Macroeconomic Crude Oil Analysis using Machine Learning

Overview

The aim of this Research Project is to analyse macroeconomic data about Crude Oil and try to understand how each variable could impacts others in order to predict possible future price shock in the market. Data that we are going to use are public and provided from the U.S. Energy Information and Administration; in particular we will explore this variables: Export,Import,Refinery Capacity,Price, Stocks,Product Supplied and Production. Major premise: these data are not comprehensive of all of the informations that affect crude oil price and his variation during the time but we could assume that they are correlated with the trend of this commodity; beside that we are going to focus only on a limited period of time of these variables (1992-2020), so models that we will obtain could be imprecise due to lack of some information (or variables) that are potentially correlated with the problem. Our studies will be focused on exploratory data analysis and the application of some supervised machine learning models like Multiple Linear Regression and Logistic Regression. Goal of this project is to discover if there could be a statistical advantage in predicting price change in order to take financial decisions.

Dataset

The dataset has 1482 rows (observations) and 9 columns (variables, first one is the date). It is composed by weekly data in the period 1992-2020.

Variables:

-Export: Amount of crude oil barrels exported every week

-Import: Amount of crude oil barrels imported every week

-Refinery: Amount of crude oil barrels refined every week

-Product Supplied: Amount of crude oil barrels supplied every week

-Stock: Amount of crude oil barrels stocked every week

-Production: Amount of crude oil barrels producted every week

-Price: commodity price

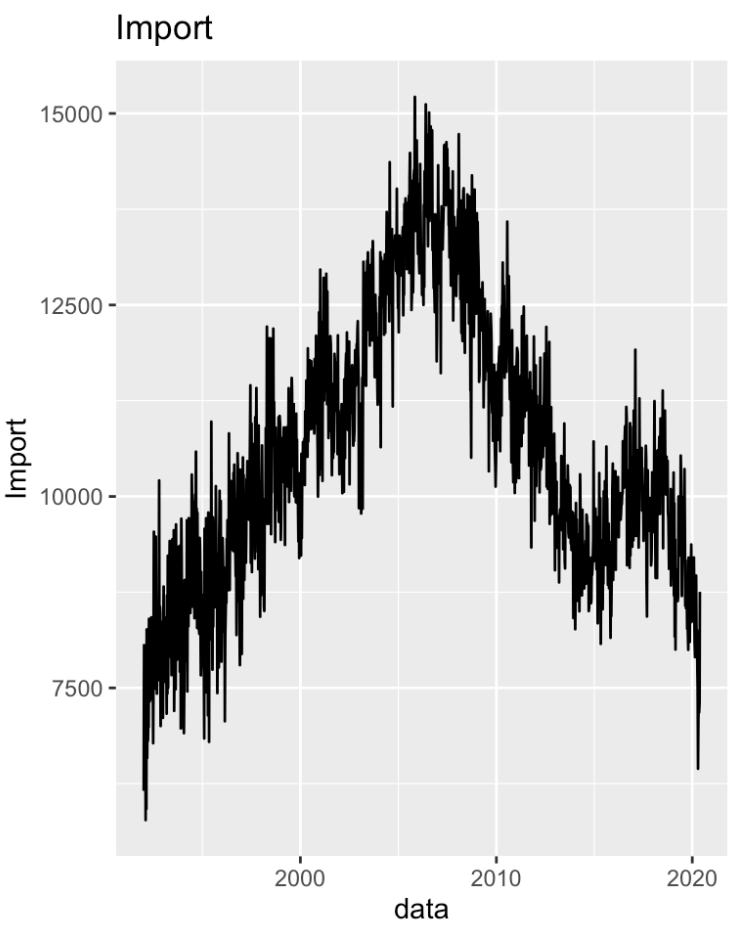
Let's look how it is:

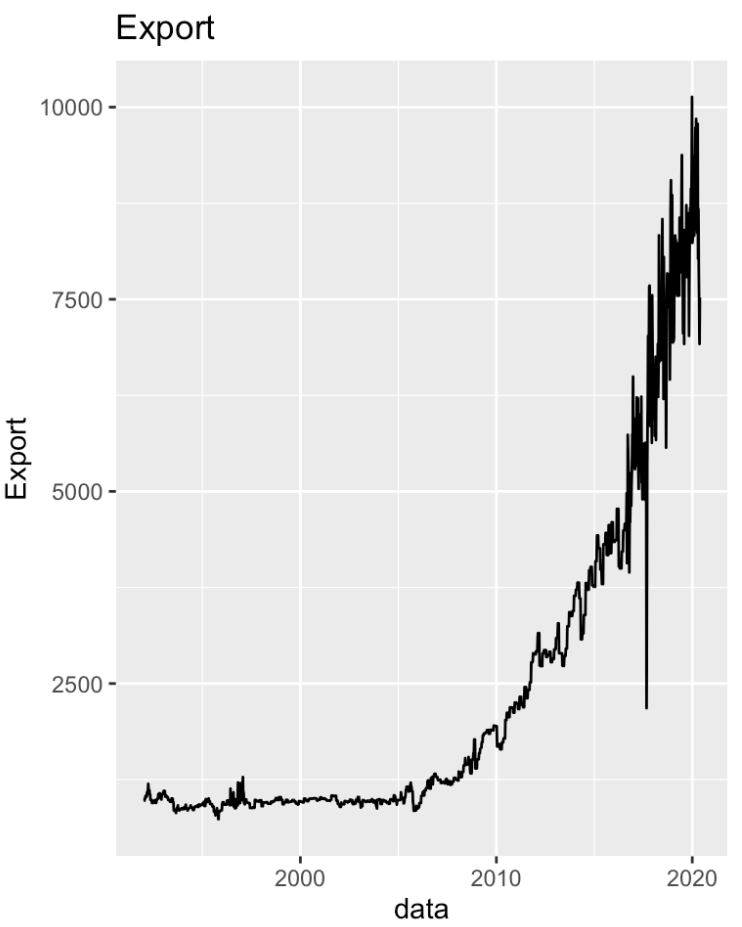
```
##         data Export Import Refinery Product_Supplied   Stock Product
## 1 1992-01-03    976   6157    13455             16707 1610942       7
## 2 1992-01-10    976   8059    13519             16923 1610172       7
## 3 1992-01-17    976   7479    13191             16600 1606059       7
## 4 1992-01-24   1007   6803    12780             17852 1597722       7
## 5 1992-01-31   1007   7713    12586             17319 1584233       7
## 6 1992-02-07   1039   5775    12903             17190 1574898       7
```

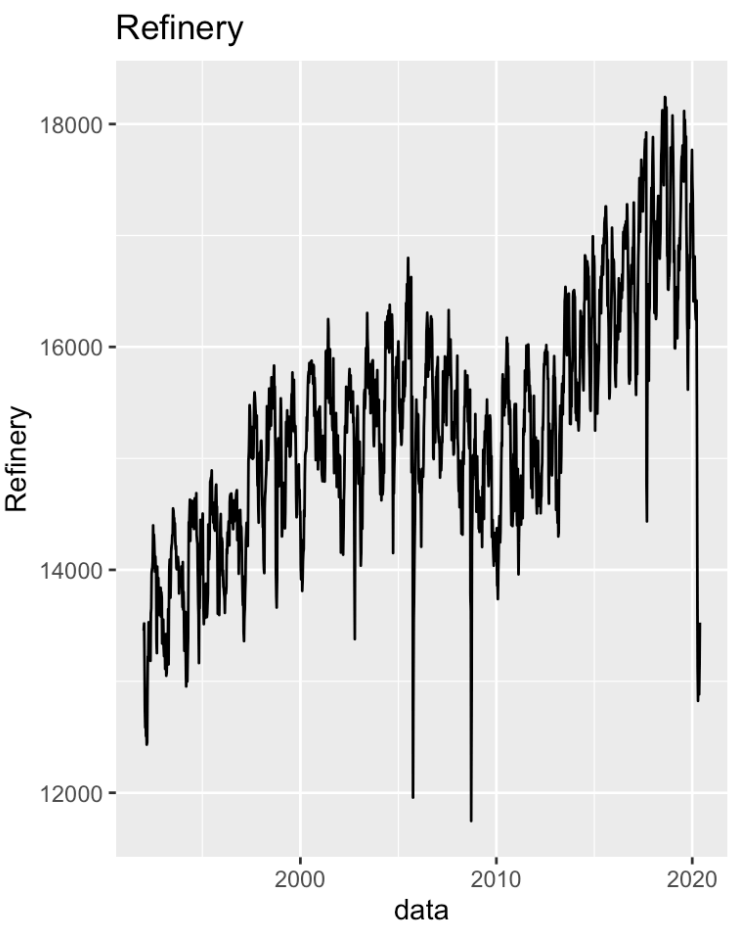
Exploratory Data Analysis

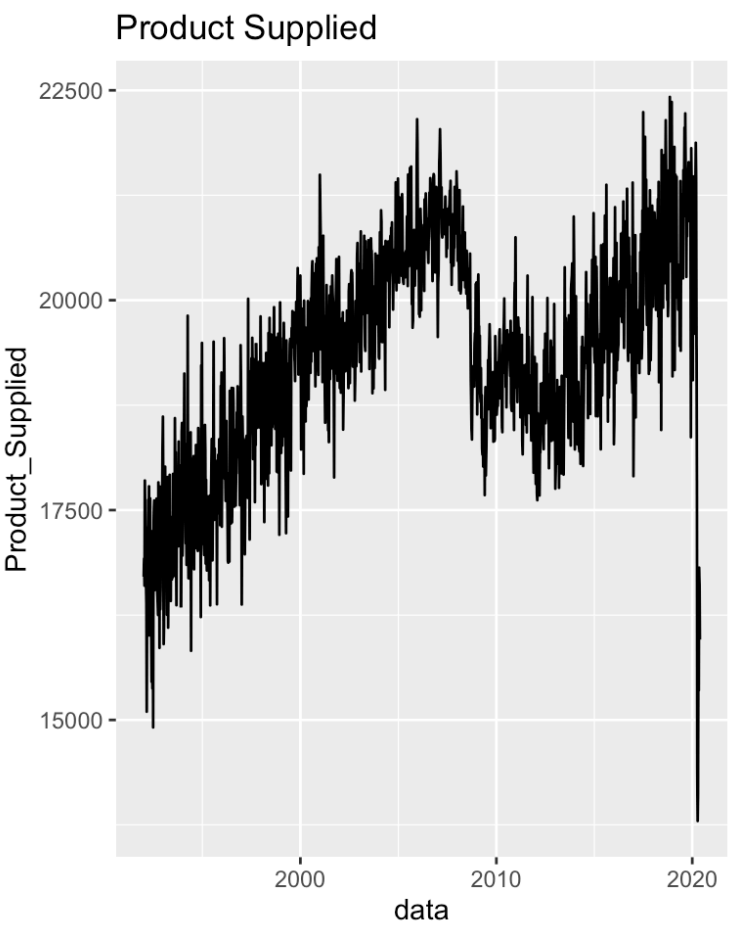
The first thing that we can do is to look at raw data in order to have a basic idea how they are and how we can analyse them. Let's plot time series.

