
mean(x)
```

```
## [1] -0.1288761
```

```
# Variance of x
var(x)
```

```
## [1] 0.7627134
```

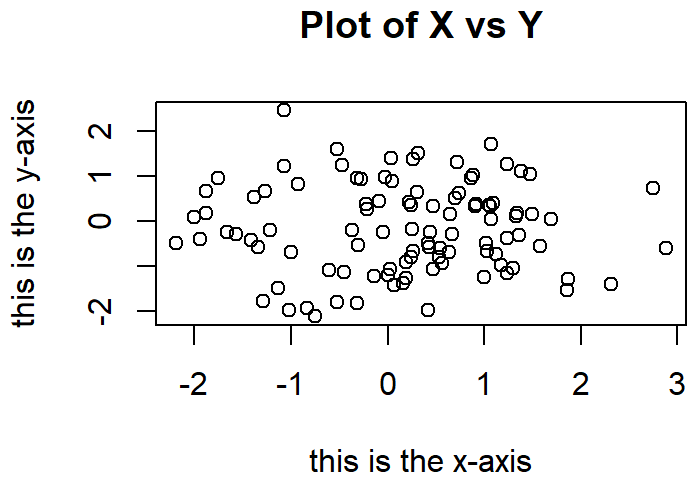
```
# Standard Deviation of x
sd(x)
```

```
## [1] 0.8733347
```

Graphics - Plot

Plotting data using the built-in `plot` function. For details see `?plot`.

```
x <- rnorm(100)
y <- rnorm(100)
plot(x, y,
     xlab="this is the x-axis", ylab="this is the y-axis", main="Plot of X vs Y")
```



Graphics - Contour

- Export the graphs using the `Plots` area of R Studio, or using `pdf()` or `jpeg()` functions.
- Create a sequence of numbers using `seq()` or `:`

```
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
1:10
```

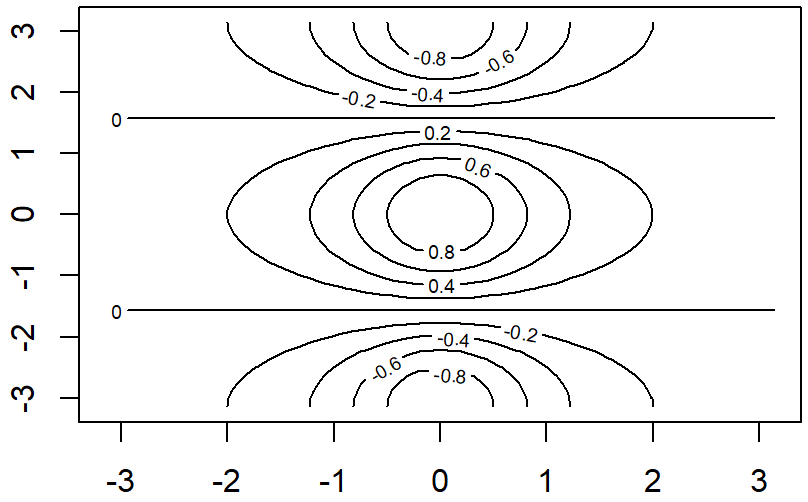
```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# Preparing Three dimensional data
x <- seq(-pi, pi, length=50) -> y
Y <- function(x, y) {
  return(cos(y) / (1 + x^2))
}
z <- outer(x, y, Y)
```

Graphics - Contour

- Plotting three dimensional data using `contour()`

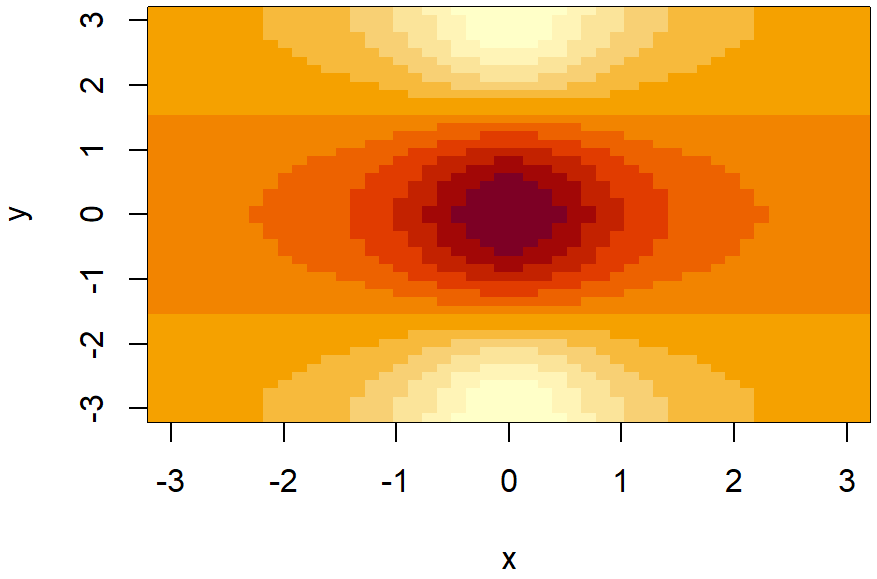
```
contour(x, y, z)
```



Graphics - Image

- Plotting three dimensional data using `image()`

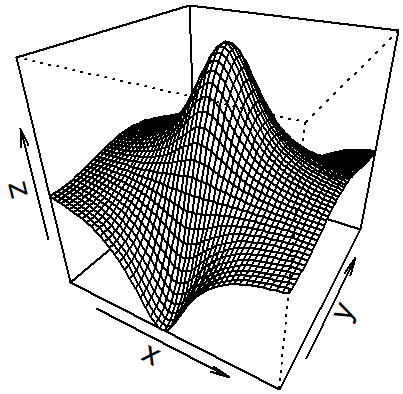
```
image(x, y, z)
```



Graphics - Persp

- Plotting three dimensional data using `persp()` (defining the perspective of the viewer)

