Econometrics R Manual

Roman Sigalov

8 August 2016

Contents

Introduction

R reminds much more of a traditional programming language when compared to STATA. Hence its capabilities are much bigger and econometrics is not the only subject where you can use it. It is widely used among researches including economists, in data science, machine learning and other fields. One of the best things about R is that it is FREE and OPEN-SOURCE which mean that if you prefer R to STATA you get (1) great variety of packages capable of doing almost anything and (2) enormous community (on such platforms as stackexchange) which is able to give you an answer on virtually every question. So, if decide to use R during this semester, you are more than welcome to ask help from me, but I strongly encourage you to google your question first, after all, this is the way that even experienced programmist work, they google.

However, there are drawbacks in using R. The most important (and some may say that the only one) is that its learning curve is very steep. It is hard to start programming in R, understand how the function work and how to deal with new problems. The second thing is that its function are limited out-of-the-box and you need to download additional packages if you want to have more functionality.

Basics

Here we will cover the main features of R that are not related to econometrics. We will cover basic data types, how to do basic operations and some trickier stuff that you will need in the future.

Scalars and Vectors

Let's start from the very beginning. You can assign value to variables using `<-`, `->` or `=`:

```
a <- 3
4 -> b # try not to do it at all
c = 5
a
```

```
## [1] 3
```

```
b
```

```
## [1] 4
```

```
c
```

```
## [1] 5
```

R is designed to work with vectors and operatins in vector/ matrix forms are done very quickly. So try to use them instead of loop whenever possible. Creating vector and doing some operations on it:

```
x <- c(1,2,3,4,5)
x[3] # first element has index 1, not 0
```

```
## [1] 3
```

```r
mean(x) # mean of vector values
```

```
## [1] 3
```

```r
sd(x) # standard deviation of vector values
```

```
## [1] 1.581139
```

```r
median(x) # median value of vector values
```

```
## [1] 3
```

By default when you are doing some operations on vector it is done element by element:

```r
y <- c(6,7,8,9,10)
x + y
```

```
## [1]  7  9 11 13 15
```

```r
x * y
```

```
## [1]  6 14 24 36 50
```

```r
5 * y
```

```
## [1] 30 35 40 45 50
```

If you want to multiply vectors in matrix form you need to specify it using **%*%**. For transposition use function `t()`:

```r
x %*% y
```

```
##      [,1]
## [1,]  130
```

```r
x %*% t(y)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    6    7    8    9   10
## [2,]   12   14   16   18   20
## [3,]   18   21   24   27   30
## [4,]   24   28   32   36   40
## [5,]   30   35   40   45   50
```

Matrices

In order to create a matrix use command `matrix()` specifying vector and number of columns/rows:

```r
mat <- matrix(c(1,2,7,4,5,10,7,11,9), nrow = 3)
mat1 <- matrix(c(1,2,3,4,5,6), ncol = 2)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5   11
## [3,]    7   10    9
```

```r
mat1
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
```

```
## [3,]    3    6
```

Pay attention to the fact that values are filled row-by-row to the matrix. Operations in scalar form work element-by-element as it was with vectors. Of course, we can multiply matrices, just pay attention to their dimesions or you will get an error:

```
mat %*% mat1
```

```
##      [,1] [,2]
## [1,]   30   66
## [2,]   45   99
## [3,]   54  132
```

```
t(mat1) %*% mat
```

```
##      [,1] [,2] [,3]
## [1,]   26   44   56
## [2,]   56  101  137
```

You can calculate determinant using `det()`:

```
det(mat)
```

```
## [1] 66
```

To practice let's calculate regression coefficients by hand without using built-in function. We are going to estimate the following regression:

$$log\_cash_i = \beta_0 + \beta_1 market\_to\_book_i + \beta_2 capex_i + \varepsilon_i$$

```
# First upload the data, convert it to data.table format
# (I will describe this incredibly useful library in more details later)
data <- read.csv("/Users/rsigalov/Documents/HSENES/R\ Econometrics\ Handbook/CRSP_data.csv", sep = ";")
library("data.table")
data <- data.table(data)

# Leave only columns that we need and manually exclude NAs
data <- data[, .(log_cash, market_to_book, capex)]
data <- data[!(is.na(log_cash) | is.na(market_to_book) | is.na(capex))]
head(data)
```

```
##      log_cash market_to_book      capex
## 1: -3.2845335       1.132868 0.03056106
## 2: -1.7862760       1.429985 0.12704740
## 3: -0.6498383       1.109031 0.01081693
## 4: -5.9390597       1.171149 0.07268614
## 5: -4.2561327       1.097371 0.02277542
## 6: -1.7464490       2.089204 0.02691105
```

```
# Next we create matrices X and Y which will contain variables of interest and
# dependent variable, respectively
# !!!! It is important to add a columns of ones to X matrix in order to
# estimate constant coefficient beta_0
X <- as.matrix(data[, .(intercept = 1, market_to_book, capex)])
Y <- as.matrix(data[, .(log_cash)])
head(X)
```

```
##      intercept market_to_book      capex
## [1,]         1       1.132868 0.03056106
```

```
## [2,]         1       1.429985 0.12704740
## [3,]         1       1.109031 0.01081693
## [4,]         1       1.171149 0.07268614
## [5,]         1       1.097371 0.02277542
## [6,]         1       2.089204 0.02691105
```

```
head(Y)
```

```
##        log_cash
## [1,] -3.2845335
## [2,] -1.7862760
## [3,] -0.6498383
## [4,] -5.9390597
## [5,] -4.2561327
## [6,] -1.7464490
```

