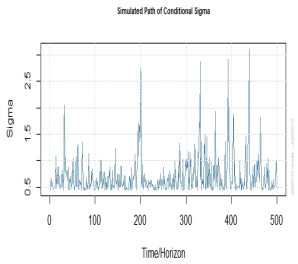
# Volatility modelling in R exercises (Part-1)




Volatility modelling is typically used for high frequency financial data. Asset returns are typically uncorrelated while the variation of asset prices (volatility) tends to be correlated across time.
In this exercise set we will use the rugarch package (package description: here) to implement the ARCH (Autoregressive Conditional Heteroskedasticity) model in R.

Answers to the exercises are available here.

**Exercise 1**
Load the rugarch package and the dmbp dataset (Bollerslev, T. and Ghysels, E. 1996, Periodic Autoregressive Conditional Heteroscedasticity, Journal
of Business and Economic Statistics, 14, 139–151). This dataset has daily logarithmic nominal returns for Deutsche-mark / Pound. There is also a dummy variable to indicate non-trading days.

**Exercise 2**
Define the daily return as a time series variable and plot the return against time. Notice the unpredictability apparent from the graph.

**Exercise 3**
Plot the graph of the autocorrelation function of returns. Notice that there is hardly any evidence of autocorrelation of

returns.

**Exercise 4**
Plot the graph of the autocorrelation function of squared returns. Notice the apparent strong serial correlation.

**Learn more** about Model Evaluation in the online course Regression Machine Learning with R. In this course you will learn how to:

- Avoid model over-fitting using cross-validation for optimal parameter selection
- Explore maximum margin methods such as best penalty of error term support vector machines with linear and non-linear kernels.
- And much more

**Exercise 5**
We will first simulate and analyze an ARCH process. Use the ugarchspec function to define an ARCH(1) process. The return has a simple mean specification with mean=0. The variance follows as AR-1 process with constant=0.2 and AR-1 coefficient = 0.7.

**Exercise 6**
Simulate the ARCH process for 500 time periods. Exercises 7 to 9 use this simulated data.

**Exercise 7**
Plot the returns vs time and note the apparent unpredictability. Plot the path of conditional sigma vs time and note that there is some persistence over time.

**Exercise 8**
Plot the ACF of returns and squared returns. Note that there is no auto-correlation between returns but squared returns have significant first degree autocorrelation as we defined in

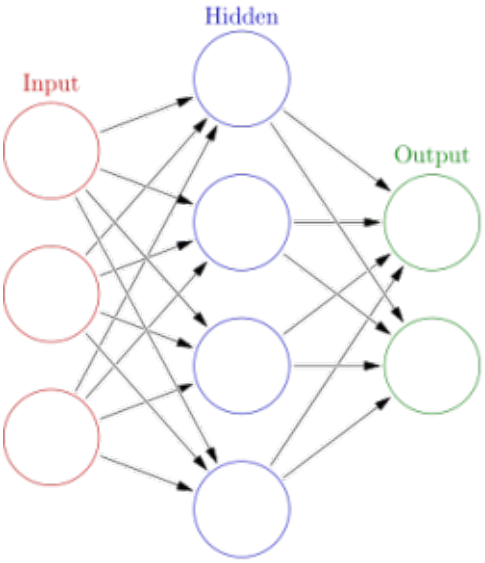
exercise-5.

**Exercise 9**
Test for ARCH effects using the Ljung Box test for the simulated data.

**Exercise 10**
Test for ARCH effects using the Ljung Box test for the currency returns data.

---

# Neural networks Exercises (Part-3)




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. In the last exercises sets, we have seen how to implement a feed-forward neural network in R. That kind of neural network is quite useful to match a single input value to a specific output value, either a dependent variable in regression problems or a class in clustering problems. However sometime, a sequence of input can give a lot more of information to the network than a single value. For example, if you want to train a neural network to predict which letter will come next in a word based on which letters have been typed, making prediction based on the last letter entered can give good results, but if all the previous letter are used for making the predictions the results should be better since the arrangement of previous letter can give important information about the rest of the word.

