Exercise 4

Business Analytics and Data Science WS16/17

Asking for help

We now come to a point where issues like errors and warnings will become more frequent and where you might want to ask for help when things won't work. This is great, remember that you have your classmates, your assignment group, the tutorial class, and your teachers to help you out - preferably in this order. We enjoy hard questions. In order to ask for help efficiently, learn the following steps by heart:

1. Try to understand the error or warning message.
2. Check if the objects that where saved before the error have the expected format and values.
3. Check for typos or errors in your logic.
4. Check the function help.
5. Google the error message or your question. I highly recommend Stackoverflow.com .

At this point, maybe you need some outside help. No matter who you aks, they will need the following information from you:

1. What you were trying to do and how you were trying to do it.
2. The exact problem that occured and the exact error message.
3. What you have tried to solve the problem and why it didn't work.
4. A reproducable example from your code. Don't send the whole code, just the parts that are needed to create the error (see the FAQ on stackoverflow). You can copy/paste objects via **dput()**.

Loading scripts and automating data cleaning

You should have created a script to automatically load the loans data set in one of the homework exercises. We can now use this script to quickly load and clean the data. If you haven't created the script yet, please take the time to do it now.

1. Use function **source()** to load and execute the script *helperfunctions.R* containing your custom function. The function should now be visible in your R environment. You can check by calling **get.loan.dataset** without the brackets.
2. Use your custom function **get.loan.dataset()** to load and clean the data and save the resulting data frame to an object **loans**.

Clustering works by calculating the distance between the observations. Because distance calculations are more complicated (although not impossible) if they include data that is not numeric, for example your field of study, we will restrict ourselves to the numeric variables for this exercise. 3. Save the indices (=column number) of all numeric variables in the data to a vector *idx_numeric*. Don't do this by hand - it may work here, but it won't work when your data comrpises 200 variables.

```
##       YOB            nKIDS             nDEP              PHON
##  Min.   : 3.00   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:42.00   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:1.0000
##  Median :55.00   Median :0.0000   Median :0.00000   Median :1.0000
##  Mean   :50.84   Mean   :0.6237   Mean   :0.03837   Mean   :0.9037
##  3rd Qu.:63.00   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:1.0000
##  Max.   :69.00   Max.   :5.0000   Max.   :2.00000   Max.   :1.0000
##
##     dINC_SP          EMPS_A        dINC_A      RES         dHVAL
##  Min.   :    0   P      :531   Min.   :    0   F:129   Min.   :    0
```

```
##  1st Qu.:    0   V      :231   1st Qu.: 9000   N: 66   1st Qu.:    0
##  Median :    0   E      :124   Median :19500   O:624   Median :    0
##  Mean   : 1990   T      :123   Mean   :21244   P:252   Mean   :15694
##  3rd Qu.: 1040   R      :104   3rd Qu.:30600   U:154   3rd Qu.:28928
##  Max.   :50000   W      : 37   Max.   :64800           Max.   :64928
##                  (Other): 75
##       dMBO           dOUTM          dOUTL             dOUTHP
##  Min.   :    0   Min.   :   0   Min.   :    0.0   Min.   :   0.00
##  1st Qu.:    0   1st Qu.:   0   1st Qu.:    0.0   1st Qu.:   0.00
##  Median :    0   Median : 256   Median :    0.0   Median :   0.00
##  Mean   :11226   Mean   : 342   Mean   :  121.9   Mean   :  28.72
##  3rd Qu.:20000   3rd Qu.: 528   3rd Qu.:    0.0   3rd Qu.:   0.00
##  Max.   :64000   Max.   :3800   Max.   :28000.0   Max.   :1600.00
##
##      dOUTCC         BAD       YOB_missing
##  Min.   :   0.0   good:902   Min.   :0.000000
##  1st Qu.:   0.0   bad :323   1st Qu.:0.000000
##  Median :   0.0              Median :0.000000
##  Mean   :  39.6              Mean   :0.005714
##  3rd Qu.:   0.0              3rd Qu.:0.000000
##  Max.   :2800.0              Max.   :1.000000
##
```

k-means Clustering

Clustering is a popular approach for unsupervised learning. A standard method used in many data mining applications is k-means clustering.

1. Cluster the data into 5 groups using the k-means algorithm and increase the maximum number of iterations to 50. Look at the *structure* of the result object and extract a vector **clusters** indicating the cluster identity for each observation. Note that the standard k-means algorithm only works for numeric variables, so you will have to select these.
2. The k-means algorithm requires that you specify the number of clusters beforehand. We can empirically test which number of clusters will give the best 'fit'. Let's say we want to test between 1 and 15 clusters using a loop. To loop over the number of clusters, k, create a vector **k.settings** with the values 1 to 15. Also create an empty vector **obj.values** with the length of **k.settings** to store the results. Then, loop over the numer of values in **k.settings** and, for each **i**, perform the following steps in the body of the for-loop:
    1. Calculate the k-means clusters for the number of clusters given in **k.settings**[**i**] and save the results in an object *cluster solution* **clu.sol**.
    2. This object is a list with, among others, an element **tot.withinss**. Extract the within-cluster sum-of-squares from **clu.sol** and save the result to the result vector at position i **obj.values**[**i**].
3. Plot the results with the number of clusters on the x-axis and the within-cluster sum-of-squares on the y axis. What is the optimal number of clusters according to the elbow criterion?

```
## List of 9
##  $ cluster     : int [1:1225] 1 5 5 1 1 2 2 2 5 4 ...
##  $ centers     : num [1:5, 1:13] 52.5 49.9 48.1 45.3 53.8 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:5] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:13] "YOB" "nKIDS" "nDEP" "PHON" ...
##  $ totss       : num 1.3e+12
##  $ withinss    : num [1:5] 3.59e+10 6.24e+10 7.86e+10 1.01e+11 4.76e+10
##  $ tot.withinss: num 3.25e+11
```

```
##  $ betweenss    : num 9.76e+11
##  $ size         : int [1:5] 370 132 147 209 367
##  $ iter         : int 4
##  $ ifault       : int 0
##  - attr(*, "class")= chr "kmeans"
```