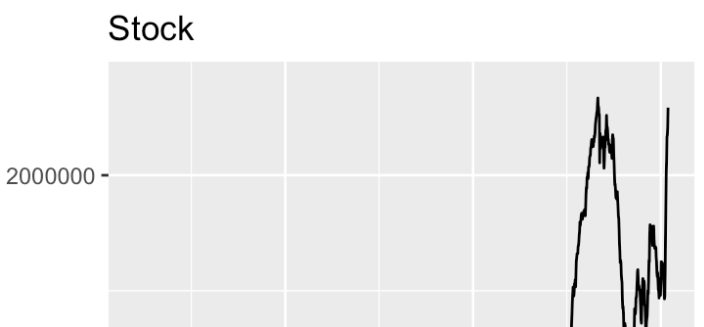
Time series plot

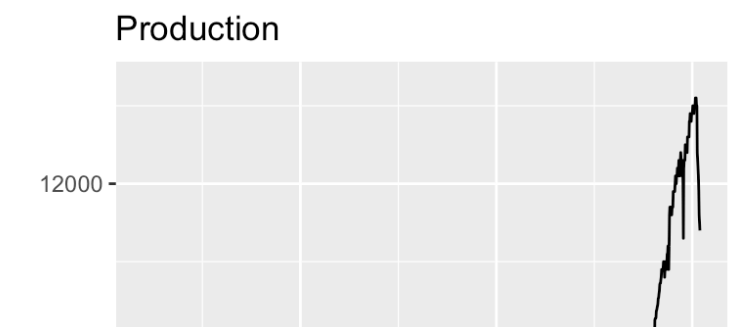

Import
15000
12500
10000
7500
Import
2000
2010
2020
data

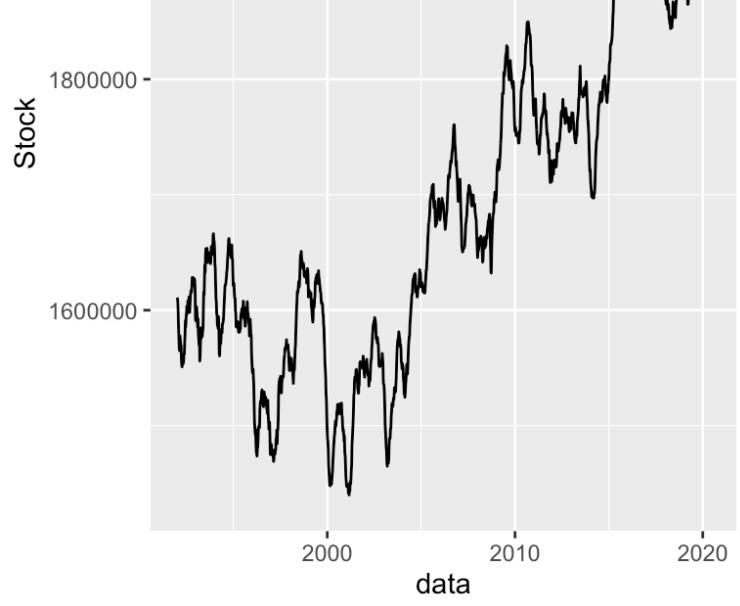

Export
10000
7500
5000
2500
Export
2000
2010
2020
data

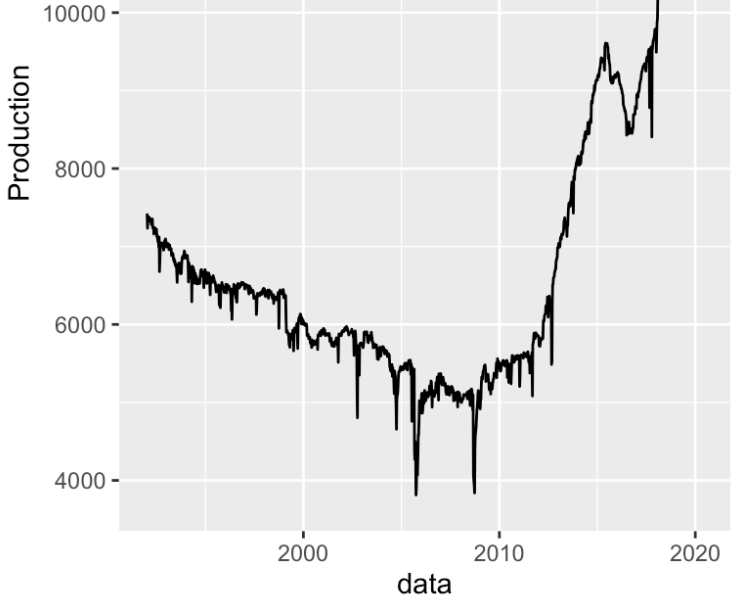

Refinery
18000
16000
14000
12000
Refinery
2000
2010
2020
data

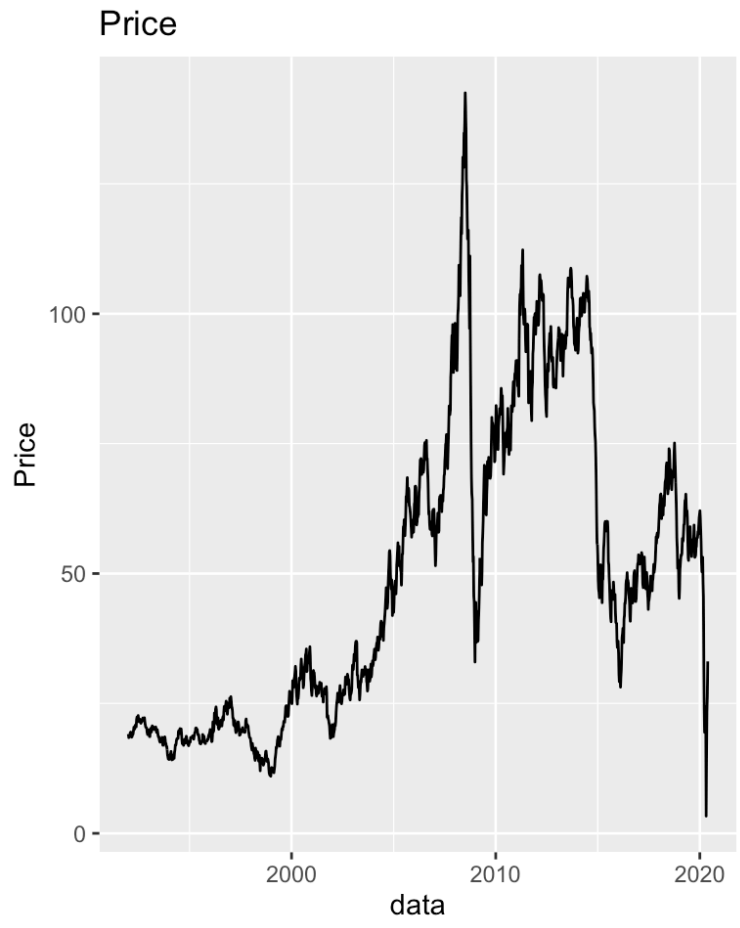

Product Supplied
22500
20000
17500
15000
Product_Supplied
2000
2010
2020
data


Stock
2000000


Production
12000





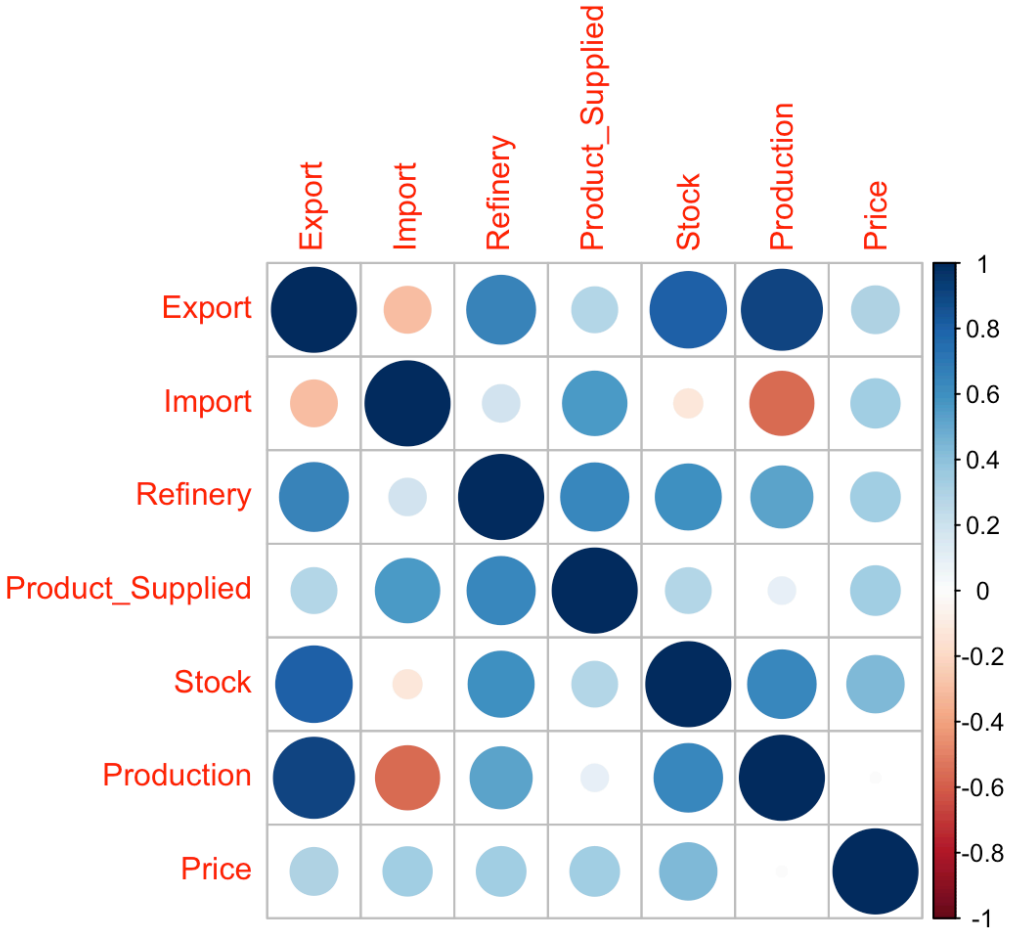


Now we can compute basic statistics and correlation plot.

```
summary(df)
```

```
##      data                Export          Import          Refinery    
##  Length:1482        Min.   :  723   Min.   : 5775   Min.   :11747  
##  Class :character   1st Qu.:  965   1st Qu.: 9353   1st Qu.:14550  
##  Mode  :character   Median : 1136   Median :10414   Median :15280  
##                     Mean   : 2335   Mean   :10625   Mean   :15299  
##                     3rd Qu.: 2944   3rd Qu.:11858   3rd Qu.:15918  
##                     Max.   :10134   Max.   :15217   Max.   :18243  
##  Product_Supplied      Stock           Production        Price       
##  Min.   :13797    Min.   :1439975   Min.   : 3813   Min.   :  3.32  
##  1st Qu.:18469    1st Qu.:1575688   1st Qu.: 5596   1st Qu.: 21.70  
##  Median :19386    Median :1665325   Median : 6356   Median : 45.31  
##  Mean   :19311    Mean   :1696305   Mean   : 6892   Mean   : 49.34  
##  3rd Qu.:20293    3rd Qu.:1788547   3rd Qu.: 7340   3rd Qu.: 70.01  
##  Max.   :22421    Max.   :2066939   Max.   :13100   Max.   :142.52
```