In today's exercise set, we will see a type of neural network that is design to make use of the information made available by using sequence of inputs. Those "recurrent neural networks" do so by using a hidden state at time t-1 that influence the calculation of the weight at time t. For more information about this type of neural network, you can read this article which is a good introduction on the subject.

Answers to the exercises are available here.

**Exercise 1**
We will start by using a recurrent neural network to predict the values of a time series. Load the tsEuStockMarkets dataset

from the dataset package and save the first 1400 observations from the “DAX” time series as your working dataset.

**Exercise 2**
Process the dataset so he can be used in a neural network.

**Exercise 3**
Create two matrix containing 10 sequences of 140 observations from the previous dataset. The first one must be made of the original observations and will be the input of our neural network. The second one will be the output and since we want to predict the value of the stock market at time t+1 based on the value at time t, this matrix will be the same as the first one were all the elements are shifted from one position. Make sure that each sequence are coded as a row of each matrix.

**Exercise 4**
Set the seed to 42 and choose randomly eight sequences to train your model and two sequences that will be used for validation later. Once it’s done, load the rnn package and use the trainr() function to train a recurrent neural network on the training dataset. For now, use a learning rate of 0.01, one hidden layer of one neuron and 500 epoch.

**Exercise 5**
Use the function predictr to make prediction on all the 10 sequences of your original data matrix, then plot the real values and the predicted value on the same graph. Also draw the plot of the prediction on the test set and the real value of your dataset.

**Exercise 6**
The last model seems to underestimate the stock values that are higher than 0.5. Repeat the step of exercise 3 and 4 but this time use 10 hidden layers. Once it’s done calculate the RMSE of your predictions. This will be the baseline model for the rest of this exercise set.

**Learn more** about neural networks in the online course Machine Learning A-Z™: Hands-On Python & R In Data Science. In this course you will learn how to:

- Work with Deep Learning networks and related packages in R
- Create Natural Language Processing models
- And much more

**Exercise 7**

One interesting method often used to accelerate the training of a neural network is the "Nesterov momentum". This procedure is based on the fact that while trying to find the weights that minimize the cost function of your neural network, optimization algorithm like gradient descend "zigzag" around a straight path to the minimum value. By adding a momentum matrix, which keeps track of the general direction of the gradient, to the gradient we can minimize the deviation from this optimal path and speeding the convergence of the algorithm. You can see this video for more information about this concept.

Repeat the last exercise, but this time use 250 epochs and a momentum of 0.7.

**Exercise 8**

As special type of recurrent neural network trained by backpropagation through time is called the Long Short-Term Memory (LSTM) network. This type of recurrent neural network is quite useful in a deep learning context, since this method is robust again the vanishing gradient problem. We will see both of those concepts more in detail in a future exercise set, but for now you can read about it here.

The trainr() function give us the ability to train a LSTM network by setting the network_type parameter to "lstm". Use this algorithm with 500 epochs and 20 neuron in the hidden

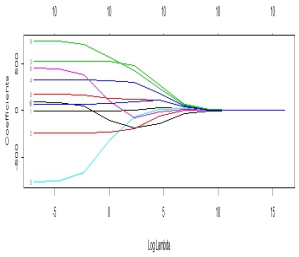
layer to predict the value of your time series.

**Exercise 9**
When working with a recurrent neural network it is important to choose an input sequence length that give the algorithm the maximum information possible without adding useless noise to the input. Until now we used 10 sequences of 140 observations. Train a recurrent neural network on 28 sequences of 50 observations, make prediction and compute the RMSE to see if this encoding had an effect on your predictions.

**Exercise 10**
Try to use all of the 1860 observation in the "DAX" time series to train and test a recurrent neural network. Then post the setting you used for your model and why you choose them in the comments.

