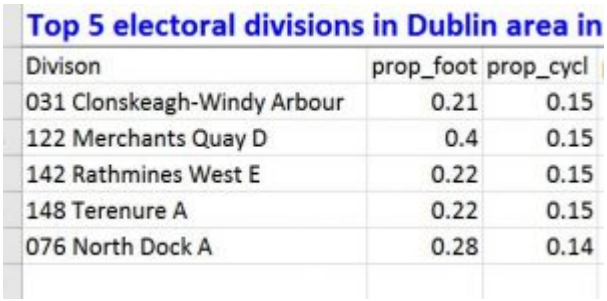
Working with the xlsx package Exercises (part 2)



Top 5 electoral divisions in Dublin area in		
Divison	prop_foot	prop_cycl
031 Clonskeagh-Windy Arbour	0.21	0.15
122 Merchants Quay D	0.4	0.15
142 Rathmines West E	0.22	0.15
148 Terenure A	0.22	0.15
076 North Dock A	0.28	0.14

This exercise set provides (further) practice in writing Excel documents using the xlsx package as well as importing and general data manipulation. Specifically we have loops in order for you to practice scaling. A previous exercise set focused on writing a simple sheet with the same package, see here.

We will use a subset of commuting data from the Dublin area from AIRO and the 2011 Irish census.

Solutions are available here.

Exercise 1
Load the xlsx package. If necessary install it as indicated in the previous xlsx exercise set.

Exercise 2
Download the data to your computer and read into your R workspace as commuting using read.xlsx2() or the slower alternative read.xlsx(). Use colClasses to set relevant classes as we will be manipulating the data later on.

Exercise 3
Clean the data a bit by removing 'Population_Aged_5_Over_By_' and 'To_Work_School_College_' from the column names.

Exercise 4
Sum the 'population aged 5 and over' variables by electoral division name using for instance aggregate() or data.table and

save the result as commuting_ed.

Learn more about working with excel and R in the online course Learn By Example: Statistics and Data Science in R. In this course you will learn how to:

- Learn some of the differences between working in Excel with regression modelling and R
- Learn about different statistical concepts
- And much more

Exercise 5
Create an xlsx workbook object in your R workspace and call it wb.

Exercise 6
Create three sheets objects in wb named sheet1, sheet2, sheet3 in wb and your workspace. Use a loop.

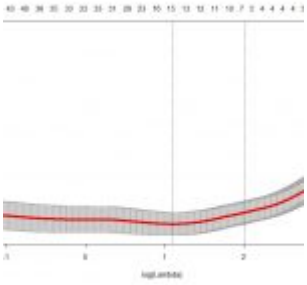
Exercise 7
Make a data.frame that lists proportion of respondents in each of the following category by electoral division: travel on foot, travel on bicycle, leave home before 6:30.

Exercise 8
Add the top 5 electoral division in each category to a previously created sheets with all the proportions using a loop. Leave the first row free.

Exercise 9
Add some great title to the first row of each sheet and apply some style to it.

Exercise 10
Save your workbook to your working directory and open using Excel. Go back to R and continue formatting and adding information to your workbook at will.

Data Visualization with googleVis exercises part 4



Adding Features to your Charts

We saw in the previous charts some basic and well-known types of charts that googleVis offers to users. Before continuing with other, more sophisticated charts in the next parts we are going to "dig a little deeper" and see some interesting features of those we already know.

Read the examples below to understand the logic of what we are going to do and then test yous skills with the exercise set we prepared for you. Lets begin!

Answers to the exercises are available here.

Package & Data frame

As you already know, the first thing you have to do is install and load the googleVis package with:

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

Secondly we will create an experimental data frame which will be used for our charts' plotting. You can create it with:

```
df=data.frame(name=c("James", "Curry", "Harden"),
Pts=c(20,23,34),
Rbs=c(13,7,9))
```

NOTE: The charts are created locally by your browser. In case they are not displayed at once press F5 to reload the page.