```
persp(x, y, z, theta = 30, phi = 30)
```



Loading data

- In `R` we can load data in a table format using `read.table` or `read.csv`
- For details see `?read.table`
- Similarly we can save our data using `write.table` or `write.csv`

```
library(ISLR2)
df <- Auto
write.table(df, 'Auto.csv')
```

Exploring data

- Use `fix(df)` to explore the data visually
- Use `names(df)` to check what are the column names in your dataframe
- Use `summary(df)` to see a column-wide data summary

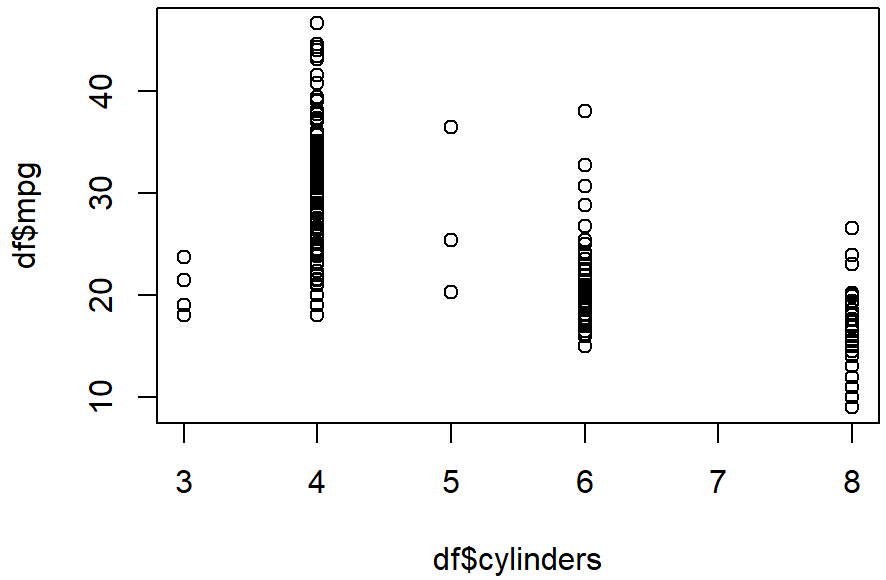
```
summary(df)
```