---

# Ridge regression in R exercises



Bias vs Variance tradeoff is always encountered in applying supervised learning algorithms. Least squares regression provides a good fit for the training set but can suffer from high variance which lowers predictive ability. To counter this problem, we can regularize the beta coefficients by employing a penalization term. Ridge regression applies l2 penalty to

the residual sum of squares. In contrast, LASSO regression, which was covered here previously, applies l1 penalty.
Using ridge regression, we can shrink the beta coefficients towards zero which would reduce variance at the cost of higher bias which can result in better predictive ability than least squares regression. In this exercise set we will use the glmnet package (package description: here) to implement ridge regression in R.

Answers to the exercises are available here.

**Exercise 1**
Load the lars package and the diabetes dataset (Efron, Hastie, Johnstone and Tibshirani (2003) "Least Angle Regression" (with discussion) Annals of Statistics). This is the same dataset from the LASSO exercise set and has patient level data on the progression of diabetes. Next, load the glmnet package that will that we will now use to implement ridge regression.
The dataset has three matrices x, x2 and y. x has a smaller set of independent variables while x2 contains the full set with quadratic and interaction terms. y is the dependent variable which is a quantitative measure of the progression of diabetes.
Generate separate scatterplots with the line of best fit for all the predictors in x with y on the vertical axis.
Regress y on the predictors in x using OLS. We will use this result as benchmark for comparison.

**Exercise 2**
Fit the ridge regression model using the glmnet function and plot the trace of the estimated coefficients against lambdas. Note that coefficients are shrunk closer to zero for higher values of lambda.

**Exercise 3**
Use the cv.glmnet function to get the cross validation curve and the value of lambda that minimizes the mean cross validation error.

**Exercise 4**
Using the minimum value of lambda from the previous exercise, get the estimated beta matrix. Note that coefficients are lower than least squares estimates.

**Exercise 5**
To get a more parsimonious model we can use a higher value of lambda that is within one standard error of the minimum. Use this value of lambda to get the beta coefficients. Note the shrinkage effect on the estimates.

**Learn more** about Model Evaluation in the online course Regression Machine Learning with R. In this course you will learn how to:

- Avoid model over-fitting using cross-validation for optimal parameter selection
- Explore maximum margin methods such as best penalty of error term support vector machines with linear and non-linear kernels.
- And much more

**Exercise 6**
Split the data randomly between a training set (80%) and test set (20%). We will use these to get the prediction standard error for least squares and ridge regression models.

**Exercise 7**
Fit the ridge regression model on the training and get the estimated beta coefficients for both the minimum lambda and the higher lambda within 1-standard error of the minimum.

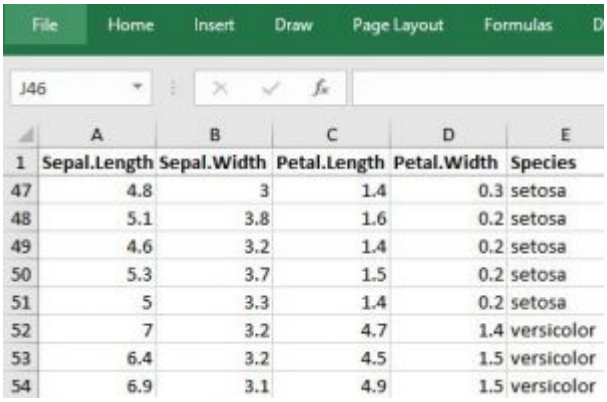
**Exercise 8**
Get predictions from the ridge regression model for the test set and calculate the prediction standard error. Do this for both the minimum lambda and the higher lambda within 1-standard error of the minimum.

**Exercise 9**
Fit the least squares model on the training set.

**Exercise 10**
Get predictions from the least squares model for the test set and calculate the prediction standard error.

---

# Using the xlsx package to create an Excel file




Microsoft Excel is perhaps the most popular data anlysis tool out there. While arguably convenient, spreadsheet software is error prone and Excel code can be very hard to review and test.

After successfully completing this exercise set, you will be able to prepare a basic Excel document using just R (no need to touch Excel yourself), leaving behind a reproducible R-script.

Solutions are available here.