Customizing Chart

We are going to use the two-axis Line Chart we created in part 1. This is the code we used, in case you forgot it:

```
LineC2 <- gvisLineChart(df, "name", c("Pts","Rbs"),
options=list(
series="[{targetAxisIndex: 0},
{targetAxisIndex:1}]",
vAxes="[{title:'Pts'}, {title:'Rbs'}]"
))
plot(LineC2)
```

Colours

To set the color of every line we can use:

```
series="[{color:'green', targetAxisIndex: 0,
```

Exercise 1

Change the colours of your line chart to green and yellow respectively and plot the chart.

Line Width

You can determine the line width of every line with:

```
series="[{color:'green',targetAxisIndex: 0, lineWidth: 3},
```

Exercise 2

Change the line width of your lines to 3 and 6 respectively and plot the chart.

Dashed lines

You can tranform your lines to dashed with:

```
series="[{color:'green', targetAxisIndex: 0,
lineWidth: 1, lineDashStyle: [2, 2, 20, 2, 20, 2]},
```

There are many styles and colours available and you can find them here.

Learn more about using GoogleVis in the online course Mastering in Visualization with R programming. In this course you will learn how to:

- Work extensively with the GoogleVis package and its functionality
- Learn what visualizations exist for your specific use case
- And much more

Exercise 3

Choose two different styles of dashed lines for every line of your chart from the link above and plot your chart.

Point Shape

With the pointShape option you can choose from a variety of shapes for your points.

We will use the scatter chart we built in part 3 to see how it works. Here is the code:

```
ScatterCD <- gvisScatterChart(cars,
options=list(
legend="none",
pointSize=3,lineWidth=2,
title="Cars", vAxis="{title:'speed'}",
hAxis="{title:'dist'}",
width=600, height=300))
plot(ScatterCD)
```

Exercise 4

Change the shape of your scatter chart’s points to ‘square’ and plot it. **HINT**: Use pointShape.

Exercise 5

Change the shape of your scatter chart's points to 'triangle', their point size to 7 and plot it.

Edit Button

A really useful and easy feature that googleVis provides is the edit button which gives the user the ability to customize the chart in an automated way.

```
options=list(gvis.editor="Edit!"))
```

Exercise 6

Add an edit button in the scatter chart you just created. **HINT**: Use gvis.editor.

Chart with more options

Now let's see how we can create a chart with many features that can enhance its appearance. We will use again the 2-axis line that we used before.

```
LineCD2 <- gvisLineChart(df, "name", c("Pts","Rbs"),
options=list(
series="[{color:'green',targetAxisIndex: 0, lineWidth: 3,
lineDashStyle: [14, 2, 2, 7]},
{color:'yellow',targetAxisIndex:1,lineWidth: 6,
lineDashStyle: [10, 2]}]",
vAxes="[{title:'Pts'}, {title:'Rbs'}]"
))
plot(LineCD2)
```

Background color

You can decide the background color of your chart with:

```
backgroundColor="red",
```

Exercise 7

Set the background color of your line chart to "lightblue" and

plot it. **HINT**: Use backgroundColor.

Title

To give a title and decide its features you can use:
title="Title",
titleTextStyle="{color:'orange',
fontName:'Courier',
fontSize:14}",

Exercise 8

Give a title of your choise to the line chart and set its font to blue, Courier of size 16. **HINT**: Use titleTextStyle.

Curve Type & Legend

Another nice-looking choise that googleVis gives you is to display the lines like curves with:
curveType="function"

You can also move the legend of your chart to the bottom with:
legend="bottom"

Exercise 9

Smooth the lines of your line chart by setting the curveType option to function and move the legend to the bottom. **HINT**: Use curveType and legend.

Axes features

Finally you can “play” with your axes. This is an example:
vAxis="{gridlines:{color:'green', count:4}}",
hAxis="{title:'City', titleTextStyle:{color:'red'}}",
series="[{color:'yellow', targetAxisIndex: 0},
{color: 'brown',targetAxisIndex:1}]",
vAxes="[{title:'val1'}, {title:'val2'}]",

Exercise 10

Give the title “Name” to your hAxis and color it orange. Separate your vAxis with 3 red gridlines. **HINT**: Use titleTextStyle and gridlines

---

Bonus: Improve Data Consistency With Vapply()

```
##   x  y  c
## 1 -2 -2 -2
## 2 -1 -1 -1
## 3  0  0  0
## 4  1  1  1
## 5  2  2  2
```

The vapply() function improves consistency via pre-specification of return value. Speed is also improved.