As you may recall, we use the following formula to estimate betas:

$$\hat{\beta} = (X'X)^{-1}X'Y$$

Let's calculate $\hat{\beta}$ and compare the results with a built-in function for regression:

```
beta <- solve(t(X) %*% X) %*% t(X) %*% Y
beta
```

```
##                 log_cash
## intercept      -3.246191
## market_to_book  0.596517
## capex          -6.455668
```

```
model1 <- lm(data = data, log_cash ~ market_to_book + capex)
model1
```

```
##
## Call:
## lm(formula = log_cash ~ market_to_book + capex, data = data)
##
## Coefficients:
##    (Intercept)  market_to_book           capex
##        -3.2462          0.5965         -6.4557
```

As expected the results are exactly the same. Next, we can calculate standard standard errors. In order to do this we first going to estimate variance-covariance matrix, then extract diagonal and take a square root of it. Assuming homoskedasticity of the data (we will remove this assumption in a moment) the consistent estimate of variance-covariance matrix is given by:

$$var - cov = \sigma^2(X'X)^{-1}, \text{ where } \sigma^2 = \frac{\hat{\varepsilon}'\hat{\varepsilon}}{N-K} \text{ and } \hat{\varepsilon} = Y - X\hat{\beta}$$

```
residuals <- Y - X %*% beta
sigma = sqrt(as.numeric((t(residuals) %*% residuals/(dim(X)[1] - dim(X)[2]) )))
vcov_manual <- sigma^2 * solve(t(X) %*% X)
se_manual <- sqrt(diag(vcov_manual))
se_manual
```

```
##      intercept market_to_book          capex
##     0.05256035     0.01978367     0.55181045
```

```
# Now compare it with se produced by built-in variance-covariance matrix function:
vcov_built <- vcov(model1)
se_built <- sqrt(diag(vcov_built))
se_built
```

```
##    (Intercept) market_to_book          capex
##     0.05256035     0.01978367     0.55181045
```

Both results match! Now let's relax the assumption for homoskedasticity and calculate White (or robust) standard errors. The formula for the heteroskedasticity consistent variance-covariance matrix is the following:

$$var - cov = (X'X)^{-1} diag(\hat{\varepsilon}_1^2, \ldots, \hat{\varepsilon}_N^2)(X'X)^{-1}$$

where $diag(\hat{\varepsilon}_1^2, \ldots, \hat{\varepsilon}_N^2)$ denotes a diagonal matrix produced with elements $\hat{\varepsilon}_1^2, \ldots, \hat{\varepsilon}_N^2$ on the main diagonal.

```
vcov_white <- solve(t(X) %*% X) %*% (t(X) %*% diag(diag(residuals %*% t(residuals))) %*% X) %*% solve(t
```

where I use first `diag()` to extract vector of diagonal elements and second `diag()` to produce a diagonal matrix out of these elements. Let's compare the results with built-in function for HC var-cov matrix (using package `sandwich`):

```
se_white_manual <- sqrt(diag(vcov_white))
library("sandwich")
se_white_built <- sqrt(diag(vcovHC(model1)))
se_white_manual
```

```
##      intercept market_to_book          capex
##     0.05165331     0.02063012     0.63433894
```

```
se_white_built
```

```
##    (Intercept) market_to_book          capex
##     0.05175722     0.02071015     0.63660199
```

The results are a little bit different since the estimator requires some normalization which is negligible for large N.

Data.

Here I will explain how to get data into R. Both from local computer in different formats as well as from the web.

Loops, conditional statements and other.

Packages

Here I will highlight several packages that are very useful for data analysis and for your econometrics class as well. Some of them may do things that will be required later in the semester. So, if you do not understand yet what is Newey-West errors don't bother, you will catch up later.

