# DSL for Multiple Linear Regression

# Documentation

## Definition with EBNF:

keyword = “lm” ;

leftBracket = “(” ;

rightBracket = “)” ;

semicolon = “;” ;

equal = “=” ;

uppercaseLetters = "A" | "B" | "C" | "D" | "E" | "F" | "G"

| "H" | "I" | "J" | "K" | "L" | "M" | "N"

| "O" | "P" | "Q" | "R" | "S" | "T" | "U"

| "V" | "W" | "X" | "Y" | "Z”;

lowercaseLetters = "a" | "b" | "c" | "d" | "e" | "f" | "g"

| "h" | "i" | "j" | "k" | "l" | "m" | "n"

| "o" | "p" | "q" | "r" | "s" | "t" | "u"

| "v" | "w" | "x" | "y" | "z" ;

variable = {uppercaseLetters | lowercaseLetters} ;

oneToNine = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ;

numbers = “0” | oneToNine ;

digit = “0” | oneToNine| {numbers} ;

delimiter = “|” ;

operator = "+” | “-”| “\*”| “/”| “[”| “]”| “.”| “^”| “{”| “}”| “%” ;

alpha = variable ;

beta = variable ;

data = "csv” | “sql”| “excel” ;

winPath = [uppercaseLetters “:” “\\”] {lowercaseLetters | uppercaseLetters | “\_”| digit | “\\” } ;

linuxPath = {“/” | lowercaseLetter | uppercaseLetter | “\_”| digit} ;

path = winPath | linuxPath ;

dataTerm = data, equal, path ;

dataTerms = dataTerm | {dataTerm} ;

nestedExpr = {leftBracket | digit | operator | rightBracket}

termPart = nestedExpr | digit | operator ;

termExpr = termPart | {termPart} ;

term = [beta], [termExpr], [beta] ;

variables = term | {term} ;

dsl = keyword, leftBracket, alpha, tild, variables, delimiter, dataTerms, rightBracket ;

inputString = dsl | dsl, inputString ;

## An example for this DSL would be

lm(Sales ~ TV \* 2, 4 / Radio, 3 \* (Newspaper ^ 2) | sql=/home/source1.db, csv=/home/source2.csv, excel=/home/source3.xls);

It is possible to define more than one data source. Only the data sources sql, csv or excel are supported. An error will occur if other data sources are used. The parser will check if the variables, like “TV” or “Radio” are defined in the data source. If the variable is not found, it will be ignored. Furthermore if a variable occurs in two given data sources an error will be displayed too, because the parser will not be sure that the corresponding value to the variable is correct. The “;” after the statement is mandatory. The parser is able to handle multiple statements in one file. Between the operators and variables must be a space. Sub-terms must be covered by brackets and must be separated by spaces. For example:

(1 + 2) not (1+2)

(1 \* 2) not (1\*2)

(- 1) not (-1)

The DSL can look like this:

lm(Sales ~ TV \* 2, (4 + 1) / (- Radio), (- 3) \* (Newspaper ^ 2) | sql=/home/source1.db, csv=/home/source2.csv, excel=/home/source3.xls);

If the result of the variable terms is negative, the absolute value will be used instead. For example:

lm(Sales ~ TV \* 2, (- 4) / Radio, (- 3) \* (Newspaper ^ 2) | sql=/home/source1.db, csv=/home/source2.csv, excel=/home/source3.xls);

==

lm(Sales ~ TV \* 2, 4 / (- Radio), 3 \* (Newspaper ^ 2) | sql=/home/source1.db, csv=/home/source2.csv, excel=/home/source3.xls);

## How to calculate multiple linear regression programmatically

For calculating the multiple linear regression, the programming language python was used. The DSL will be parsed with the library pyparsing. In pyparsing the syntax of the DSL will be defined.