The usage of vapply():
vapply(X, FUN, FUN.VALUE, ..., USE.NAMES = TRUE)

Answers to the exercises are available here.

Exercise 1
Beginning Level-

The dataframe used for this exercise:
dataset1 <- data.frame(observationA = 16:8, observationB = c(20:19, 6:12))

Using vapply(), find the length of dataset1‘s observations.

Exercise 2

Beginning Level

Find the mean of dataset1‘s observations.

Exercise 3

Beginning Level

Using vapply(), find the sums of dataset1‘s observations.

Learn more about the apply family in the online course R Programming: Advanced Analytics In R For Data Science. In this course you will learn how to:

- Work with all common data types like dates, integers and characters
- Indepth use and analysis of the apply family of functions
- And much more

Exercise 4

Intermediate Level

Find the class of dataset1‘s observations.

Exercise 5

Intermediate Level

Use vapply() to verify all the “mtcars” columns are numeric.

Exercise 6

Intermediate Level

Find the range of dataset1.

Exercise 7

Intermediate Level

Print dataset1 with the vapply() function.

Exercise 8

Advanced Level

Find the quantiles of dataset1, with the vapply() function.

Exercise 9

Advanced Level

Find the number of characters in the following dataset of strings:
cars <- c("Corolla", "Firebird", "Europa")

Exercise 10

Advanced Level

Using the following function, process dataset1's observations with toValue set to FALSE.

Required function:

```
convert <- function(x, toValue=TRUE) {
if (toValue) { x = x * 25.4} else {
x = x / 25.4}
return(x)
}
```

Data wrangling : Reshaping

Data wrangling is a task of great importance in data analysis. Data wrangling, is the process of importing, cleaning and transforming raw data into actionable information for analysis. It is a time-consuming process which is estimated to take about 60-80% of analyst's time. In this series we will go through this process. It will

be a brief series with goal to craft the reader's skills on the data wrangling task. This is the second part of this series and it aims to cover the reshaping of data used to turn them into a tidy form. By tidy form, we mean that each feature forms a column and each observation forms a row.

Before proceeding, it might be helpful to look over the help pages for the spread, gather, unite, separate, replace_na, fill, extract_numeric.

Moreover please load the following libraries.

```
install.packages("magrittr")
library(magrittr)
install.packages("tidyr")
library(tidyr)
```

Please run the code below in order to load the data set:

```
data <- airquality[4:6]
```

Answers to the exercises are available here.

If you obtained a different (correct) answer than those listed on the solutions page, please feel free to post your answer as a comment on that page.

Exercise 1

Print out the structure of the data frame.

Exercise 2

Let's turn the data frame in a wider form, from above and and turn the Month variable into column headings and spread the Temp values across the months they are related to.

Exercise 3

Turn the wide (exercise 2) data frame into its initial format using the gather function, specify the columns you would like to gather by index number.

Exercise 4

Turn the wide (exercise 2) data frame into its initial format using the gather function, specify the columns you would like to gather by column name.

Learn more about Data Pre-Processing in the online course R Data Pre-Processing & Data Management – Shape your Data!. In this course you will learn how to:

- import data into R in several ways while also beeing able to identify a suitable import tool
- use SQL code within R
- And much more

Exercise 5

Turn the wide (exercise 2) data frame into its initial format using the gather function, specify the columns by using remaining column names(the ones you don't use for gathering).

Exercise 6

Unite the variables Day and Month to a new feature named Date with the format %d-%m .

Exercise 7

Create the data frame at its previous format (exercise 6). Separate the variable you have created before (Date) to Day, Month.

Exercise 8

Replace the missing values (NA) with 'Unknown'.

Exercise 9

Run the script below, so that you make a new feature year.

```
back2long_na$year <- rep(NA, nrow(back2long_na))

back2long_na$year[1] <- '2015'
back2long_na$year[as.integer(nrow(back2long_na)/3)] <- '2016'

back2long_na$year[as.integer(2*nrow(back2long_na)/3)]   <-
'2017'
```

You have noticed, that the new column has many values. Fill the NAs with the non-missing value write above it. (eg.the NA's that are below the '2016' and '2017' value assign it to '2016'.