**Exercise 1**
Install and load the xlsx package, using the dependencies = TRUE option.

**Exercise 2**
Create an xlsx workbook object in your R workspace and call it wb.

**Exercise 3**
Create a sheet object in wb named iris assign it the name sheet1 in your workspace.

**Exercise 4**
Write the built-in Iris data.frame to the iris sheet without row names. Hint: use the addDataFrame() function.

Now you can write your workbook anytime to your working directory using saveWorkbook(wb, "filename.xlsx").

**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**
Apply ‘freeze pane’ on the top row.

**Exercise 6**
Set width of columns 1 through 5 to 12, that is 84 pixels.

**Exercise 7**
Use Font, CellBlock and CB.setFont to make the header in bold.

**Exercise 8**
Using tapply generate a table with the mean of ‘petal width’ by species and write to a new sheet called pw, from row 2 down.

**Exercise 9**
Add a title in cell A1 above the table, merge the cells of the first three columns.

**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 Manipulation with Data Table -Part 1

In the exercises below we cover the some useful features of data.table ,data.table is a library in R for fast manipulation of large data frame .Please see the data.table vignette before trying the solution .This first set is intended for the begineers of data.table package and does not cover set keywords, joins of data.table which will be covered in the next set . Load the data.table library in your r session before starting the exercise
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**
Load the iris dataset ,make it a data.table and name it iris_dt ,Print mean of Petal.Length, grouping by first letter of Species from iris_dt .

**Exercise 2**
Load the diamonds dataset from ggplot2 package as dt (a data.table) ,Find mean price for each group of cut and color .

**Exercise 3**
Load the diamonds dataset from ggplot2 package as dt . Now group the dataset by price per carat and print top 5 in terms of count per group . Dont use head ,use chaining in data.table to achieve this

**Exercise 4**
Use the already loaded diamonds dataset and print the last two carat value of each cut .

**Exercise 5**
In the same data set , find median of the columns x,y,z per cut . Use data.table's methods to achieve this .

**Exercise 6**
Load the airquality dataset as data.table, Now I want to find Logarithm of wind rate for each month and for days greater than 15

**Exercise 7**
In the same data set , for all the odd rows ,update Temp column by adding 10 .

**Exercise 8**
data.table comes with a powerful feature of updating column by reference as you have seen in the last exercise,Its even possible to update /create multiple columns .Now to test that in the airquality data.table that you have created previously,add 10 to Solar.R ,Wind .

**Exercise 9**
Now you have a fairly good idea of how easy its to create

multiple column ,Its even possible to use delete multiple column using the same idea. In this exercise , use the same airquality data.table that you have created previously from airquality and delete Solar.R,Wind,Temp using a single expression

**Exercise 10**

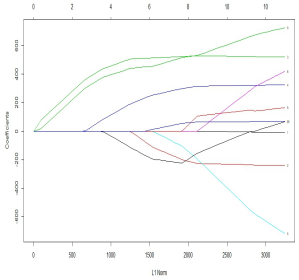
Load the airquality dataset as data.table again , I want to create two columns a,b which indicates temp in Celcius and Kelvin scale . Write a expression to achieve same.

Celcius = (Temp-32)*5/9

Kelvin = Celcius+273.15

---

# LASSO regression in R exercises



Least Absolute Shrinkage and Selection Operator (LASSO) performs regularization and variable selection on a given model. Depending on the size of the penalty term, LASSO shrinks less relevant predictors to (possibly) zero. Thus, it enables us to consider a more parsimonious model. In this exercise set we will use the glmnet package (package description: here) to implement LASSO regression in R.

Answers to the exercises are available here.

**Exercise 1**

Load the lars package and the diabetes dataset (Efron, Hastie, Johnstone and Tibshirani (2003) “Least Angle Regression” Annals of Statistics). This has patient level data on the progression of diabetes. Next, load the glmnet package that will be used to implement LASSO.

**Exercise 2**
The dataset has three matrices x, x2 and y. While x has a smaller set of independent variables, x2 contains the full set with quadratic and interaction terms. y is the dependent variable which is a quantitative measure of the progression of diabetes.
It is a good idea to visually inspect the relationship of each of the predictors with the dependent variable. Generate separate scatterplots with the line of best fit for all the predictors in x with y on the vertical axis. Use a loop to automate the process.