It is a console Program. The command for executing the program looks like this:

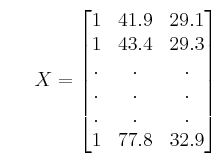


As a parameter the file with the DSL has to be defined.

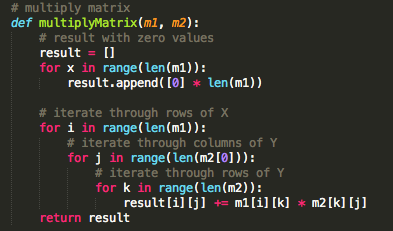
The multiple linear regression calculates the correlation coefficients. They describe, how closely the relationship of variables follows a straight line. For calculation the following formula will be used:



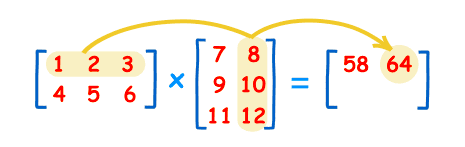
Now we have to insert our values. The “X” value will be a term after the “~” and the Y value is the variable before the “~”. The values of each variable will be presented in a Matrix. The “T” stands for transpose matrix. One important step has to be made for the X Matrix. A prefix with 1 will be applied to the matrix. For example[[1]](#footnote-1):



For multiplying matrices the following code is used:

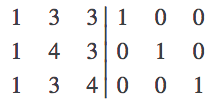


At first the result matrix will be generated and filled with zeroes. If the matrix has not the same dimensions as the first input matrix, the calculation would fail and an out of bounds error will appear. The second part of the code snippet shows the iteration through the matrices. The first value of the column in Matrix m1 will be multiplied with the first value of the row of Matrix m2, and so on. The results of the multiplication will be summed up. This value is the first element in our multiplied Matrix. For a better understanding look on the following picture[[2]](#footnote-2):



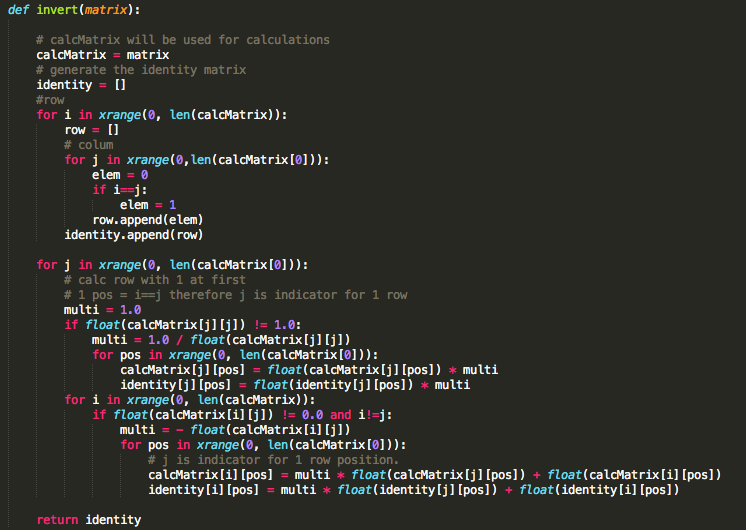
After multiplying the matrices, the result matrix has to be inverted. We need to invert matrices, because they can’t be divided. Multiplying a matrix by its inverse, is the matrix division.

In this project the Gauss-Jordan Elimination algorithm will be used for inverting matrices. For this algorithm you need a matrix and an identity matrix. The identity matrix consists of 0 with 1 diagonal arranged. As an example look at the following picture[[3]](#footnote-3):



After the calculation, the matrix will look like the identity matrix. They switch places. The identity matrix is now the inverted matrix.