Hint: use the fill function.

Exercise 10

Extract the numeric values from the Temp feature.

Hint: extract_numeric, this is a very important function when the variable we apply the function on is a character with 'noise', for example '$40' and you want to transform it to 40.

Data Visualization with googleVis exercises part 3

Scatter & Bubble chart

This is the third part of our data visualization series and at

this part we will explore the features of two more of the charts that googleVis provides.

Read the examples below to understand the logic of what we are going to do and then test yous skills with the exercise set we prepared for you. Lets begin!

Answers to the exercises are available here.

Package Installation

As you already know, the first thing you have to do is install and load the googleVis package with:

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

NOTE: The charts are created locally by your browser. In case they are not displayed at once press F5 to reload the page.

Scatter chart

It is quite simple to create a scatter chart with googleVis. We will use the cars dataset. Look at the example below:

```
ScatterC <- gvisScatterChart(cars)
plot(ScatterC)
```

Exercise 1

Create a list named “ScatterC” and pass to it the cars dataset as a scatter chart. **HINT**: Use gvisScatterChart().

Exercise 2

Plot the the scatter chart. **HINT**: Use plot().

Titles

It is time to learn how to enhance the appearance of our googleVis charts. We shall give a title to the chart and also name hAxis and vAxis. Look at the example:

```
options=list(title="Cars", vAxis="{title:'speed'}",
```

hAxis="{title:'dist'}")

Exercise 3

Name your chart “Cars”, your chart’s vAxis “speed”, your chart’s hAxis “dist” and plot the chart. **HINT**: Use list().

Size

You can adjust the size with width and height.

Exercise 4

Set your chart’s width to 600 and height to 300.

Legend

You can deactivate your chart’s legend if you set it to “none”.

Exercise 5

Deactivate your chart’s legend.

Point size & Line width

You can determine the size of the chart’s points with pointsize and also choose to unite them with line with linewidth. For example:
pointSize=4,linewidth=3

Exercise 6

Set point size to 3 and line width to 2.

Bubble Chart

Another amazing type of chart that googleVis provides is the bubble chart. You can create a simple Bubble Chart of the Fruits dataset like this:

```
BubbleC <- gvisBubbleChart(Fruits)
plot(BubbleC)
```

Exercise 7

Create a list named “BubbleC” and pass to it the Fruits dataset as a bubble chart. **HINT**: Use gvisBubbleChart().

Exercise 8

Plot the chart. **HINT**: Use plot().

Bubble Chart’s Features

As you can see, you created a bubble chart but it seems to be useless. In order to make it useful you should pass to it some of your dataset’s variables as features. It depends on what you want to be displayed and how. If you type head(Fruits) you can easily recognize the numeric variables of your dataset. Then you can use them like this:

```
BubbleC <- gvisBubbleChart(Fruits,idvar="VAR1",
xvar="VAR2", yvar="VAR3",
colorvar="VAR4", sizevar="VAR5")
```

Exercise 9

Find the numeric variables of Fruits, then set “Fruit” as idvar, “Sales” as xvar, “Expenses” as yvar, “Year” as colorvar and “Profit” as sizevar and plot your chart. **HINT**: Use head().

Data range

You can also adjust the minimum and maximum number of hAxis and vAxis that you want to be displayed. Look at the example below:
options=list(
hAxis='{minValue:50, maxValue:150}')

Exercise 10

Set your hAxis range from 70 to 130 and your vAxis range from 50 to 100.

Manipulate Biological Data Using Biostrings Package Exercises(Part 4)

Bioinformatics is an amalgamation Biology and Computer science. Biological Data is manipulated using Computers and Computer software's in Bioinformatics. Biological Data includes DNA; RNA & Proteins. DNA & RNA is made of Nucleotide which are our genetic material in which we are coded.Our Structure and Functions are done by protein, which are build of Amino acids
Here in this we try to manipulate DNA, RNA, Protein strings using Biostring Package
Install Packages
Biostrings

Answers to the exercises are available here.