**Exercise 3**
Regress y on the predictors in x using OLS. We will use this result as benchmark for comparison.

**Exercise 4**
Use the glmnet function to plot the path of each of x’s variable coefficients against the L1 norm of the beta vector. This graph indicates at which stage each coefficient shrinks to zero.

- And much more

**Exercise 5**
Use the cv.glmnet function to get the cross validation curve and the value of lambda that minimizes the mean cross validation error.

**Exercise 6**
Using the minimum value of lambda from the previous exercise, get the estimated beta matrix. Note that some coefficients have been shrunk to zero. This indicates which predictors are important in explaining the variation in y.

**Exercise 7**
To get a more parsimonious model we can use a higher value of lambda that is within one standard error of the minimum. Use this value of lambda to get the beta coefficients. Note that more coefficients are now shrunk to zero.

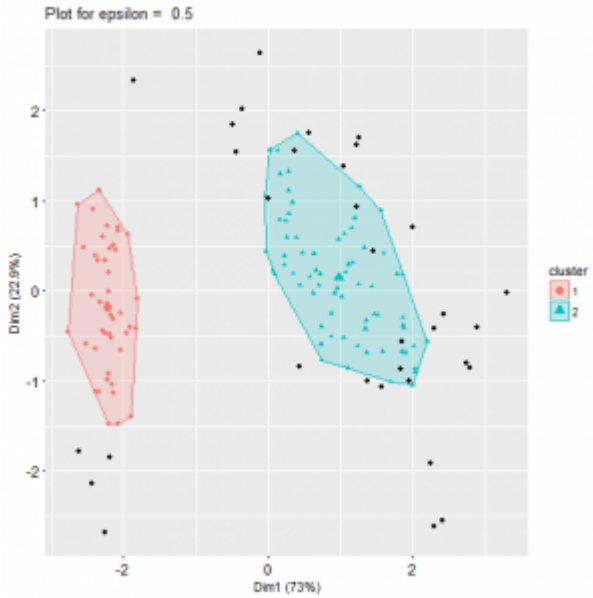
**Exercise 8**
As mentioned earlier, x2 contains a wider variety of predictors. Using OLS, regress y on x2 and evaluate results.

**Exercise 9**
Repeat exercise-4 for the new model.

**Exercise 10**
Repeat exercises 5 and 6 for the new model and see which coefficients are shrunk to zero. This is an effective way to narrow down on important predictors when there are many candidates.

---

# Density-Based Clustering Exercises




Density-based clustering is a technique that allows to partition data into groups with similar characteristics (clusters) but does not require specifying the number of those groups in advance. In density-based clustering, clusters are defined as dense regions of data points separated by low-density regions. Density is measured by the number of data points within some radius.
Advantages of density-based clustering:

- as mentioned above, it does not require a predefined number of clusters,
- clusters can be of any shape, including non-spherical ones,
- the technique is able to identify noise data (outliers).

Disadvantages:

- density-based clustering fails if there are no density drops between clusters,
- it is also sensitive to parameters that define density (radius and the minimum number of points); proper parameter setting may require domain knowledge.

There are different methods of density-based clustering. The most popular are DBSCAN (density-based spatial clustering of applications with noise), which assumes constant density of clusters, OPTICS (ordering points to identify the clustering structure), which allows for varying density, and “mean-

shift".
This set of exercises covers basic techniques for using the DBSCAN method, and allows to compare its result to the results of the k-means clustering algorithm by means of the silhouette analysis.
The set requires the packages dbscan, cluster, and factoextra to be installed. The exercises make use of the iris data set, which is supplied with R, and the wholesale customers data set from the University of California, Irvine (UCI) machine learning repository (download here).
Answers to the exercises are available here.

