Homework 1

A note on R Markdown

This document is written in a format called R Markdown. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com. It allows us to easily create a pdf file with the exercises, some readable code to give to you, and a nice html file with the solutions and outputs. It will also give you an idea on how to print nice html or pdf reports with R.

After you receive the solutions, you will see that the R code for the solutions are so-called code chunks. When you click the **Knit** button in RStudio ("Knit HTML"), a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. If you want to create a pdf file, you will need to install LaTeX to your computer.

We hope that this will be both easier for you to read in RStudio and easier to save or print.

Prelimiaries

The homework tasks are designed 1) to help you exercise the new material that we discuss extensively in the lecture and exercise classes and 2) to lead you towards solving more complex technical challenges armed with the theoretical knowledge from the lecture. During the R introduction sessions, we will work through a set of exercises in class before you receive a homework assignment like this for repetition. Later on during the class, you will receive only one homework exercise, which you must work on during the week before we discuss it in class. A large part of learning the *applied* part of machine learning and data science is *application*, so your own effort will largely determine your learning results.

Data scientists don't usually work in R markdown files like this one. R Markdown is a good way to produce a report that also displays code and results, but having text and code chunks can get messy.

When coding for yourself, we advise you to work in simple .R code files (File > New > R Script).

Variables and classes

1. Create two variables a and b and assign values of 3 and 4.5.
2. Query the type of variable a.
3. Check whether variable b is a text variable (of class character).
4. Calculate $a^2 + \frac{1}{b}$, $\sqrt{a * b}$, and $log_2(a)$.

```
## [1] "numeric"
```

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

```
## [1] 9.222222
```

```
## [1] 3.674235
```

```
## [1] 1.584963
```

```
## [1] 1.584963
```

Matrix algebra

Create three additional variables as follows:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix} \quad y = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

Calculate

1. $a * A$
2. $A * B$
3. The inverse of matrix A and store the result in a variable $invA$
4. Multiply A and $invA$ and verify that the result is the identity matrix (i.e. only 1s on the diagonal). You'll probably find that it isn't, because computers usually make very small rounding error when handling real numbers. The reason is interesting, but you'll have to look it up if you're interested.
5. The transpose of matrix B
6. Fill the first row of matrix B with ones
7. Calculate the ordinary least squares estimator β (i.e. a standard regression)

$$\beta = (A^\top A)^{-1} A^\top y$$

```
##      [,1] [,2] [,3]
## [1,]    3    6    9
## [2,]   12   15   18
## [3,]   21   24   30
```

```
##      [,1] [,2] [,3]
## [1,]   14   32   50
## [2,]   32   77  122
## [3,]   53  128  203
```

```
##      [,1]         [,2]          [,3]
## [1,]    1 8.881784e-16 -4.440892e-16
## [2,]    0 1.000000e+00 -1.776357e-15
## [3,]    0 0.000000e+00  1.000000e+00
```

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

```
##               [,1]
## [1,] -3.333333e-01
## [2,]  6.666667e-01
## [3,] -5.684342e-14
```

Indexing

1. Look at values of variables A, B, and y from the last exercise
2. Access the second element in the third row of A and the first element in the second row of B, and compute their product
3. Multiply the first row of A and the third column of B
4. Access the elements of y that are greater than 1 (without looking up their position manually)
5. Access the elements of A in the second column, for which the values in the first column are greater or equal to 4)
6. Access the 4th row of A. If this returns an error message, use Google to investigate the problem and find out what went wrong.

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

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

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

```
## [1] 16
```

```
## [1]  1 16 27
```

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

```
## [1] 5 8
```

Custom functions

For many statistical applications, including model training, it is practical to standardize variable values. One way to standardize is *centering and scaling*. In simple words, we make the variables comparable by reducing them to the same scale.