If you obtained a different (correct) answer than those listed

on the solutions page, please feel free to post your answer as a comment on that page.

Exercise 1

Create an RNA string and find palindromes in the sequence

Exercise 2

Create a DNA string and find palindromes in the sequence

Exercise 3

Create a DNA string and find the dinucleotide frequency of the sequences

Exercise 4

Create an RNA string and find the dinucleotide frequency of the sequences

Exercise 5

Create a DNA string and find the oligonucleotide frequency in the sequences

Exercise 6

Create an RNA string and find the oligonucleotide frequency in the sequences

Exercise 7

Create a DNA string and find the trinucleotide frequency in the sequences

Exercise 8

Create an RNA string and find the trinucleotide frequency in the sequences

Exercise 9

Print amino acid alphabets

Exercise 10

Create an Amino acid string and print the frequency of the amino acid strings in the sequence

---

Forecasting for small business Exercises (Part-2)

Uncertainty is the biggest enemy of a profitable business. That is especially true of small business who don't have enough resources to survive an unexpected diminution of revenue or to capitalize on a sudden increase of the demand. In this context, it is especially important to be able to predict accurately the change in the markets to be able to make better decision and stay competitive.

This series of posts will teach you how to use data to make sound prediction. In the first set of exercises, we've seen how to make predictions in the case where the mean and the variance of the time series is constant. Now, we'll see how to

deal with time series whose mean change over time.

The basic type of time series who is not stationary is called a random walk. This random process is obtained by adding each random fluctuation of a stationary time series to the sum of all the previous one. As a consequence, the mean of such a time series will change over time, so it's not stationary, but his variance will stay the same. You can find more information about random walk here. Once you get the hang of how to use random walks, make sure to read part three of this series of exercise, where you will see how to make predictions with this kind of models.

To be able to do theses exercice, you have to have installed the packages forecast and tseries.

Answers to the exercises are available here.

Exercise 1
Set the seed to 42 and generate 100 random points from a normal distribution of mean 0 and standard deviation of 1 to simulate a white noise. Then use the cumsum() function to sum the point of the white noise and simulate a random walk. Finally, use the as.ts() function with the correct argument to create a time series named random.walk1.

Exercise 2
Do a basic plot of random.walk1.

Exercise 3
Follow the steps of exercise 1, but this time generates 100 points from a normal distribution with a mean of 0.5 and a standard deviation of 1. Save the time series in a variable called random.walk2, then plot it.

If we look at the previous plot, we see that there is a general positive trend in the data. This resulted from the fact that the white noise component of the random walk has a mean greater than 0 and a relatively small standard

derivation. In consequence, those values are generally positive and since those components are adding together, each observation of the random walk has a good chance to be greater than their predecessor.

Exercise 4
To get a sense of how the mean and the standard deviation of the white noise affect the shape of the resulting time series, generate 100 points of a random walk with a white noise component of:

- mean = -0.5 and st=1
- mean = 2 and st=1
- mean = 10 and st=1
- mean = 0.5 and st=10

Again, set the seed to 42 and plot your result.

From the first plot, we can see that the sign of the mean of the white noise component determines the sign of the trend line of the curve. If the mean is positive the trend tend to be positive and vice versa. From the other plot, we see how the relative magnitude of the mean and the standard deviation affect the shape of the curve. Since the standard deviation of the white noise determines the degree of randomness of the observations, the greater is this value compared to the mean, the less we can see a trend in the data. The opposite is also true: greater is the mean compared to the standard deviation, the more the curve of the time series tend to be close to the trend line.

In the last set of exercises, we have seen how to do prediction on a time series who's stationery. Since a random walk doesn't have a constant mean, we cannot directly follow the same steps.

Exercise 5
Create a new time series by subtracting of each observation x_i in the random.walk2 time series from exercise 3 the

previous observation x_i-1. Name this time series white.noise and plot it. As usual, set the seed to 42.

Since the random walk is the cumulative sum of each individual white noise component the result of this subtraction is those random components. Since a white noise is stationary, we should be able to use the same step we have seen in the last set of exercises to make predictions on this time series.