In python the code for inverting matrices looks as follows:



At first the identity matrix will be generated with the exact dimensions as for the matrix to invert. The calculation will be from column to column. In the calculation block, we have to check at first, if a 1 is already on its diagonal place in the matrix. If this is the case, we don’t need to divide 1 through the value in the column and multiply the result with every element in the row. Multiplying has to be applied throughout the matrix and identity matrix. After this step, we check if the rest of the values in the column are 0 or not. If this is not the case, we have to multiply the columns with the negative factor, to switch to 0 values. This must be done throughout the matrix and identity matrix. And now we can go to the next column. Now you can see, that the matrix will turn into the identity matrix and the identity matrix will change into the inverted matrix.

To check if the conversion to the identity matrix was successful, you have to multiply the matrix with its inverse matrix. The result matrix should be the identity matrix, with its 0 and 1 in diagonal order.

A good step by step explanation is shown on [YouTube](https://www.youtube.com/watch?v=w2t9VADaw10)[[4]](#footnote-4).

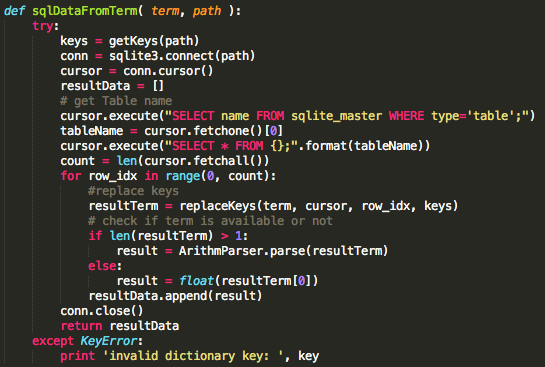
## Datasources

The data for the calculation of the multiple linear regression can be parsed from 3 data sources. SQL, Excel and as CSV file. The structure looks nearly identical in all three classes. There is a method *getKeys,* and the calculation method, which replaces the variables with the parameters stored in the data source. If the variable is part of a term, the term will be calculated.

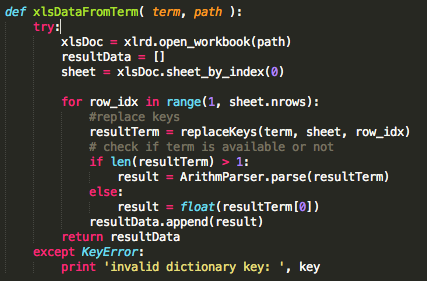
The code below shows the CSV Implementation of the parser. The *replaceKeys* method is a Helper for retrieving the value behind the variable.



The sql data parser needs the additional python library sqlite3. Now it is possible to execute sql statements and read the data from the database file.



The excel data parser needs the library xlrd for reading excel file type. As the default behavior the first workbook will be chosen to retrieve the data. Therefore the x Values have to be defined in the first workbook.



The Helper class *ArithmParser* will be used for calculating the terms. It uses the python library operation, where you can calculate terms by adding the fractions of the term in an array. The following code snippet shows the *ArithmParser* class.



These are the arithmetic operations which will be supported: +,-,\*,/,^. In the *parse* Method will be checked if the component is an operator or a value. If the there is only one element in the *calcArray*, then we assume, that the term looks like this: + 2 or – 2. If no “-” or “+” is used, an exception will be thrown.

## Installation guide

You have to prepare your machine before you can use the Muliple Linear Regression python script.

First thing you need is a python environment. This script was not tested for python 3. Please install a python 2.7 version on your computer. After installing python, you can use the pip (Python Package Index) for installing the following packages by typing “pip install <packagename>” into the commandline:

* pyparsing
* xlrd

After installing the packages the Multiple Linear Regression script should be fully functional.

1. http://reliawiki.org/index.php/Multiple\_Linear\_Regression\_Analysis [↑](#footnote-ref-1)
2. https://www.mathsisfun.com/algebra/matrix-multiplying.html [↑](#footnote-ref-2)
3. http://www.vikparuchuri.com/blog/inverting-your-very-own-matrix/ [↑](#footnote-ref-3)
4. https://www.youtube.com/watch?v=w2t9VADaw10 [↑](#footnote-ref-4)