1. Start writing a custom R function by assigning **function()** to an object **standardize**. The input to the function must be specified in the brackets. This function should take an argument **x**. To keep things simple, we expect x to be a numeric vector.
2. After the closing bracket, the body of the function is usually enclosed in curly brackets because it encompasses several lines. In the body of the function, calculate the mean and standard deviation of **x** and save them to the objects **mu** and **std** respectively. Then for each element in the vector, substract the mean and divide by the standard deviation.
3. In the body of the function, **return** the standardized vector. This tells R that the function is done and returns the argument to **return()**.
4. You should always test your functions. Create a vector **a** with the elements (-100, -25, -10, 0, 10, 25, 100).
5. Let's include a simple check in the function. Before doing any calculations, use **if()** and **is.numeric()** to check if the input is a numeric vector. If not, skip the computations and simply output the original vector.

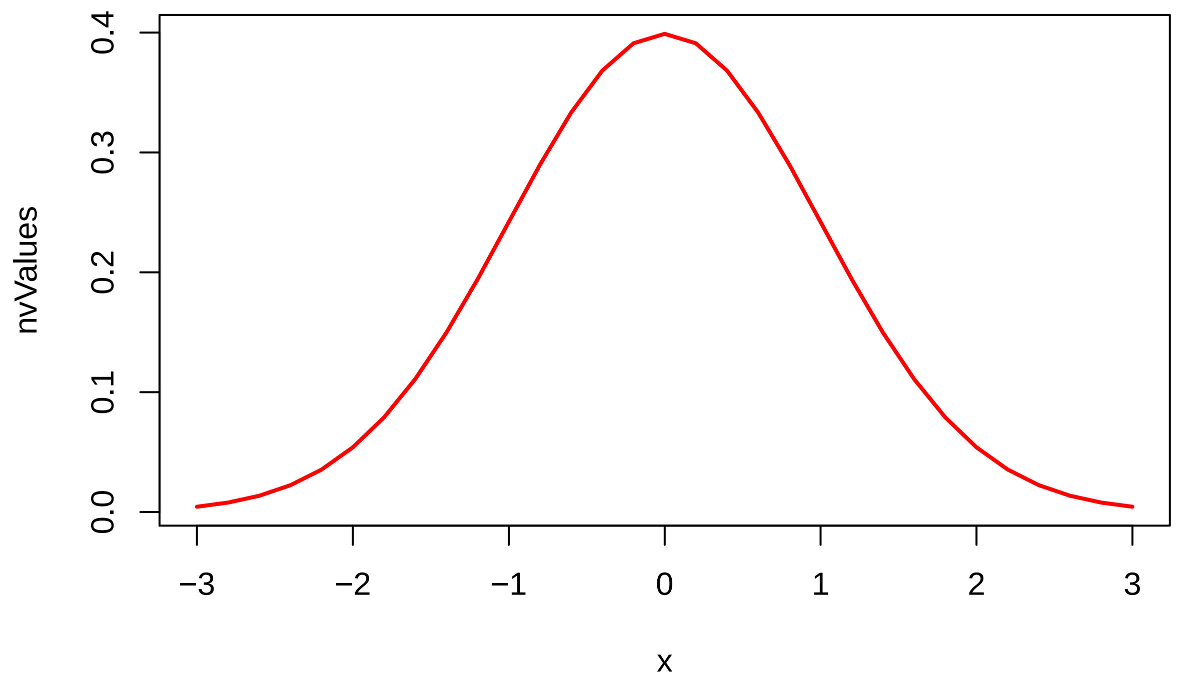
```
## [1] -1.6724840 -0.4181210 -0.1672484  0.0000000  0.1672484  0.4181210
## [7]  1.6724840
```

Using inbuilt functions

The density of the normal distribution with expected value μ and variance σ is given as

$$f(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

1. Let's make a nice plot of the bell shape that is so famous. The function **dnorm(x)** outputs the relative probability (or density) for a value x that the value of a normally distributed random variable would be arbitrarily close to that value. Check out the help to find out how to compute values of the standard normal distribution for some values **x** using this function.
2. Calculate the values of the probability density function (with $\mu = 0$ and $\sigma = 1$) at $x = -3, -2.8, \ldots, +3$ and store the results in a variable **nvValues**
3. Create a simple graph of the resulting values using the **plot()** function. Use the R help and Google (do it twice) to find out how to plot a line instead of dots and change the color to red.



Well done!