Exercise 6
Confirm that the time series white.noise is indeed stationary by drawing his autocorrelation plot.

Exercise 7
Use the Box-Ljung test, the keeps test and the adf test to see if the time series is stationary. You can use the part-1 of this series as reference.

Exercise 8
Use the function HoltWinters and forecast.HoltWinters to apply an exponential smoothing on the white.noise time series and make predictions for the next 5 points. Save the result in a variable called prediction, then print them and plot them.

Exercise 9
Knowing that the prediction are stored in the $mean column of the prediction variable and that the lower and upper limit of the 80\% confidence intervals are stored in the $lower[,1] and $upper[,1] column respectively, plot random.walk2 and the predictions.

So far, we made some assumptions on the random walk to be able to make predictions.

- Constance of the mean of the white noise component
- Constance of the variance of the white noise component
- The white noise component are add together to create the random walk
- The white noise is the only random component of the

random walk

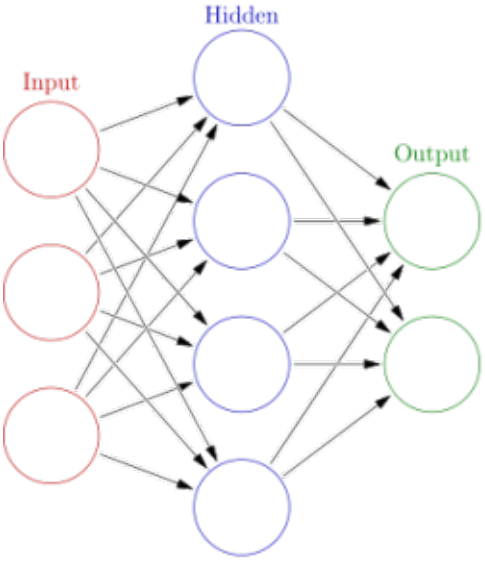
If we use the method shown today to make predictions on a time series that doesn’t verify at least one of those conditions, the results shouldn’t be trusted.

Exercise 10

Load the Load the EuStockMarkets time series from the dataset library and plot the first time series of this set. Then use the diff() function to create a white.noise2 time series containing the white noise component and plot it.

Looking at the time series plot we see a curve which look a lot like a random walk we’ve seen before, but by plotting the white noise component, we see that the amplitude of his random variations increase over time. In this situation, using the method we’ve seen today wouldn’t produce accurate predictions. In the next post in this series of exercises, we’ll see how to transform a random walk to make it respect all the necessary assumptions.

Neural networks Exercises (Part-2)



Source: Wikipedia

Neural network have become a corner stone of machine learning in the last decade. Created in the late 1940s with the intention to create computer programs who mimics the way neurons process information, those kinds of algorithm have long been believe to be only an academic curiosity, deprived of practical use since they require a lot of processing power and other machine learning algorithm outperform them. However since the mid 2000s, the creation of new neural network types and techniques, couple with the increase availability of fast computers made the neural network a powerful tool that every data analysts or programmer must know.

In this series of articles, we'll see how to fit a neural network with R, we'll learn the core concepts we need to know to well apply those algorithms and how to evaluate if our model is appropriate to use in production. Today, we'll practice how to use the nnet and neuralnet packages to create a feedforward neural networks, which we introduce in the last set of exercises. In this type of neural network, all the neuron from the input layer are linked to the neuron from the hidden layer and all of those neuron are linked to the output layer, like seen on this image. Since there's no cycle in this network, the information flow in one direction from the input layer to the hidden layers to the output layer. For more

information about those types of neural network you can read this page.

Answers to the exercises are available here.

Exercise 1
We'll start by practicing what we've seen in the last set of exercises. Load the MASS package and the biopsy dataset, then prepare your data to be feed to a neural network.

Exercise 2
We'll use the nnet() function from the package of the same name to do a logistic regression on the biopsy data set using a feedforward neural network. If you remember the last set of exercises you know that we have to choose the number of neuron in the hidden layer of our feedforward neural network. There's no rule or equation which can tell us the optimal number of neurons to use, so the best way to find the better model is to do a bunch of cross-validation of our model with different number of neurons in the hidden layer and choose the one who would fit best the data. A good range to test with this process is between one neuron and the number of input variables.