**Exercise 1**
Create a new data frame using all but the last variable from the iris data set, which is supplied with R.

**Exercise 2**
Use the scale function to normalize values of all variables in the new data set (with default settings). Ensure that the resulting object is of class data.frame.

**Exercise 3**
Plot the distribution of distances between data points and their fifth nearest neighbors using the kNNdistplot function from the dbscan package.
Examine the plot and find a tentative threshold at which distances start increasing quickly. On the same plot, draw a horizontal line at the level of the threshold.

**Exercise 4**
Use the dbscan function from the package of the same name to find density-based clusters in the data. Set the size of the epsilon neighborhood at the level of the found threshold, and set the number of minimum points in the epsilon region equal to 5.
Assign the value returned by the function to an object, and print that object.

**Exercise 5**

Plot the clusters with the fviz_cluster function from the factoextra package. Choose the geometry type to draw only points on the graph, and assign the ellipse parameter value such that an outline around points of each cluster is not drawn.

(Note that the fviz_cluster function produces a 2-dimensional plot. If the data set contains two variables those variables are used for plotting, if the number of variables is bigger the first two principal components are drawn.)



**Exercise 6**

Examine the structure of the cluster object obtained in Exercise 4, and find the vector with cluster assignments. Make a copy of the data set, add the vector of cluster assignments to the data set, and print its first few lines.

**Exercise 7**

Now look at what happens if you change the epsilon value.

1. Plot again the distribution of distances between data points and their fifth nearest neighbors (with the kNNdistplot function, as in Exercise 3). On that plot, draw horizontal lines at levels 1.8, 0.5, and 0.4.
2. Use the dbscan function to find clusters in the data with the epsilon set at these values (as in Exercise 4).
3. Plot the results (as in the Exercise 5, but now set the

ellipse parameter value such that an outline around points is drawn).

**Exercise 8**

This exercise shows how the DBSCAN algorithm can be used as a way to detect outliers:

1. Load the Wholesale customers data set, and delete all variables with the exception of Fresh and Milk. Assign the data set to the customers variable.
2. Discover clusters using the steps from Exercises 2-5: scale the data, choose an epsilon value, find clusters, and plot them. Set the number of minimum points to 5. Use the db_clusters_customers variable to store the output of the dbscan function.

**Exercise 9**

Compare the results obtained in the previous exercise with the results of the k-means algorithm. First, find clusters using this algorithm:

1. Use the same data set, but get rid of outliers for both variables (here the outliers may be defined as values beyond 2.5 standard deviations from the mean; note that the values are already expressed in unit of standard deviation about the mean). Assign the new data set to the customers_core variable.
2. Use kmeans function to obtain an object with cluster assignments. Set the number of centers equal to 4, and the number of initial random sets (the nstart parameter) equal to 10. Assign the obtained object to the variable km_clusters_customers variable.
3. Plot clusters using the fviz_cluster function (as in the previous exercise).

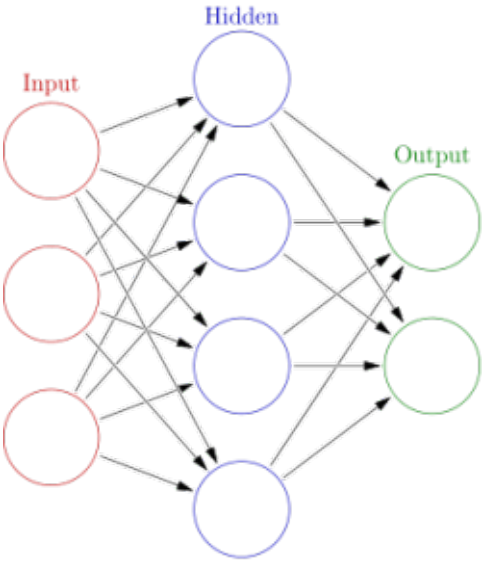
**Exercise 10**

Now compare the results of DBSCAN and k-means using silhouette analysis:

1. Retrieve a vector of cluster assignments from the db_clusters_customers object.
2. Calculate distances between data points in the customers data set using the dist function (with the default parameters).
3. Use the vector and the distances object as inputs into the silhouette function from the cluster package to get a silhouette information object.
4. Plot that object with the fviz_silhouette function from the factoextra package.
5. Repeat the steps described above for the km_clusters_customers object and the customers_core data sets.
6. Compare two plots and the average silhouette width values.