Correlation plot

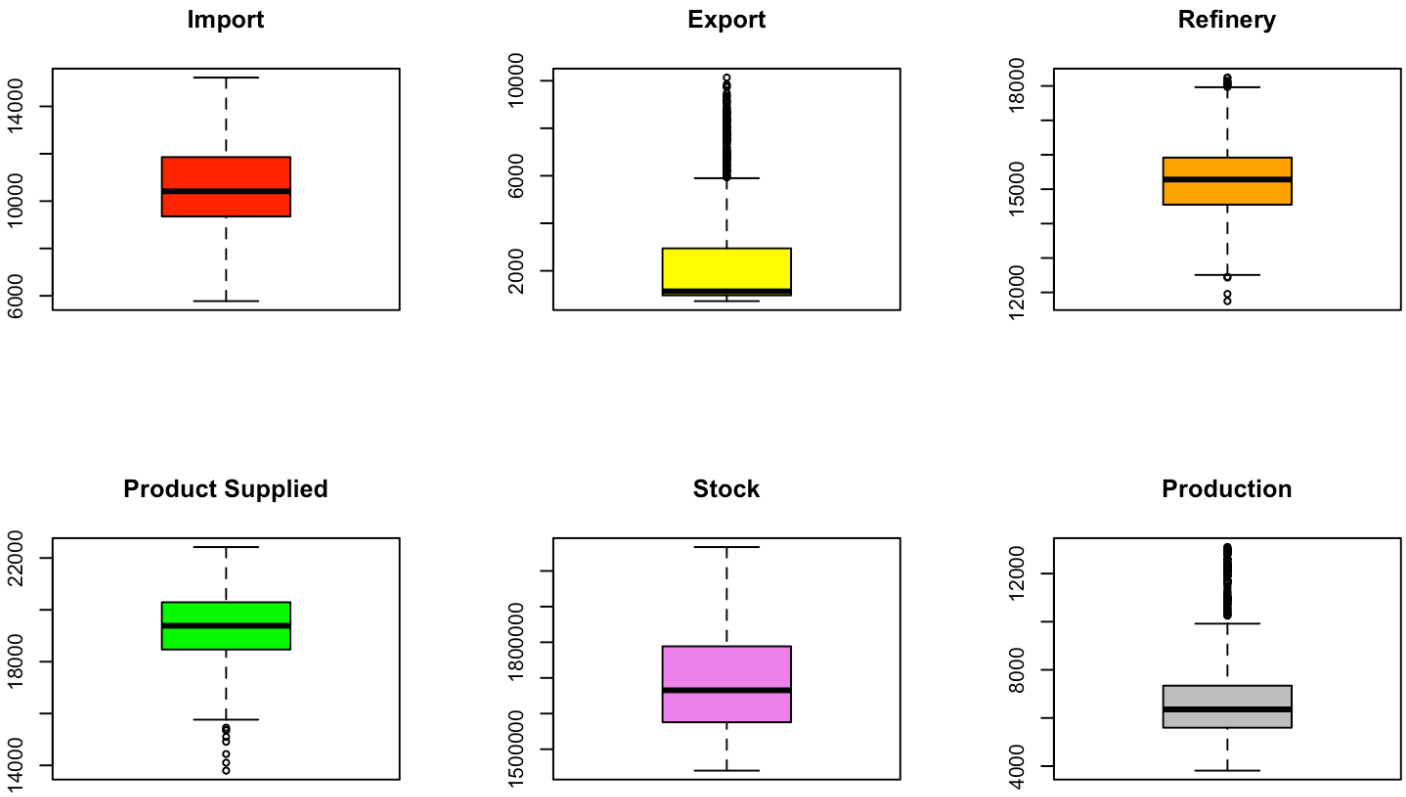


Correlation values

```
##                    Export     Import  Refinery Product_Supplied  
## Export           1.0000000 -0.2972313 0.6519375       0.28509549  0
## Import          -0.2972313  1.0000000 0.1883126       0.56936199 -0
## Refinery         0.6519375  0.1883126 1.0000000       0.63797407  0
## Product_Supplied 0.2850955  0.5693620 0.6379741       1.00000000  0
## Stock            0.8057500 -0.1118421 0.5981112       0.28309364  1
## Production       0.9062245 -0.5642425 0.5285764       0.09949712  0
## Price            0.3098645  0.3300835 0.3353705       0.33501408  0
##                  Production      Price
## Export           0.90622451 0.30986447
## Import          -0.56424246 0.33008346
## Refinery         0.52857637 0.33537046
## Product_Supplied 0.09949712 0.33501408
## Stock            0.64512612 0.44939094
## Production       1.00000000 0.01156482
## Price            0.01156482 1.00000000
```

As we can observe Export and Production is negative correlated with Import, while Price seems to be uncorrelated with the amount of crude oil producted; all of the others variables seems to have strong correlation between each other. If we want to look deeper at the distribution of our data we can compute boxplots.

Boxplot



As we can notice just from the previous plot, we are working with time series data; this means we need to adjust our analysies to these specific kind of data. The first thing we can do is to test the series for stationarity or non-stationarity in order to see if we can compute multiple linear regression with the original values or with the “log-differeces”.

Test for non-stationarity

We are going to use the Augmented Dickey-Fuller Test to discover the nature of these series.

ADF test

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$Export
## Dickey-Fuller = -0.69718, Lag order = 11, p-value = 0.9708
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$Import
## Dickey-Fuller = -1.783, Lag order = 11, p-value = 0.6702
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$Refinery
## Dickey-Fuller = -6.1262, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$Product_Supplied
## Dickey-Fuller = -2.4101, Lag order = 11, p-value = 0.4047
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$Stock
## Dickey-Fuller = -2.9265, Lag order = 11, p-value = 0.1861
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$Production
## Dickey-Fuller = -0.83609, Lag order = 11, p-value = 0.9584
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  df$Price
## Dickey-Fuller = -2.7302, Lag order = 11, p-value = 0.2692
## alternative hypothesis: stationary
```

The null hypothesis of non-stationarity is accepted for each variable except for "Refinery" so this means that it's not possible to regress "Price" on the other variables with the original values. One way to solve this problem is to trasform our data taking the first difference for each time series, in this case, the log-first-difference in order to obtain the percentage change. Due to the fact that we are using weekly data, the new dataset will show how each variable changes week by week in percentage (for instance "Production"=-0,11 means that the crude oil production decreases of 11% from the previous week). Let's look the new data.

Final dataset

```
##     d_l_date d_l_Export  d_l_Import d_l_Refinery d_l_Prod_Suppl
## 1 1992-01-10 0.00000000  0.26919983  0.004745319    0.012845850 -0.0
## 2 1992-01-17 0.00000000 -0.07469039 -0.024561325   -0.019270948 -0.0
## 3 1992-01-24 0.03126831 -0.09473540 -0.031653330    0.072712851 -0.0
## 4 1992-01-31 0.00000000  0.12554353 -0.015296364   -0.030311382 -0.0
## 5 1992-02-07 0.03128310 -0.28936896  0.024874757   -0.007476345 -0.0
## 6 1992-02-14 0.00000000  0.31661024 -0.031091403   -0.033119526 -0.0
##       d_l_Prod    d_l_Price
## 1 -0.001618123 -0.037317763
## 2 -0.023900079  0.021494648
## 3  0.014817118 -0.008006448
## 4  0.001904762  0.020684906
## 5  0.003392364  0.019236361
## 6 -0.002441341  0.003598051
```

Finally we can test again for non-stationarity.

ADF test on new data

```
##
##  Augmented Dickey-Fuller Test
##
## data:  d_l_df$d_l_Export
## Dickey-Fuller = -15.145, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  d_l_df$d_l_Import
## Dickey-Fuller = -15.168, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  d_l_df$d_l_Refinery
## Dickey-Fuller = -14.096, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  d_l_df$d_l_Prod_Suppl
## Dickey-Fuller = -14.719, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  d_l_df$d_l_Stock
## Dickey-Fuller = -8.88, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  d_l_df$d_l_Prod
## Dickey-Fuller = -13.006, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

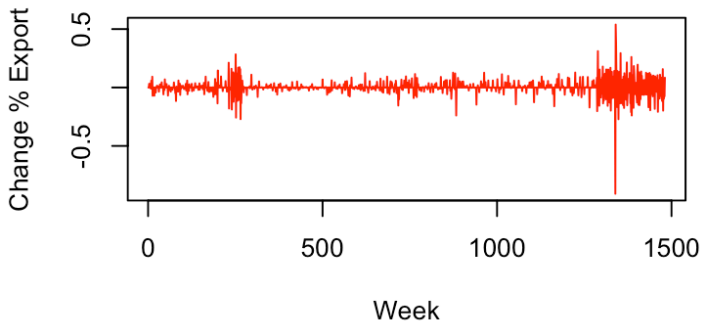
```
##
##  Augmented Dickey-Fuller Test
##
## data:  d_l_df$d_l_Price
## Dickey-Fuller = -11.036, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

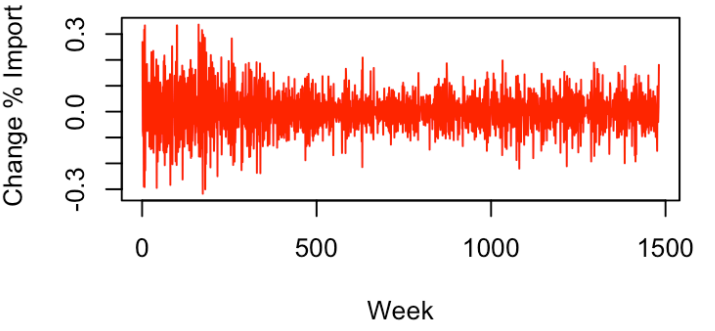
The null hypothethesis of non-stationarity is refused for each variable, this means that each one is stationary and we can use them in regression model in order to continue our analysis.

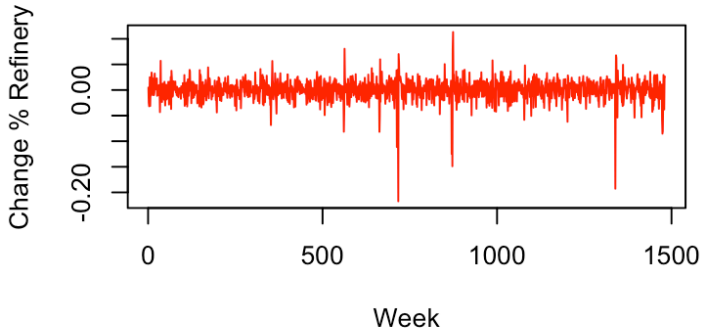
Exploratory data analysis on new data

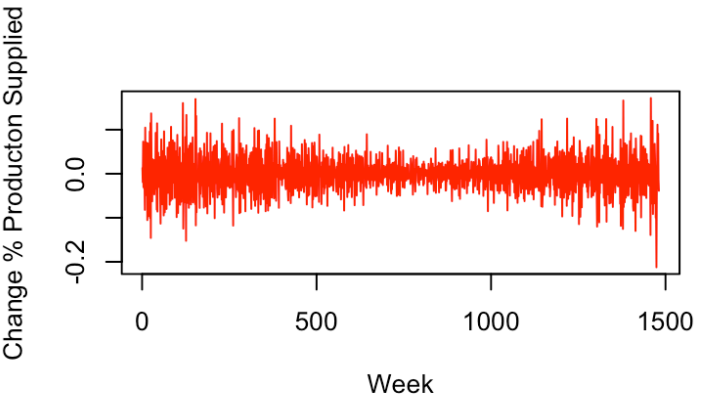
At this point we can plot the new data to see how they are.

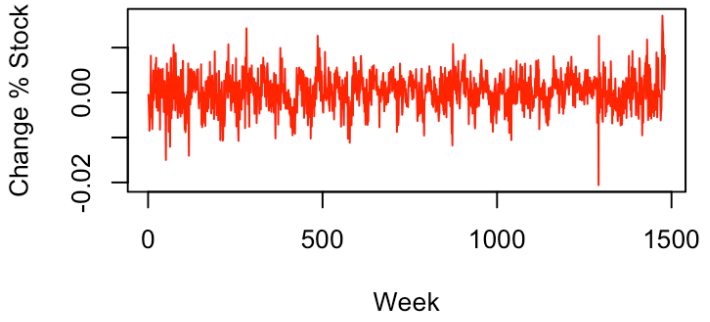
Plot

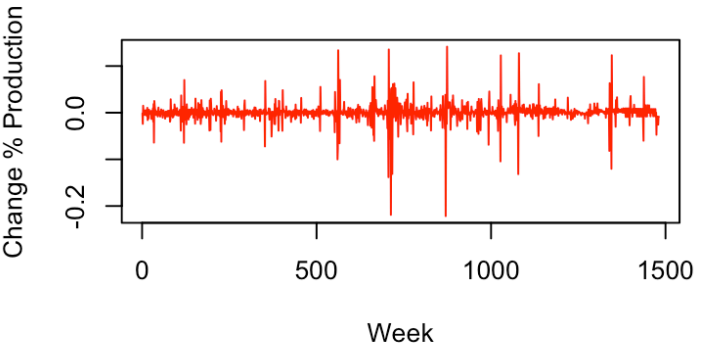


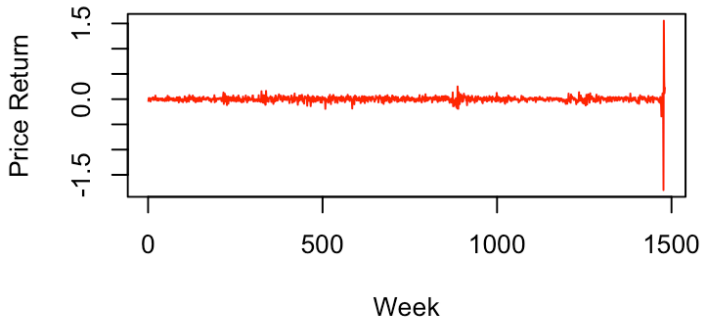








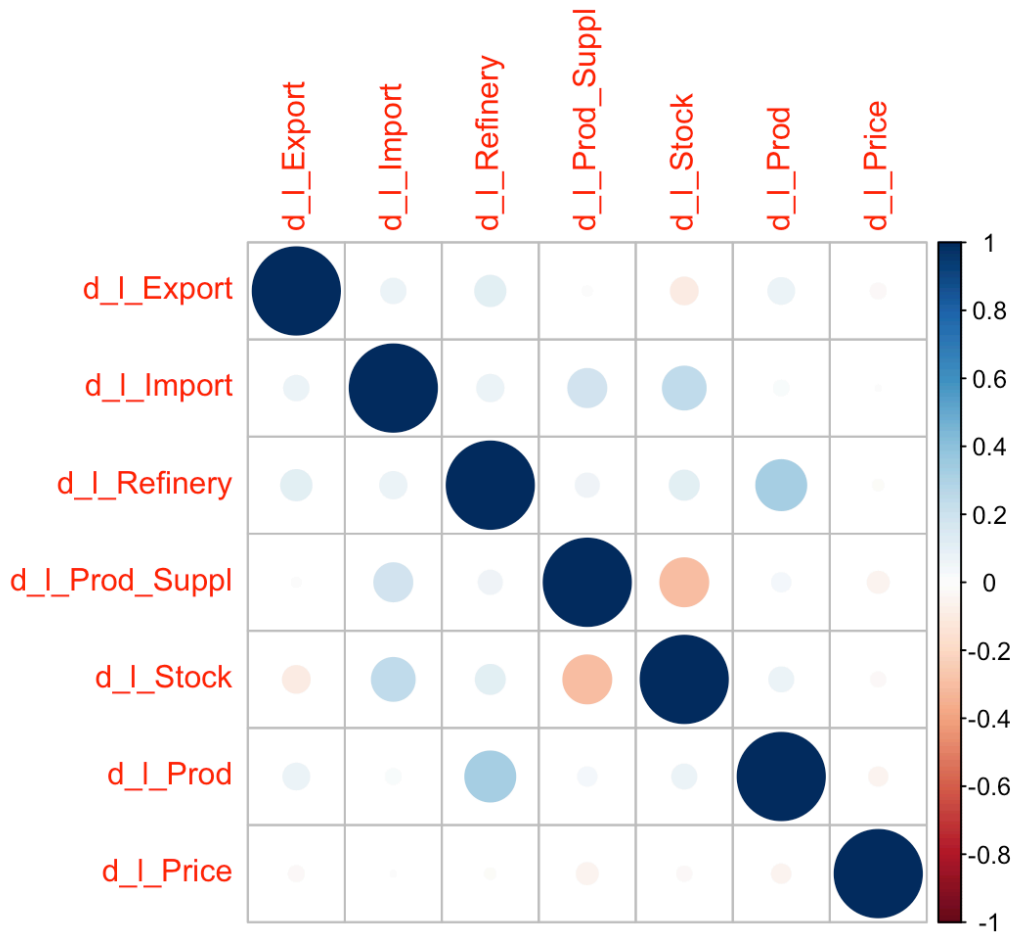




We can compute again basic statistics, correlation and boxplots.

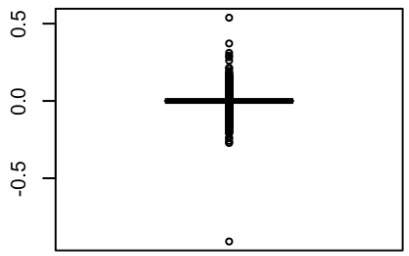
Correlation plot