Write a function that take a train data set, a test data set and a range of integer corresponding to the number of neurons to be used as parameter. Then this function should, for each possible number of neuron in the hidden layer, train a neural network made with nnet(), make prediction on the test set and return the accuracy of the prediction.

Exercise 3
Use your function on your data set and plot the result. Which should be the number of neurons to use in the hidden layer of your feedforward neural network.

Exercise 4
The nnet() function is easy to use, but doesn't give us a lot of option to customize our neural network. As a consequence,

it’s a good package to use if you have to do a quick model to test a hypothesis, but for more complex model the neuralnet package is a lot more powerful. Documentation for this package can be found here.

Use the neuralnet() function with the default parameter and the number of neuron in the hidden layer set to the answer of the last exercise. Note that this function can only handle numeric value and cannot deal with factors. Then use the compute() function to make prediction on the values of the test set and compute the accuracy of your model.

Exercise 5

The nnet() function use by default the BFGS algorithm to adjust the value of the weights until the output values of our model are close to the values of our data set. The neuralnet package give us the option to use more efficient algorithm to compute those value which result in faster processing time and overall better estimation. For example, by default this function use the resilient backpropagation with weight backtracking.

Use the neuralnet() function with the parameter algorithm set to ‘rprop-‘, which stand for resilient backpropagation without weight backtracking.
Then test your model and print the accuracy.

Exercise 6

Two other algorithm can be used with the neuralnet() function: 'sag' and 'slr'. Those two strings tell the function to use the globally convergent algorithm (grprop) and to modify the learning rate associated with the smallest absolute gradient (sag) or the smallest learning rate (slr). When using those algorithm, it can be useful to pass a vector or list containing the lowest and highest limit for the learning rate to the learningrate.limit parameter.

Again, use the neuralnet() function twice, once with parameter algorithm set to 'sag' and then to 'slr'. In both cases set the learningrate.limit parameter to c(0.1,1) and change the stepmax parameter to 1e+06.

Exercise 7

The learning rate determine how much the backpropagation can affect the weight at each iteration. A high learning rate mean that during the training of the neural network, each iteration can strongly change the value of the weight or, to put in other way, the algorithm learn a lot of each observation in your data set. This mean that outlier could easily affect your weight and make your algorithm diverge from the path of the ideal weights for your problem. A small learning rate mean that the algorithm learn less from each observation in your data set, so your neural network is less affected by outlier, but this mean that you will need more observations to make a good model.

Use the neuralnet() function with parameter algorithm set to ‘rprop+’ twice: once with the learningrate parameter set to 0.01 and another time with the learningrate parameter set to 1. Notice the difference in running time in both cases.

Exercise 8

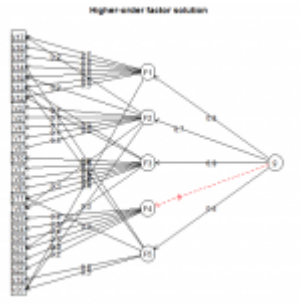
The neuralnet package give us the ability of make a visual representation of the neural network you made. Use the plot() function to visualize one of the neural networks of the last exercise.

Exercise 9
Until now, we've used feedfordward neural network with one hidden layer of neurons, but we could use more. In fact, the state of the art neural network use often 100 of hidden layer for modeling complex behavior. For basic regression problems or even basic digits recognition problems, one layer is enough, but if you want to use more, you can do so with the neuralnet() function by passing a vector of integer to the hidden parameter representing the number of neurons in each layer.

Create a feedforward neural network with three hidden layers of nine neurons and use it on your data.

Exercise 10
Plot the feedforward neural network from the last exercise.

Exploratory Factor Analysis – Exercises



This set of exercises is about exploratory factor analysis. We shall use some basic features of psych package. For quick introduction to exploratory factor analysis and psych package, we recommend this short "how to" guide.

You can download the dataset here. The data is fictitious.

Answers to the exercises are available here.

If you have different solution, feel free to post it.