```
##       mpg          cylinders      displacement     horsepower        weight
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2225
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5   Median :2804
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5   Mean   :2978
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0   3rd Qu.:3615
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140
##
##   acceleration        year           origin                      name
##  Min.   : 8.00   Min.   :70.00   Min.   :1.000   amc matador       :  5
##  1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
##  Median :15.50   Median :76.00   Median :1.000   toyota corolla    :  5
##  Mean   :15.54   Mean   :75.98   Mean   :1.577   amc gremlin       :  4
##  3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
##  Max.   :24.80   Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
##                                                  (Other)           :365
```

More on plot

Plot data using columns of a dataframe

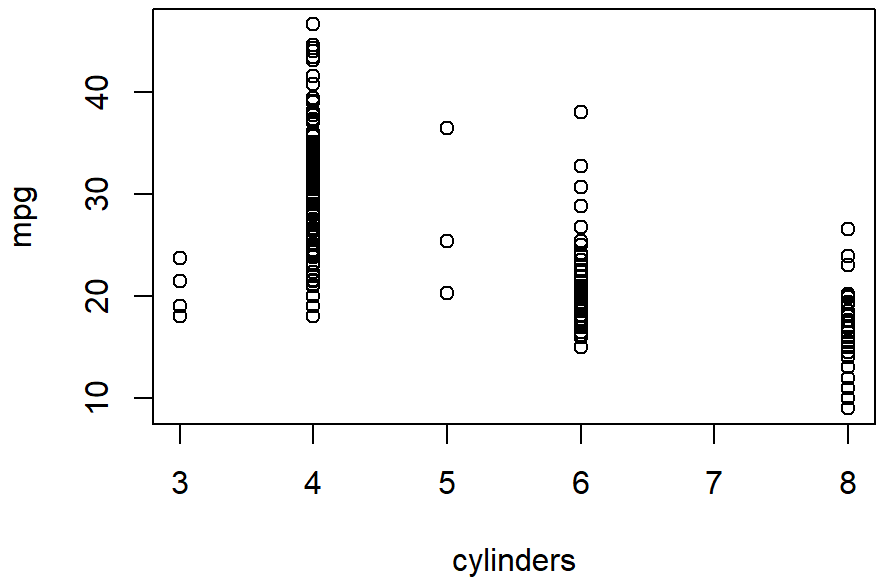
```
plot(df$cylinders, df$mpg)
```



More on plot

Alternatively attach dataframe as main data source

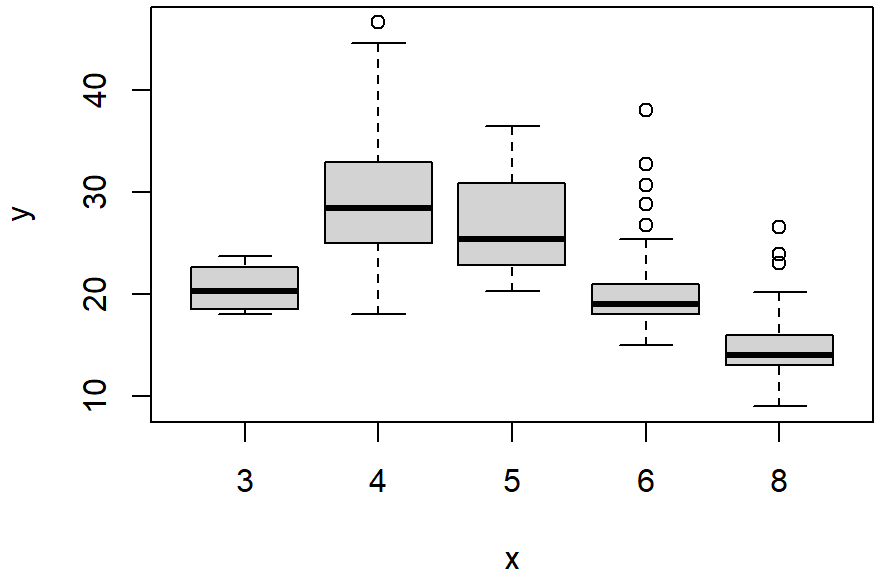
```
attach(df)
plot(cylinders, mpg)
```



More on plot

The `as.factor()` function converts quantitative variables into qualitative variables. If the variable plotted on the x-axis is categorial, then boxplots will automatically be produced by the plot() function.

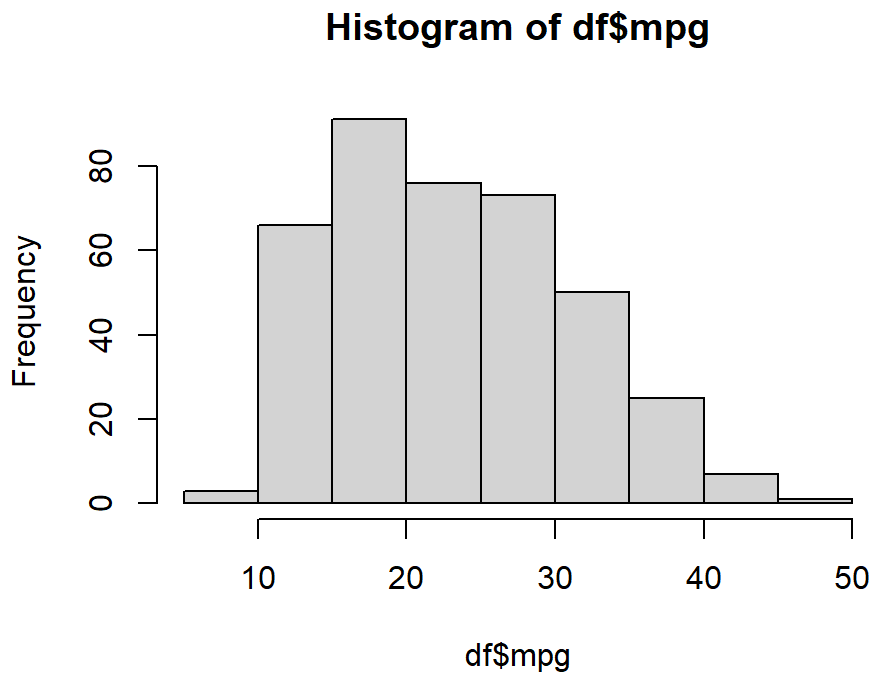
```
cylinders <- as.factor(df$cylinders)
plot(cylinders, df$mpg)
```



More on plot

We can create histograms of a categorical column using `hist()`

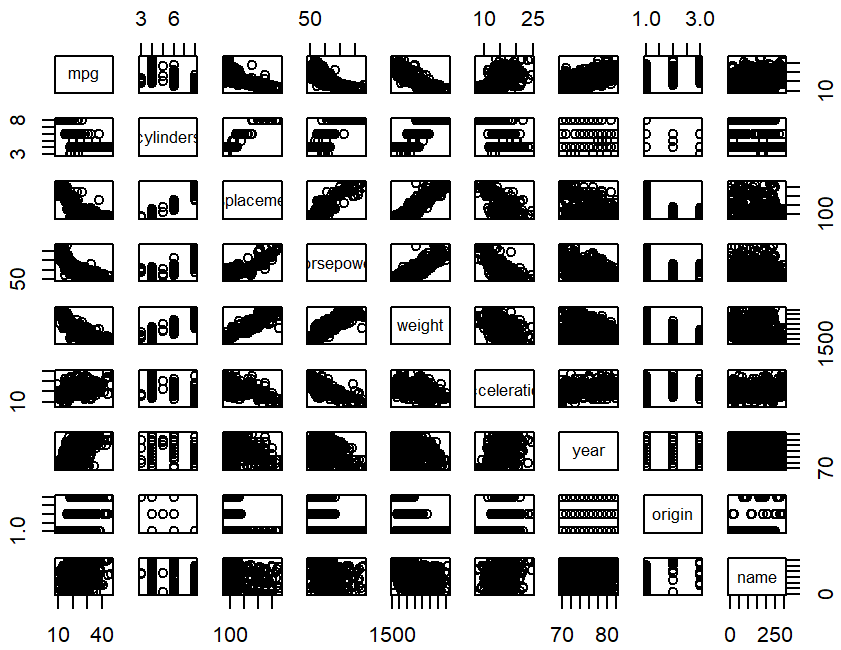
```
hist(df$mpg)
```



More on plot

The `pairs()` function creates a scatterplot matrix i.e. a scatterplot for every pair of variables for any given data set.

```
pairs(df)
```



More on plot

We can also produce scatterplots matrix for just a subset of the variables.

```
pairs(~ + mpg + displacement + horsepower + weight + acceleration, df)
```