```
##    d_l_Export          d_l_Import          d_l_Refinery
##  Min.   :-0.909109   Min.   :-0.3174366   Min.   :-0.2172889
##  1st Qu.: 0.000000   1st Qu.:-0.0492871   1st Qu.:-0.0096963
##  Median : 0.000000   Median :-0.0014599   Median : 0.0010741
##  Mean   : 0.001379   Mean   : 0.0002377   Mean   : 0.0000036
##  3rd Qu.: 0.000000   3rd Qu.: 0.0482908   3rd Qu.: 0.0112944
##  Max.   : 0.538423   Max.   : 0.3366393   Max.   : 0.1131799
##  d_l_Prod_Suppl         d_l_Stock            d_l_Prod
##  Min.   :-2.123e-01   Min.   :-0.0205806   Min.   :-0.2215752
##  1st Qu.:-2.469e-02   1st Qu.:-0.0021554   1st Qu.:-0.0035236
##  Median :-4.809e-05   Median : 0.0002891   Median : 0.0004376
##  Mean   :-3.097e-05   Mean   : 0.0001656   Mean   : 0.0002898
##  3rd Qu.: 2.260e-02   3rd Qu.: 0.0026038   3rd Qu.: 0.0046972
##  Max.   : 1.718e-01   Max.   : 0.0170986   Max.   : 0.1413802
##    d_l_Price
##  Min.   :-1.8017496
##  1st Qu.:-0.0240976
##  Median : 0.0024722
##  Mean   : 0.0003709
##  3rd Qu.: 0.0255795
##  Max.   : 1.5543327
```

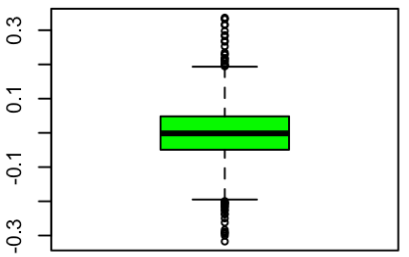


As we can see the correlation between each variable is very low due to data transformation and this leads to the hypothesis that we will not suffer of multicollinearity between the regressors; we will use later the Variance Inflation Factor to test if there is this problem in our dataset.

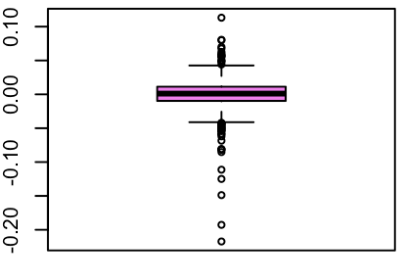
Boxplot



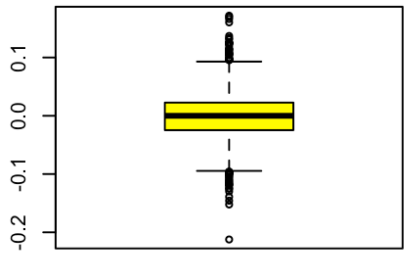
Export



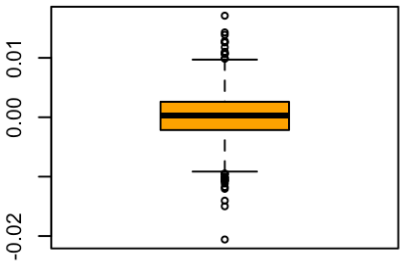
Import



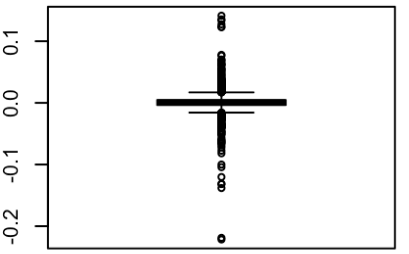
Refinery



Prodution Supplied



Stock



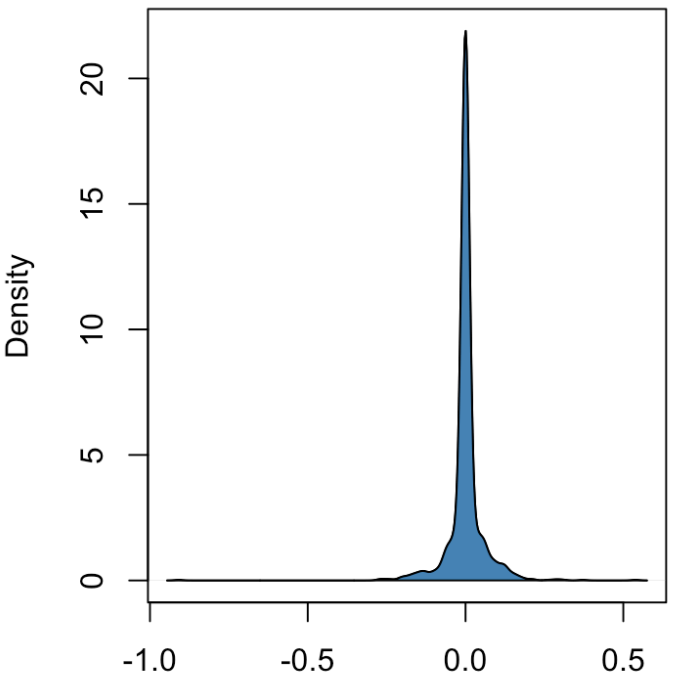
Production

Kernel Density Estimator

As we can see the distribution of percertage change of each variable is quite different from each other, so we can deduce that there are non symmetrical distributions and possible ouliers in our data. Besides this basic data visualization we can go deeper with our analysis using a Kernel Density function to have a better idea of how data are distributed.

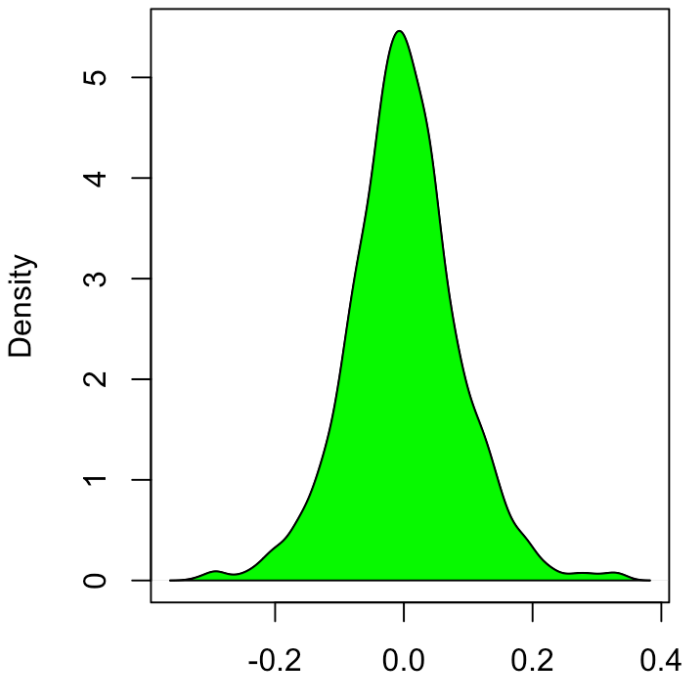
Kernel density plot

Kernel Density Export



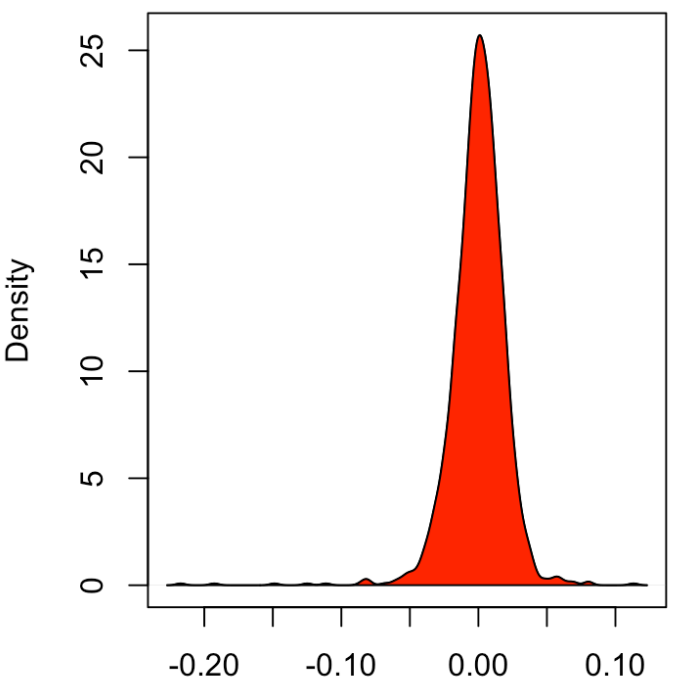
N = 1481 Bandwidth = 0.01217

Kernel Density Import



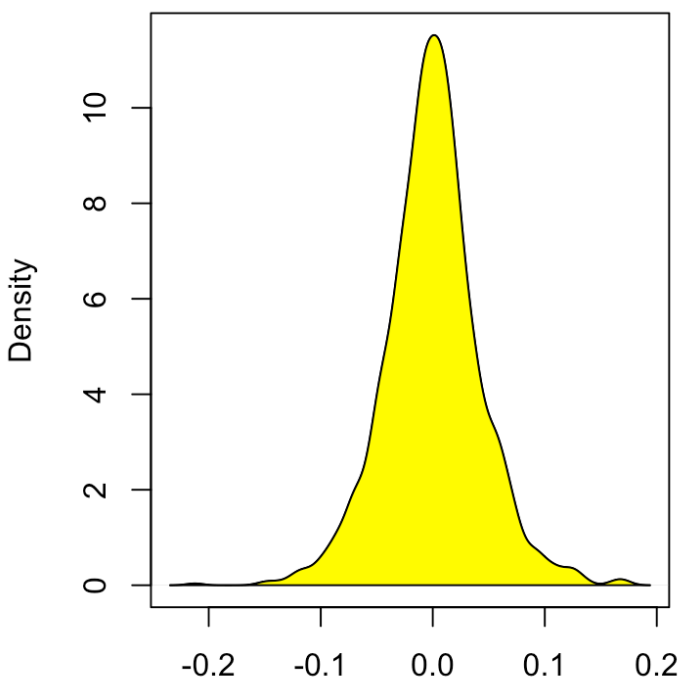
N = 1481 Bandwidth = 0.01522

Kernel Density Refinery



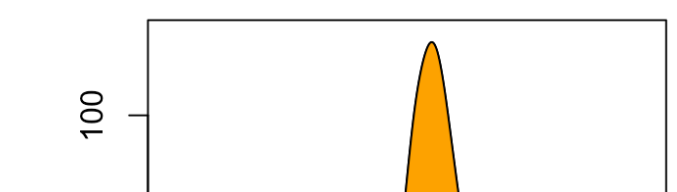
N = 1481 Bandwidth = 0.003274

Kernel Density Product Supplied

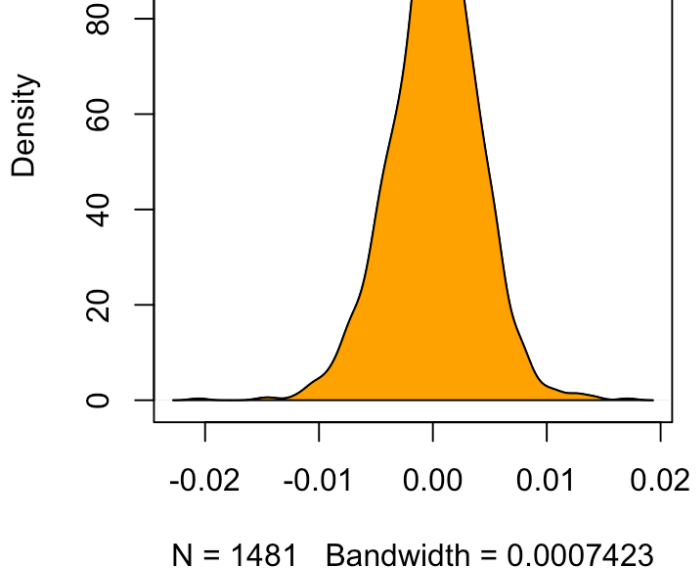


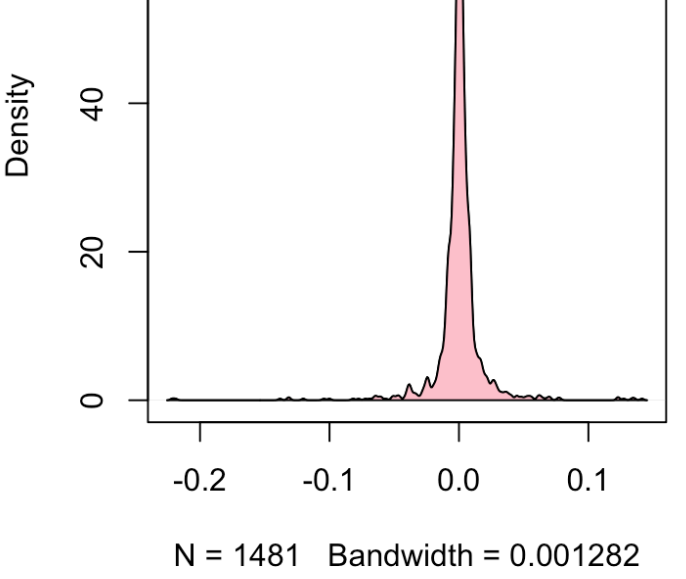
N = 1481 Bandwidth = 0.007376

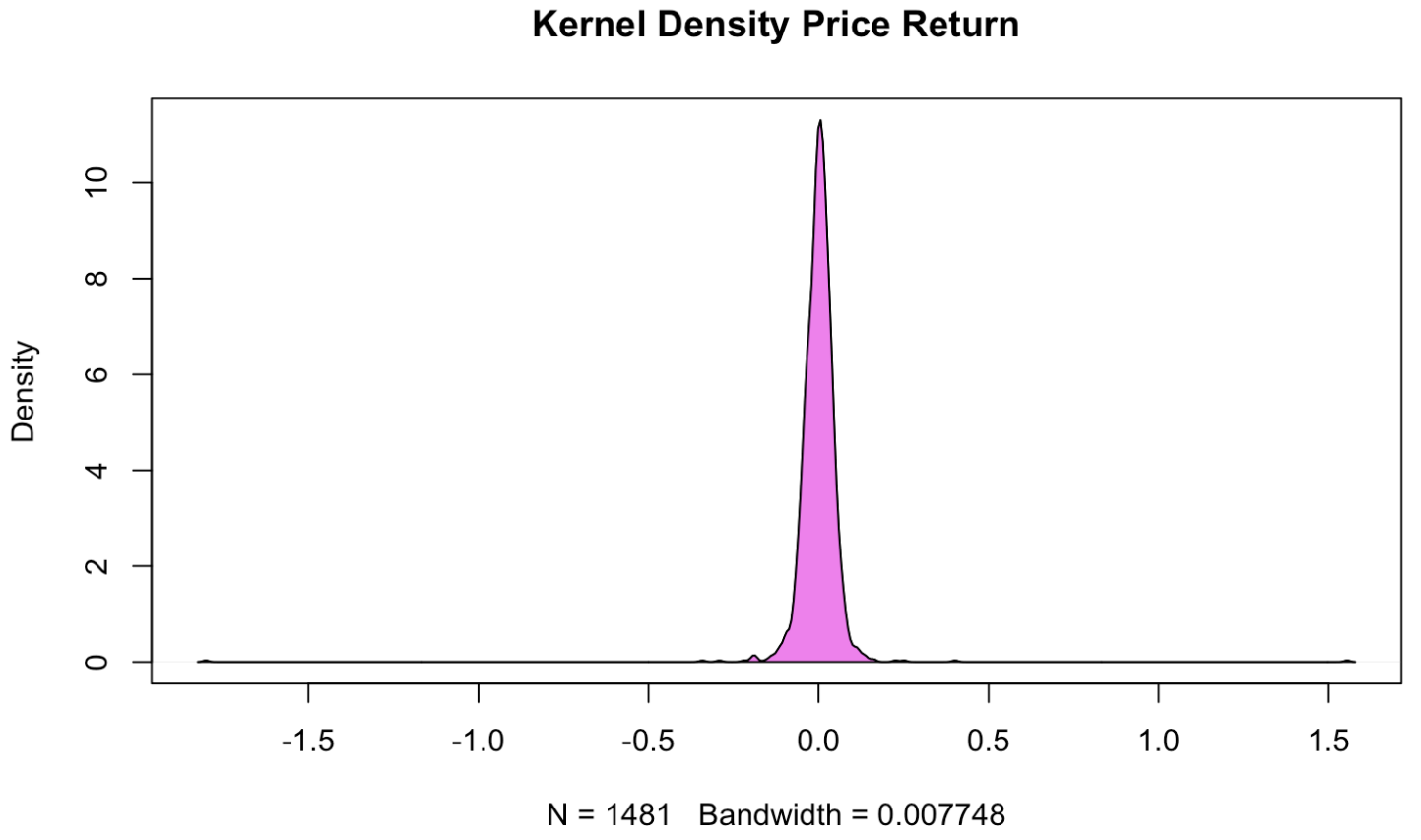
Kernel Density Stock



Kernel Density Production







Normality Test

Once we have obtained the the previous distributions we can test them for normality using Jarque-Bera test.