---

# Neural networks Exercises (Part-1)




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. This set of exercises is an introduction to neural networks where we'll use them to create two simple regression and clustering model. Doing so, we'll use a lot of basic concepts we'll explore further in future sets. If you want more informations about neural network, your can see this page.

Answers to the exercises are available here.

**Exercise 1**
We'll start by creating the data set on which we want to do a simple regression. Set the seed to 42, generate 200 random points between -10 and 10 and store them in a vector named X. Then, create a vector named Y containing the value of sin(x). Neural network are a lot more flexible than most regression algorithms and can fit complex function with ease. The biggest challenge is to find the appropriate network function appropriate to the situation.

**Exercise 2**
A network function is made of three components: the network of neurons, the weight of each connection between neuron and the activation function of each neuron. For this example, we'll

use a feed-forward neural network and the logistic activation which are the defaults for the package nnet. We take one number as input of our neural network and we want one number as the output so the size of the input and output layer are both of one. For the hidden layer, we'll start with three neurons. It's good practice to randomize the initial weights, so create a vector of 10 random values, picked in the interval [-1,1].

**Exercise 3**

Neural networks have a strong tendency of overfitting your data, meaning they become really good at describing the relationship between the values in your data set, but are not effective with data that wasn't used to train your model. As a consequence, we need to cross-validate our model. Set the seed to 42, then create a training set containing 75% of the values in your initial data set and a test set containing the rest of your data.

**Exercise 4**

Load the nnet package and use the function of the same name to create your model. Pass your weights via the Wts argument and set the maxit argument to 50. We want to fit a function which can have for output multiple possible values. To do so, set the linout argument to true. Finally, take the time to look at the structure of your model.



**Exercise 5**

Predict the output for the test set and compute the RMSE of your predictions. Plot the function sin(x) and then plot your predictions.

**Exercise 6**
The number of neurons in the hidden layer, as well as the number of hidden layer used, has a great influence on the effectiveness of your model. Repeat the exercises three to five, but this time use a hidden layer with seven neurons and initiate randomly 22 weights.

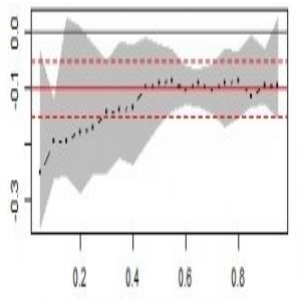
**Exercise 7**
Now let us use neural networks to solve a classification problem, so let’s load the iris data set! It is good practice to normalize your input data to uniformize the behavior of your model over different range of value and have a faster training. Normalize each factor so that they have a mean of zero and a standard deviation of 1, then create your train and test set.

**Exercise 8**
Use the nnet() and use a hidden layer of ten neurons to create your model. We want to fit a function which have a finite amount of value as output. To do so, set the linout argument to true. Look at the structure of your model. With classification problem, the output is usually a factor that is coded as multiple dummy variables, instead of a single numeric value. As a consequence, the output layer have as one less neuron than the number of levels of the output factor.

**Exercise 9**
Make prediction with the values of the test set.

**Exercise 10**
Create the confusion table of your prediction and compute the accuracy of the model.

# Quantile Regression in R exercises




The standard OLS (Ordinary Least Squares) model explains the relationship between independent variables and the conditional mean of the dependent variable. In contrast, quantile regression models this relationship for different quantiles of the dependent variable.
In this exercise set we will use the quantreg package (package description: here) to implement quantile regression in R.

Answers to the exercises are available here.

**Exercise 1**
Load the quantreg package and the barro dataset (Barro and Lee, 1994). This has data on GDP growth rates for various countries.
Next, summarize the data.