Exercise 1

Load the data, install the packages psych and GPArotation which we will use in the following exercises, and load it. Describe the data.

Exercise 2

Using the parallel analysis, determine the number of factors.

Exercise 3

Determine the number of factors using Very Simple Structure method.

Exercise 4

Based on normality test, is the Maximum Likelihood factoring method proper, or is OLS/Minres better? *(Tip: Maximum Likelihood method requires normal distribution.)*

Exercise 5

Using oblimin rotation, 5 factors and factoring method from the previous exercise, find the factor solution. Print loadings table with cut off at 0.3.

Exercise 6

Plot factors loadings.

Exercise 7

Plot structure diagram.

Exercise 8

Find the higher-order factor model with five factors plus

general factor.

Exercise 9

Find the bifactor solution.

Exercise 10

Reduce the number of dimensions using hierarchical clustering analysis.

Data Visualization with googleVis exercises part 2

Area, Stepped & Combo charts

In the second part of our series we are going to meet three more googleVis charts. More specifically these charts are **Area Chart**, **Stepped Area Chart** and **Combo** Chart.

Read the examples below to understand the logic of what we are going to do and then test yous skills with the exercise set we prepared for you. Lets begin!

Answers to the exercises are available here.

Package & Data frame

As you already know, the first thing you have to do is install

and load the googleVis package with:

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

Secondly we will create an experimental data frame which will be used for our charts' plotting. You can create it with:

```
df=data.frame(name=c("James", "Curry", "Harden"),
Pts=c(20,23,34),
Rbs=c(13,7,9))
```

NOTE: The charts are created locally by your browser. In case they are not displayed at once press F5 to reload the page.

Area chart

It is quite simple to create an area chart with googleVis with:

```
AreaC <- gvisBarChart(df)
plot(AreaC)
```

Exercise 1

Create a list named "AreaC" and pass to it the "df" data frame you just created as an area chart. **HINT**: Use gvisAreaChart().

Exercise 2

Plot the area chart. **HINT**: Use plot().

Stepped Area chart

Creating a stepped area chart is a little different than the area chart. You have to set the X and Y variables and also make it stacked in order to display the values correctly. Here is an example:

```
SteppedAreaC <- gvisSteppedAreaChart(df, xvar="name",
yvar=c("val1", "val2"),
options=list(isStacked=TRUE))
plot(SteppedAreaC)
```

Exercise 3

Create a list named “SteppedAreaC” and pass to it the “df” data frame you just created as a stepped area chart. You should also set the X variable as the players’ names and Y variable as their values. **HINT**: Use gvisSteppedAreaChart(), xvar and yvar.

Exercise 4

Plot the stepped area chart. **HINT**: Use plot().

Learn more about using GoogleVis in the online course Mastering in Visualization with R programming. In this course you will learn how to:

- Work extensively with the GoogleVis package and its functionality
- Learn what visualizations exist for your specific use case
- And much more

Exercise 5

Now transform your stepped area chart to stacked to correct it and plot it.

Combo chart

The next chart we are going to meet is the combination of lines and bars chart known as combo chart. You can produce it like this:

```
ComboC <- gvisComboChart(df, xvar="country",
yvar=c("val1", "val2"),
options=list(seriesType="bars",
series='{1: {type:"line"}}'))
plot(ComboC)
```

Exercise 6

Create a list named “ComboC” and pass to it the “df” data frame you just created as a combo chart. You should also set the X variable as the players’ names and Y variable as their values. **HINT**: Use gvisComboChart(), xvar and yvar.

Exercise 7

Plot the chart. What kind of chart do you see? **HINT**: Use plot().

In order to add the bars we have to set it as the example below.

```
options=list(seriesType="bars",
series='{1: {type:"line"}}')
```

Exercise 8

Transform the chart you just created into a combo chart with bars and lines and plot it. **HINT**: Use list().

Exercise 9

In the previous exercise “Pts” are represented by the bars and “Rbs” by the lines. Try to reverse them.

You can easily transform your combo chart into a column chart or a line chart just be setting series='{1: {type:"line"}}' to 2

Exercise 10

Transform the combo chart into a column chart and then into a line chart.