```
##
##  Jarque Bera Test
##
## data:  kd_Export$x
## X-squared = 30.72, df = 2, p-value = 2.134e-07
```

```
##
##  Jarque Bera Test
##
## data:  kd_Import$x
## X-squared = 30.72, df = 2, p-value = 2.134e-07
```

```
##
##  Jarque Bera Test
##
## data:  kd_Prod$x
## X-squared = 30.72, df = 2, p-value = 2.134e-07
```

```
##
##  Jarque Bera Test
##
## data:  kd_Prod_Suppl$x
## X-squared = 30.72, df = 2, p-value = 2.134e-07
```

```
##
##  Jarque Bera Test
##
## data:  kd_Refinery$x
## X-squared = 30.72, df = 2, p-value = 2.134e-07
```

```
##
##  Jarque Bera Test
##
## data:  kd_Stock$x
## X-squared = 30.72, df = 2, p-value = 2.134e-07
```

```
##
##  Jarque Bera Test
##
## data:  kd_Price$x
## X-squared = 30.72, df = 2, p-value = 2.134e-07
```

For each test we refuse the null hypothesis of normality distribution for a significance level of 0.05.

Multiple Linear Regression

Once we have obtained our new dataset with the finale variables we can start applying the fist machine learning model: Multiple Regression. In order to do that we have to create a “training” and a “test” dataset. The partition is 70% for the training data and 30% for the test data.

Training data dimension