**Exercise 2**
The dependent variable is y.net (Annual change per capita GDP). The remaining variables will be used to explain y.net. It is easier to combine variables using cbind before applying regression techniques. Combine variables so that we can write Y ~ X.

**Exercise 3**

Regress y.net on the independent variables using OLS. We will use this result as benchmark for comparison.

**Exercise 4**
Using the rq function, estimate the model at the median y.net. Compare results from exercise-3.

**Learn more** about Model Evaluation in the online course [Regression Machine Learning with R](). In this course you will learn how to:

- Avoid model over-fitting using cross-validation for optimal parameter selection
- Explore maximum margin methods such as best penalty of error term support vector machines with linear and non-linear kernels.
- And much more

**Exercise 5**
Estimate the model for the first and third quartiles and compare results.

**Exercise 6**
Using a single command estimate the model for 10 equally spaced deciles of y.net.

**Exercise 7**
quantreg package also offers shrinkage estimators to determine which variables play the most important role in predicting y.net. Estimate the model with LASSO based quantile regression at the median level with lambda=0.5.

**Exercise 8**
Quantile plots are most useful for interpreting results. To do that we need to define the sequence of percentiles. Use the seq function to define the sequence of percentiles from 5% to 95% with a jump of 5%.

**Exercise 9**
Use the result from exercise-8 to plot the graphs. Note that the red line is the OLS estimate bounded by the dotted lines which represent confidence intervals.

**Exercise 10**
Using results from exercise-5, test whether coefficients are significantly different for the first and third quartile based regressions.

---

# A Primer in functional Programming in R (part -2)


In the last exercise, We have seen how powerful functional programming principles can be and how it can drammatically increase the readablity of the code and how easily you can work with them .In this set of exercises we will look at functional programming principles with purrr.Purrr comes with a number of interesting features and is really useful in writing clean and concise code . Please check the documentation and load the purrr library in your R session before starting these exercise set .
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**
From the airquality dataset( available in base R ) , Find the mean ,median ,standard deviation of all columns using map functions .

**Exercise 2**

In the same dataset,find 95th percentile of each column excluding the NA values

**Exercise 3**
Load the iris dataset ,with help of pipe and map functions find out the mean of the relavant columns.Keep in mind mean is meant for numeric columns ,so you may need multiple map like functions.I expect the output as a dataframe .

**Exercise 4**
I have a vector x <- 1:20 ,I want to multiply every odd element with 2 and every even element with 4 , find a solution using purrr .

**Exercise 5**
I have a sequence x <- 1:100 , I want to increase the 1st,11th, 21st…91st element by 1 . How can I achieve that .
**Exercise 6**
Suppose I have two character vectors
x <- letters # letters is a vector of alphabets in small available in R
y <- LETTERS # LETTERS is a vector of alphabets in caps available in R

How do I join each capital letter with the small letter so that the end result looks like this
"A,a" "B,b" ,.. and so on

**Exercise 7**
The previous exercise gave you the intuition of how to work

parallely on two vectors.Now accepts a list of character vectors of same size and joins them like the previous exercise . so if I have a
list like x <- list(c("a","b","c"),c("d","e",f"),c("g","h","i")) ,it should give me output as .
[1] "a d g" "b e h" "c f i"

**Exercise 8**
using a functional tool from purrr ,reverse the letters so that my output is a string of 26 character starting from z and ending at a
like “z y x w v u t s r q p o n m l k j i h g f e d c b a”

**Exercise 9**
Exercise on Purrr wont be complete if We dont mention its “Filter” like functions . take a numeric vector of 1:100 , keep all the numbers which are divisible by 2 and after that remove the one’s which are divisible by 5

**Exercise 10**
Ability to create partial functions is a powerful tool in functional programming. This allow functions to behave a little like data structures .Consider creating a partial function whenever you see you are repeating functions argument. This may not sound very useful in context of this exercise but I am sure you will find it very useful in your R gigs .Now create a Partial function
ql ,which is to find the 25th percentile and 50th percentile of a column from a dataframe .use it to find the same from airquality dataset .Dont use quantile twice in this exercise .