```
## [1] 1000    7
```

Test data dimension

```
## [1] 481   7
```

Results model

Let's apply the model on the training data.

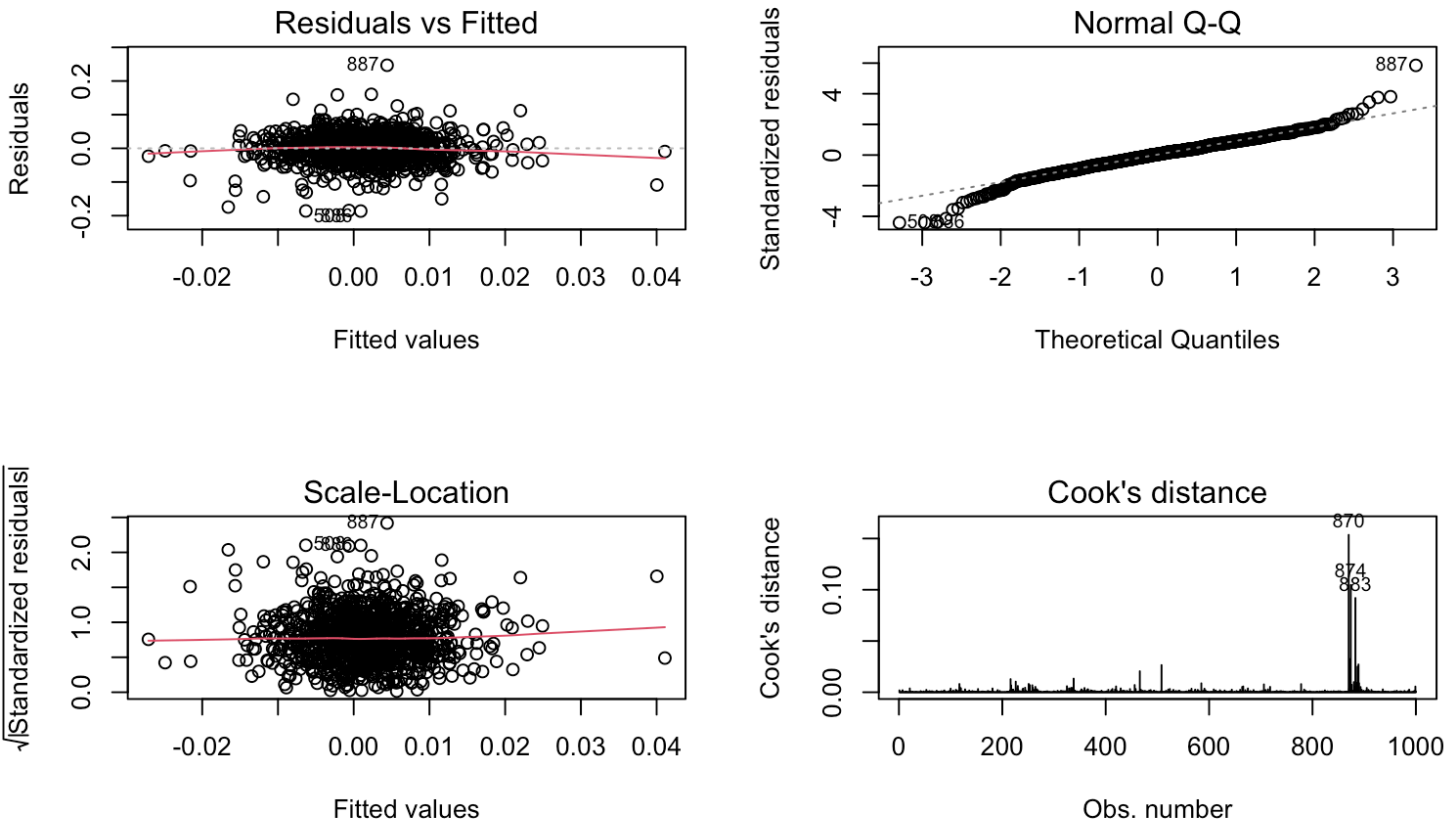
```
##
## Call:
## lm(formula = dl.df.train$d_l_Price ~ ., data = dl.df.train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.186379 -0.024177  0.001748  0.026802  0.246838
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.001678   0.001338   1.254 0.210133
## d_l_Export      0.056795   0.037425   1.518 0.129443
## d_l_Import      0.022533   0.015737   1.432 0.152496
## d_l_Refinery    0.117814   0.071230   1.654 0.098444 .
## d_l_Prod_Suppl -0.090326   0.036258  -2.491 0.012894 *
## d_l_Stock      -1.405013   0.382361  -3.675 0.000251 ***
## d_l_Prod       -0.182914   0.068869  -2.656 0.008035 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04229 on 993 degrees of freedom
## Multiple R-squared:  0.02552,    Adjusted R-squared:  0.01963
## F-statistic: 4.334 on 6 and 993 DF,  p-value: 0.0002508
```

Export, Import and Refinery has positive coefficients so this means that a change in those variables lead to an increase of price return while Product Supplied, Stock and Production (with negative coefficients) brings to a decrease in price return. This dynamic can be explained from a macroeconomical poit of view (in our analysis we will not cover this part). As we can see every coefficient is statistically significant at the 0.1 level except for Export and Import that have a p-value greater than 10%. Statistically this two coefficients are not relevant for the model if we consider this approach but if we look at the p-values of these regressors we can notice the they are still relatively small and it's very likely that

they have influence on the dependent variable. In addition to this reflection we can verify with the F-test the goodness of fit of this model and due to the fact that p-value is smaller than 0.001 we reject the null hypothesis and we accept the assumption of relevance of these specific variables.

Diagnostic plot

Another important analysis that we have to do is to look at the residual distribution with the aim of verify the assumption of normal distribution. To do that we use diagnostic plot.



From the previous plot we can verify that the relation between residual and fitted values is not totally linear and the same applies to theoretical quantiles and standardized residual. This leads to the conclusion than residuals are not normally distribuited. In order to be sure about this assumption we will use the Jarque-Bera test.

Normality test on residuals

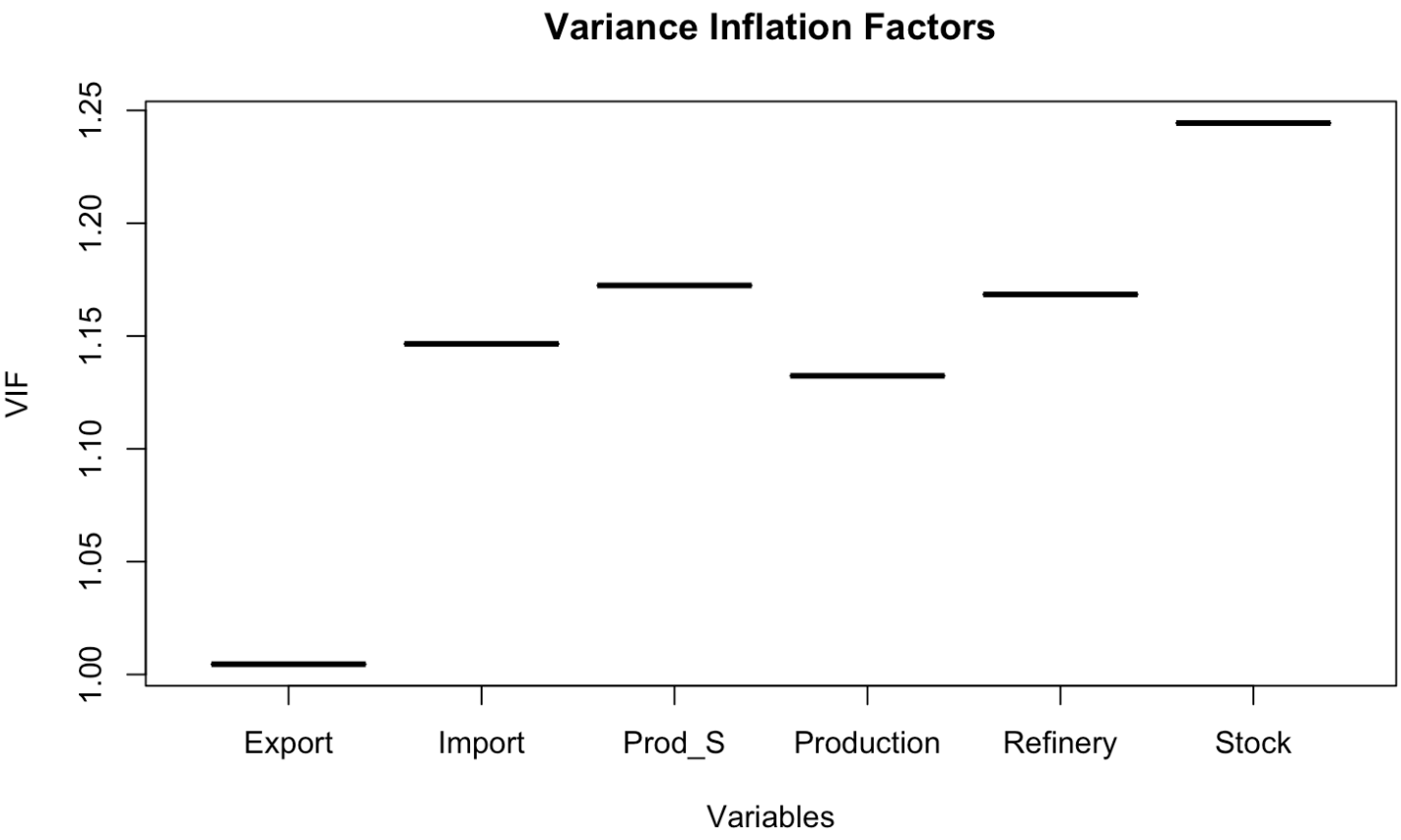
```
jarque.bera.test(train.model$residuals)
```

```
##
##  Jarque Bera Test
##
## data:  train.model$residuals
## X-squared = 339.74, df = 2, p-value < 2.2e-16
```

We reject the null hypothesis of normality distribuition at 0.01 level of significance.

VIF

At this point we can test for multicollinearity using the Variance Inflation Factors.



VIF for each regressor is always minor than 5 so we can exclude the possibility of multicollinearity;

Model accuracy

One way to test the accuracy of the model is to use it to make prediction; at first we will apply it to the training data (just remember that this procedure is adopted to give us a basic idea of the model performs on data, final valuation will be made using test data). Mean Square Error will be used to calculate model performance(also RMSE).

Mean Square Error

```
## [1] 0.001776202
```

Root Mean Square Error

```
## [1] 0.04214501
```

Obviously we could obtain better results in term of goodness of fit using R^2 as a comparison parameter with other linear models with different regressors; later we will try to optimize the accuracy of the model with feature selection.

Prediction

Now that we have the model we can apply it to the test data in order to see how well it performs. As before we evaluate the model using MSE.

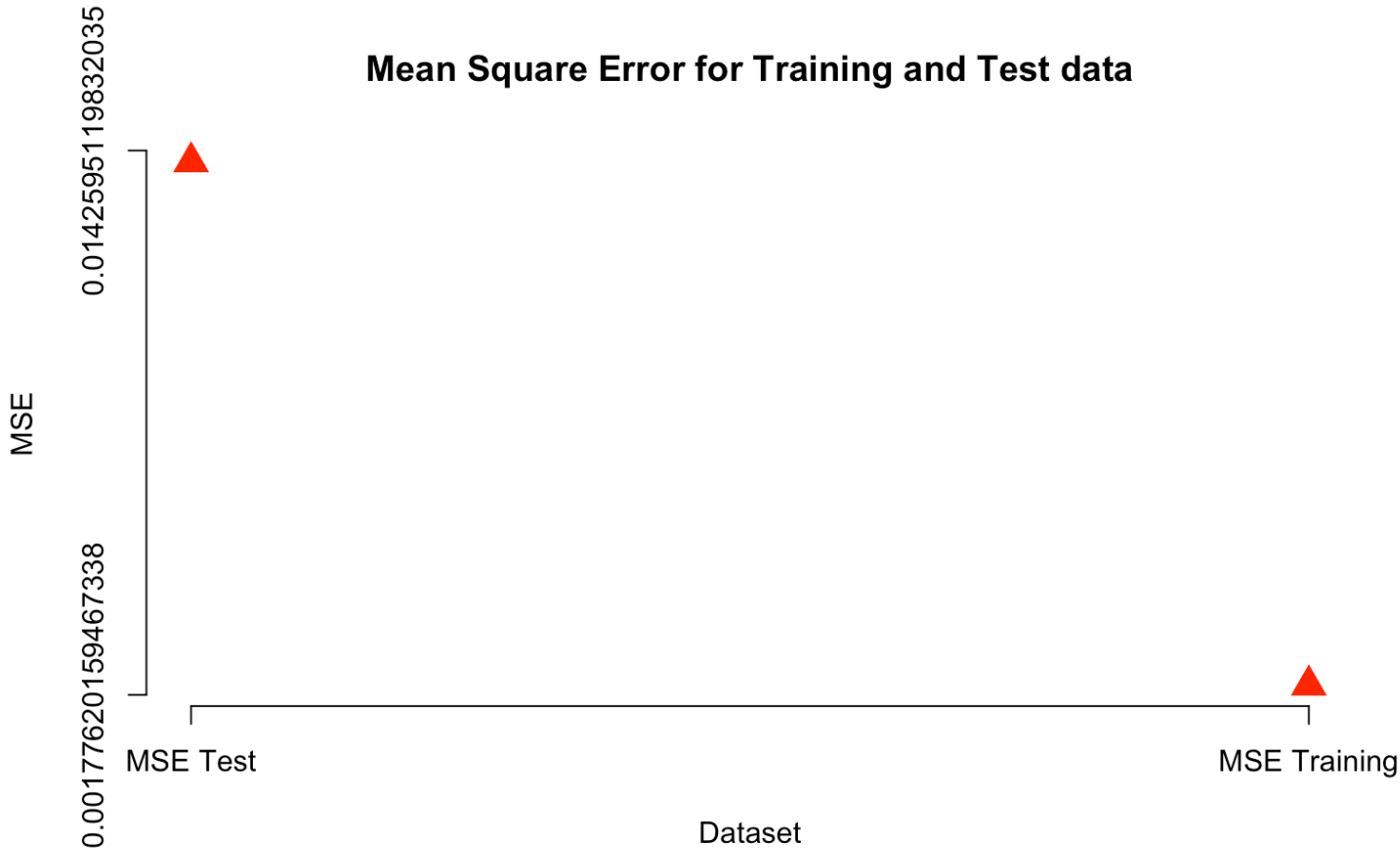
MSE

```
## [1] 0.01425951
```

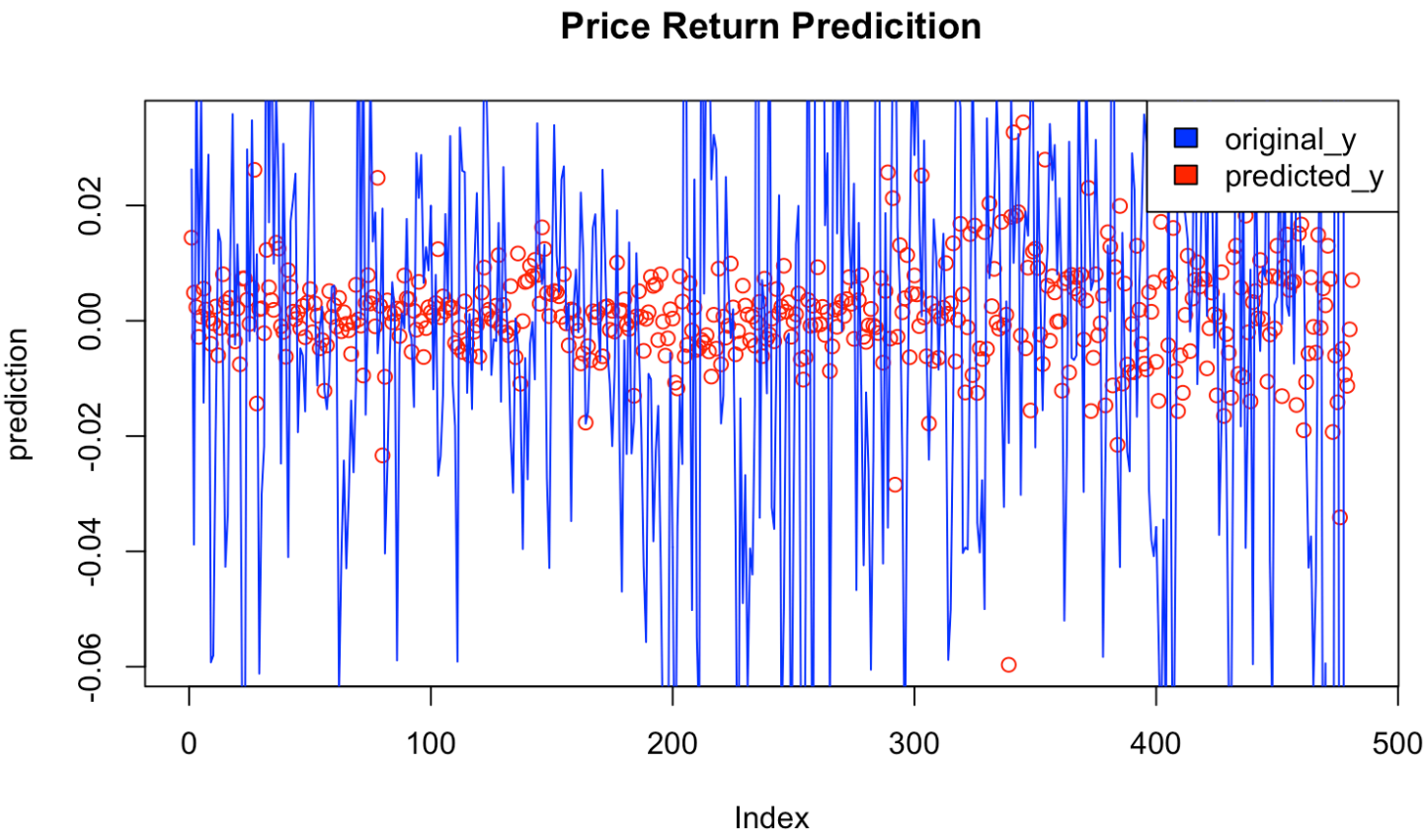
RMSE

```
## [1] 0.1194132
```

As we could expect MSE(and obviously RMSE) are much higher in the test data, so this means that our model in a real world application would underperform the accuracy previously calculated. Let's plot them.



Before move on, let's visualize a plot of predicted values.



Features selection

Features selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model.

Best Subsets Regression

As told before, one way to improve model accuracy is features selection; there are multiple ways and procedures to do it but the one that we are going to use it is called “Best Subsets Regression”.

Best six combination models

```
## Subset selection object
## Call: regsubsets.formula(dl.df.train$d_l_Price ~ ., data = dl.df.tra
##     nvmax = 6)
## 6 Variables  (and intercept)
##                Forced in Forced out
## d_l_Export         FALSE      FALSE
## d_l_Import         FALSE      FALSE
## d_l_Refinery       FALSE      FALSE
## d_l_Prod_Suppl     FALSE      FALSE
## d_l_Stock          FALSE      FALSE
## d_l_Prod           FALSE      FALSE
## 1 subsets of each size up to 6
## Selection Algorithm: exhaustive
##          d_l_Export d_l_Import d_l_Refinery d_l_Prod_Suppl d_l_Stock
## 1  ( 1 ) " "        " "        " "           " "            "*"
## 2  ( 1 ) " "        " "        " "           " "            "*"
## 3  ( 1 ) " "        " "        " "           "*"            "*"
## 4  ( 1 ) " "        " "        "*"           "*"            "*"
## 5  ( 1 ) "*"        " "        "*"           "*"            "*"
## 6  ( 1 ) "*"        "*"        "*"           "*"            "*"
```

Model with minimun adjusted R^2

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

Model with minimun BIC

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

Between all of the possible combination of independent variables the model with the lowest adjusted R^2 and BIC is the one that uses only “Stock” as regressor. From an economic point of view it is very unlikely that only one feature can explain price return change but matematically it is the best.

New Model

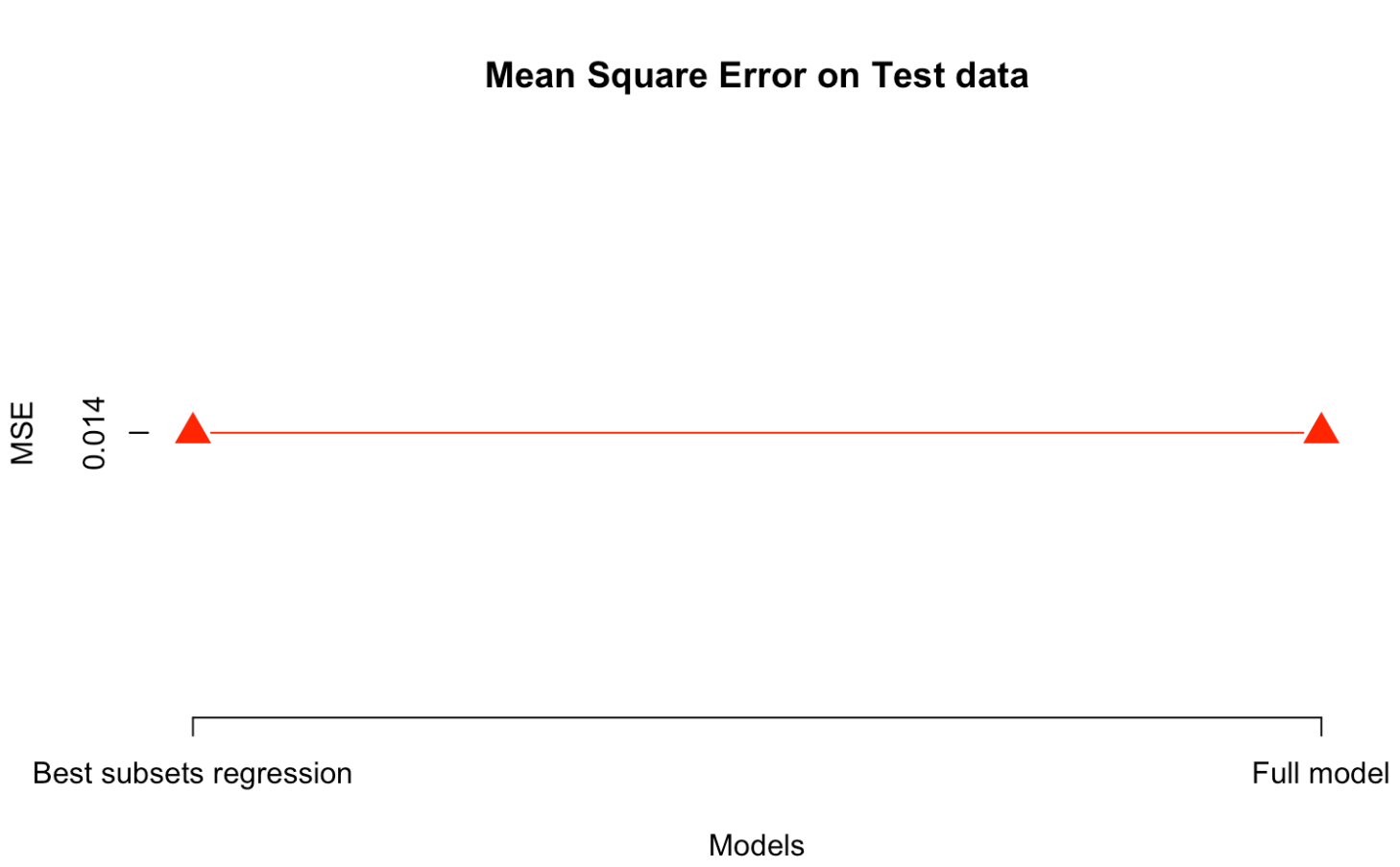
```
##
## Call:
## lm(formula = dl.df.train$d_l_Price ~ dl.df.train$d_l_Stock, data = d
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.190194 -0.025160  0.001478  0.026502  0.253775
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              0.001751   0.001346   1.301  0.19359
## dl.df.train$d_l_Stock -1.032529   0.344807  -2.995  0.00282 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04255 on 998 degrees of freedom
## Multiple R-squared:  0.008905,   Adjusted R-squared:  0.007912
## F-statistic: 8.967 on 1 and 998 DF,  p-value: 0.002817
```

MSE with the new model

```
## [1] 0.01368499
```

Comparison between MSE on test data

Althought the new MSE is lower than the one obtained with the full model, there isn't a big difference in their values so we didn't get a real improvement in the accuracy. Let's plot their values.

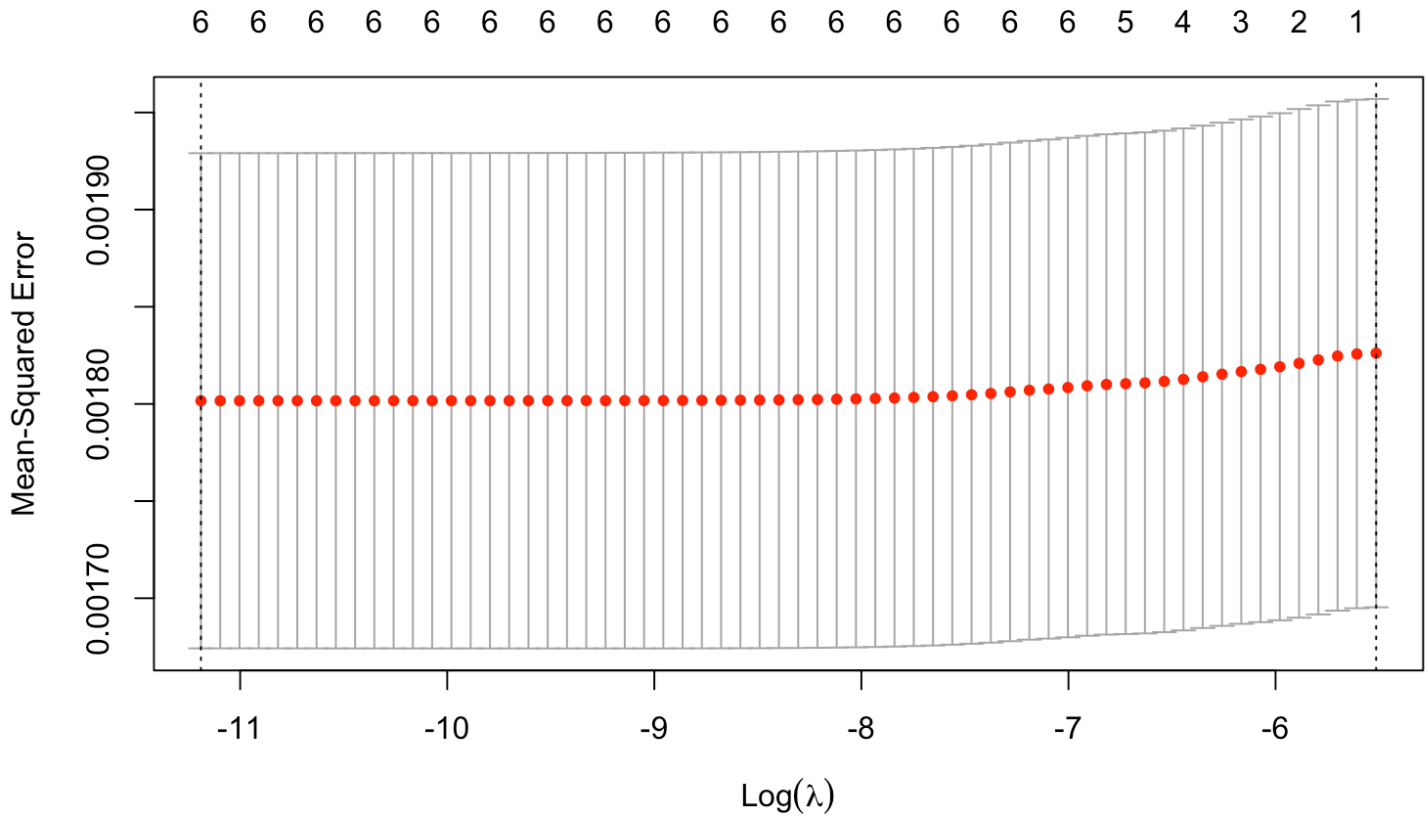


Lasso Regression

Best subsets regression seems to be useless to improve accuracy so we could decide to use another approach : Lasso regression. This Regularization technique assigns a penalty to the coefficients that are less related to the dependent variable. In order to implement this model we will perform a 10-fold cross-validation to find optimal value for lambda (unknown parameter for this regression model). We will choose the value that minimizes the mean squared error.

Optimal Lambda

Finally we can plot the relation between Lambda value and the corresponding mean-squared error; above the graph there is the relative number of regressors for each parameter's value.



Optimal value for Lambda

```
## [1] "Best Lambda: 1.38207697796775e-05"
```

Now we run the model using the new parameter; let's see the new coefficients.

```
## 7 x 1 sparse Matrix of class "dgCMatrix"
##                              s0
## (Intercept)     0.001678366
## d_l_Export      0.056432539
## d_l_Import      0.022264171
## d_l_Refinery    0.116557157
## d_l_Prod_Suppl -0.089642669
## d_l_Stock      -1.397461720
## d_l_Prod       -0.181977518
```

```
## [1] 0.0255194
```

As we can notice the coefficients are almost identical to the one obtained with OLS (sign and values); we expect similar predictions to the previous model.

Mean squared error with Lasso regression

```
## [1] "Lasso MSE: 0.014"
```

As we had anticipated before, the results in accuracy are identical to the one obtained with the firs model(OLS).

Until now, we have tried to predict the weekly change (in percentage) but with poor results; the mean squared erros is always too high to be used in a real-world implementation of these models. Just remember that MSE=0.014 consists in a root mean squared error of 0.12 so in order to take financial decisions the risk level due to inaccuracy could be dangerous for the potential investors. Are there any possible solutions? An alternative idea is to transform this regression problem to a classification problem. The aim of this approach is to classify as “1” weeks where the price change is positive and as “0” where it’s negative. From quantitative predictions now we will have binary response.

Logistic Regression

As explained before, we have to convert to binary values the “Price Change” the column of our dataset in order to apply this classification model. If the value is positive it will be converted to “1”, if it’s negative to “0”.

Price return converted to binary variable

```
##   d_l_Export  d_l_Import d_l_Refinery d_l_Prod_Suppl     d_l_Stock
## 1 0.00000000  0.26919983  0.004745319    0.012845850 -0.0004780955
## 2 0.00000000 -0.07469039 -0.024561325   -0.019270948 -0.0025576535
## 3 0.03126831 -0.09473540 -0.031653330    0.072712851 -0.0052044873
## 4 0.00000000  0.12554353 -0.015296364   -0.030311382 -0.0084784862
## 5 0.03128310 -0.28936896  0.024874757   -0.007476345 -0.0059098703
## 6 0.00000000  0.31661024 -0.031091403   -0.033119526 -0.0054296679
##   d_l_Price
## 1         0
## 2         1
## 3         0
## 4         1
## 5         1
## 6         1
```

Contingency table for training data

```
##
##   0   1
## 463 537
```

There are 463 weeks where the price return was negative and 537 where it was positive.

Contingency table for test data

```
##
##   0   1
## 237 244
```

There are 237 weeks where the price return was negative and 244 where it was positive.

##Results model

Let's run the model.

```
##
## Call:
## glm(formula = df.log.train$d_l_Price ~ ., family = binomial(link = '
##     data = df.log.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8453  -1.2145   0.9509   1.1166   1.4621
##
## Coefficients:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      0.15393    0.06405   2.403  0.01625 *
## d_l_Export       1.72181    1.81037   0.951  0.34156
## d_l_Import      -0.49449    0.75377  -0.656  0.51181
## d_l_Refinery     3.07752    3.42948   0.897  0.36952
## d_l_Prod_Suppl  -2.79827    1.74117  -1.607  0.10803
## d_l_Stock      -58.87458   18.53710  -3.176  0.00149 **
## d_l_Prod        -4.87154    3.44091  -1.416  0.15684
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1380.8  on 999  degrees of freedom
## Residual deviance: 1363.3  on 993  degrees of freedom
## AIC: 1377.3
##
## Number of Fisher Scoring iterations: 4
```

Variable "Stock" results statistically significant at 0.01 level while "Production" and "Product Supplied" althought has an higher p-value (but lower than 0.2) we can still consider them significant for the model. For all of the other variables we accept the null hypothesis of coefficients equal to zero. Negative coefficients for "Import","Product supplied","Stock" and "Production" mean that an increment in those variables decrease the log odds.

ANOVA

At this point using analysis of variance we can visualize the deviance table. The difference between the null deviance and the residual deviance shows how our model is doing against the null model (a model with only the intercept). Analyzing the table we can see the drop in deviance when adding each variable one at a time. As expected “Stock” has the major impact in decreasing residual deviance (and it has the lowest p-value)

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: df.log.train$d_l_Price
##
## Terms added sequentially (first to last)
##
##
##                Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                           999     1380.8
## d_l_Export      1   0.7727       998     1380.0 0.3793814
## d_l_Import      1   3.4038       997     1376.6 0.0650462 .
## d_l_Refinery    1   0.0479       996     1376.6 0.8267820
## d_l_Prod_Suppl  1   0.3272       995     1376.3 0.5673192
## d_l_Stock       1  10.8624       994     1365.4 0.0009814 ***
## d_l_Prod        1   2.0727       993     1363.3 0.1499557
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

McFadden Index

While no exact equivalent to the R2 of linear regression exists, the McFadden R2 index can be used to assess the model fit.

```
## fitting null model for pseudo-r2
```

```
##           llh       llhNull            G2      McFadden          r2M
## -681.66334737 -690.40667619   17.48665764    0.01266403    0.017334
##          r2CU
##    0.02315529
```

In our case the model didn't fit quite well the data as we can notice by the low index value(below 0.2)

Prediction

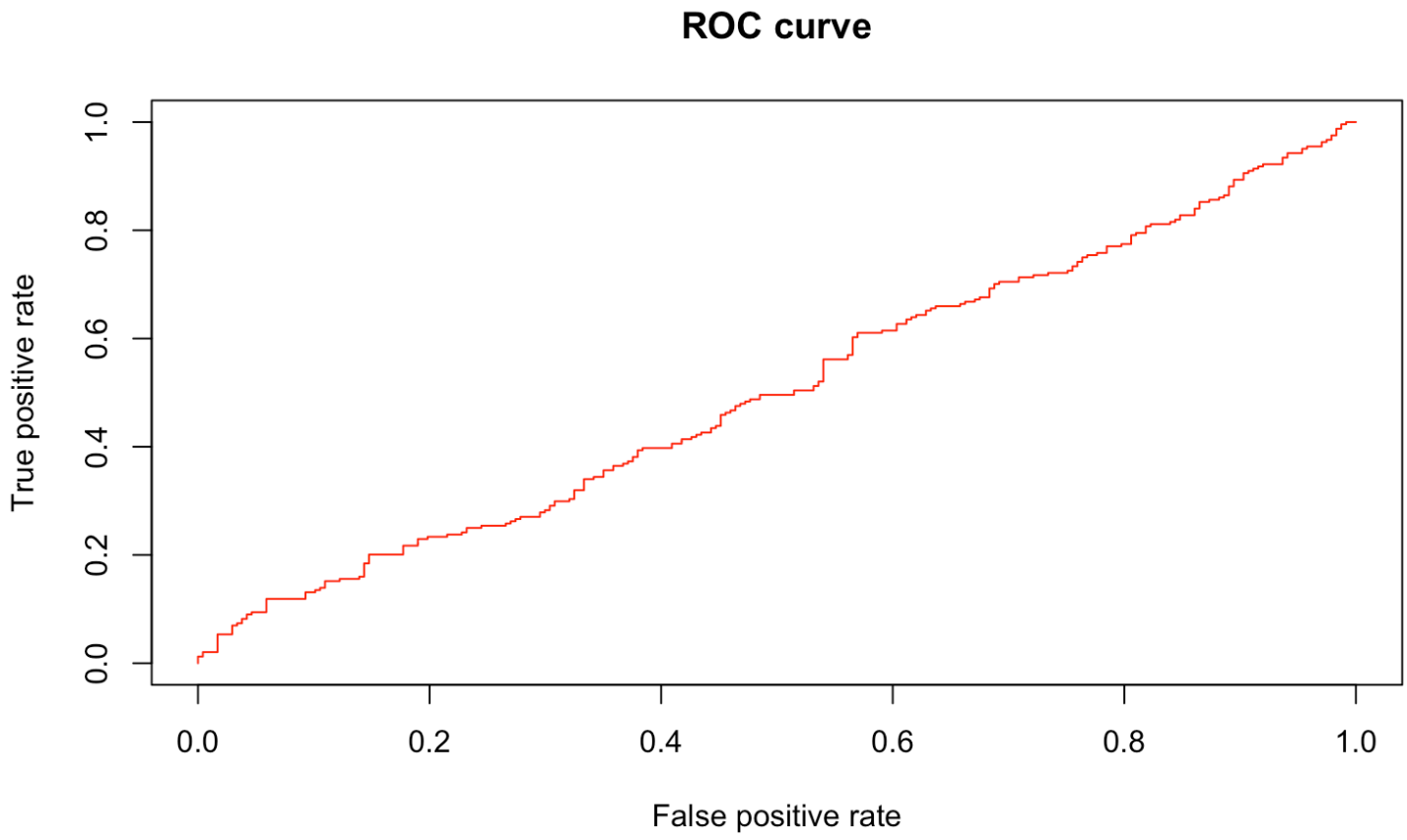
Finally we fit the model on the test data; classification rule that we will adopt consists in assigning all predictions with value below 0.5 to class "0" and all the predictions with value above 0.5 to class "1". Now we can calculate the misclassification error in order to see the accuracy.

```
## [1] "Accuracy 0.51"
```

Unfortunately, model performance is not satisfying for a possible financial application; having 51% chance of righ prediction means that you don't have a real statistical advantage in order to take investment decision.However, keep in mind that this result is somewhat dependent on the manual split of the data that I made earlier, therefore if you wish for a more precise score, you would be better off running some kind of cross validation such as k-fold cross validation.

ROC Curve

As a last step, we are going to plot the ROC curve and calculate the AUC (area under the curve) which are typical performance measurements for a binary classifier. The ROC is a curve generated by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings while the AUC is the area under the ROC curve. As a rule of thumb, a model with good predictive ability should have an AUC closer to 1 (1 is ideal) than to 0.5.



AUC

```
## [1] 0.5043405
```

Final considerations

After all of these analysis, we have established that linear regression and logistic regression don't fit this kind of data too well; the problem could't be in the selection model but in choise of the independent variables. There are many factors that influences price return on financial markets and it's hard to create a model that can fit all of these informations. Another approach that could be adopted is to use Lasso (or Ridge) regression on original data without transforming theme due to the fact that they can solve the problem of multicollinearity. In future researches it will be interesting to apply other regression and classification models like KNN, Decision Trees and Naive Bayes.

That being said, for a good model accuracy that most important thing it's data quality so a good option is to look for new kind of financial data potentially correlated with crude oil price return. As a final thought, i think that our results, although they aren't satisfactory, are still a good starting point for more deep statistical analysis; financial data requires strong knowledge of macroeconomics so having a strong understanding in economics and computational mathematics could help a lot for better results.