Almost every package that you will need are stored in so called The Comprehensive R Archive Network (CRAN). It means that you can simply install them by typing

```
install.packages("ggplot2")
```

in R-console or RStudio. Lets list the packages that are either important for the course or give some interesting new function.

```
library("sandwich")
```

is able to compute various kinds of standard errors including, White (Heteroskedasticity consistent errors, hence HC) and Newey-West (Heteroskedasticity and Autocorrelation Consistent errors, hence HAC).

```
library("ggplot2")
library("ggthemes")
```

first is able to plot almost anything you will every need to plot. Second adds additional themes to ggplot: you can make you graph look like The Economist Graph, WSJ or even STATA graph.

```
library("sqldf")
```

allows you to manipulate data using SQLite language. If you you know SQL well it will probably become your favorite package. Otherwise, I will either show you how to do some stuff in it, or you may not bother much

```
library("reshape")
```

allows you to do reverse pivot and other interesting things. **show reverse pivot example**

```
library("DataCombine")
```

allows you to easily create lags for you regression

```
library("xts")
```

additional capabilities for time series data manipulation, will be covered later. Below you can find other interesting packages.

```
library("stargazer")
```

```
##
## Please cite as:
##  Hlavac, Marek (2015). stargazer: Well-Formatted Regression and Summary Statistics Tables.
##  R package version 5.2. http://CRAN.R-project.org/package=stargazer
```

allows you to export regression results into LaTeX format.

```
library("TTR") # for rolling volatility
library("lubridate") # for operations on dates
library("ghyp")
library("MASS")
library("xts")
library("memisc")
library("grid") # for arranging plots in grid
library("psych")
library("FinTS") # For GARCH regressions
library("fGarch")
```

Regressions I. Basics

Here we are going to do a simple regression on our demo dataset. I will show how to access coefficients, predict, convert output to LaTeX and plot results.

We are going to use data from CRSP and study companys' cash balances and what affects them. Let's import data and look what columns do we need:

```
setwd("~")
data <- read.csv("/Users/rsigalov/Documents/HSENES/R\ Econometrics\ Handbook/CRSP_data.csv", sep = ";")
colnames(data)
```

```
##  [1] "GVKEY"               "DATADATE"            "SIC"
##  [4] "AT"                  "CHE"                 "DLC"
##  [7] "DLTT"                "SEQ"                 "WCAP"
## [10] "DVC"                 "OIBDP"               "SALE"
## [13] "TXT"                 "XINT"                "XRD"
## [16] "AQC"                 "CAPX"                "CSHO"
## [19] "PRCC_F"              "market_to_book"      "log_firm_size"
## [22] "cf"                  "nwc"                 "capex"
## [25] "leverage"            "div"                 "rd_to_sales"
## [28] "sic_2"               "ind_cf_vol"          "ind_cf_no_level_vol"
## [31] "cash"                "log_cash"            "aqc_to_asset"
## [34] "year"
```

Let's first estimate regression:

$$cash_i = \beta_0 + \beta_1 ind\_cf\_vol_i + \varepsilon_i$$

since volatility of cash flow of the industry (ind_cf_vol) may be positively related to the cash that the firm wants to hold to absorb risks. Use command summary to print the output of the regression:

```
model <- lm(data = data, cash ~ ind_cf_vol)
summary(model)
```

```
##
## Call:
## lm(formula = cash ~ ind_cf_vol, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.43348 -0.12046 -0.04927  0.07161  0.77684
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.03463    0.00576   6.013 1.99e-09 ***
## ind_cf_vol   2.53310    0.08101  31.270  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2045 on 4071 degrees of freedom
##   (535 observations deleted due to missingness)
## Multiple R-squared:  0.1937, Adjusted R-squared:  0.1935
## F-statistic: 977.8 on 1 and 4071 DF,  p-value: < 2.2e-16
```

The object that `summary()` command creates contains a lot of useful information. For example:

```
sum <- summary(model)
sum$coefficients # outputs a matrix
```

```
##               Estimate  Std. Error   t value      Pr(>|t|)
## (Intercept) 0.03463513 0.005760495  6.012526  1.986903e-09
## ind_cf_vol  2.53310493 0.081007696 31.269929 1.441064e-192
```

```
sum$r.squared
```

```
## [1] 0.1936711
```

```
sum$fstatistic
```

```
##     value     numdf     dendf
##  977.8085    1.0000 4071.0000
```

Using coefficients we can manually predict cash balance:

```
beta_0 <- sum$coefficients[1,1]
beta_1 <- sum$coefficients[2,1]
ind_cf_vol_1 <- 0.05
beta_0 + beta_1 * ind_cf_vol_1
```

```
## [1] 0.1612904
```

But, of course, we can predict easier using our model object:

```
predict(model, newdata = (ind_cf_vol = 0.05))
```

```
##         1
## 0.1612904
```

or we can do it for several points (but we need to insert dataframe)

```
predict(model, newdata = data.frame(ind_cf_vol = c(0.05, 0.06, 0.07, 0.08)))
```

```
##         1         2         3         4
## 0.1612904 0.1866214 0.2119525 0.2372835
```

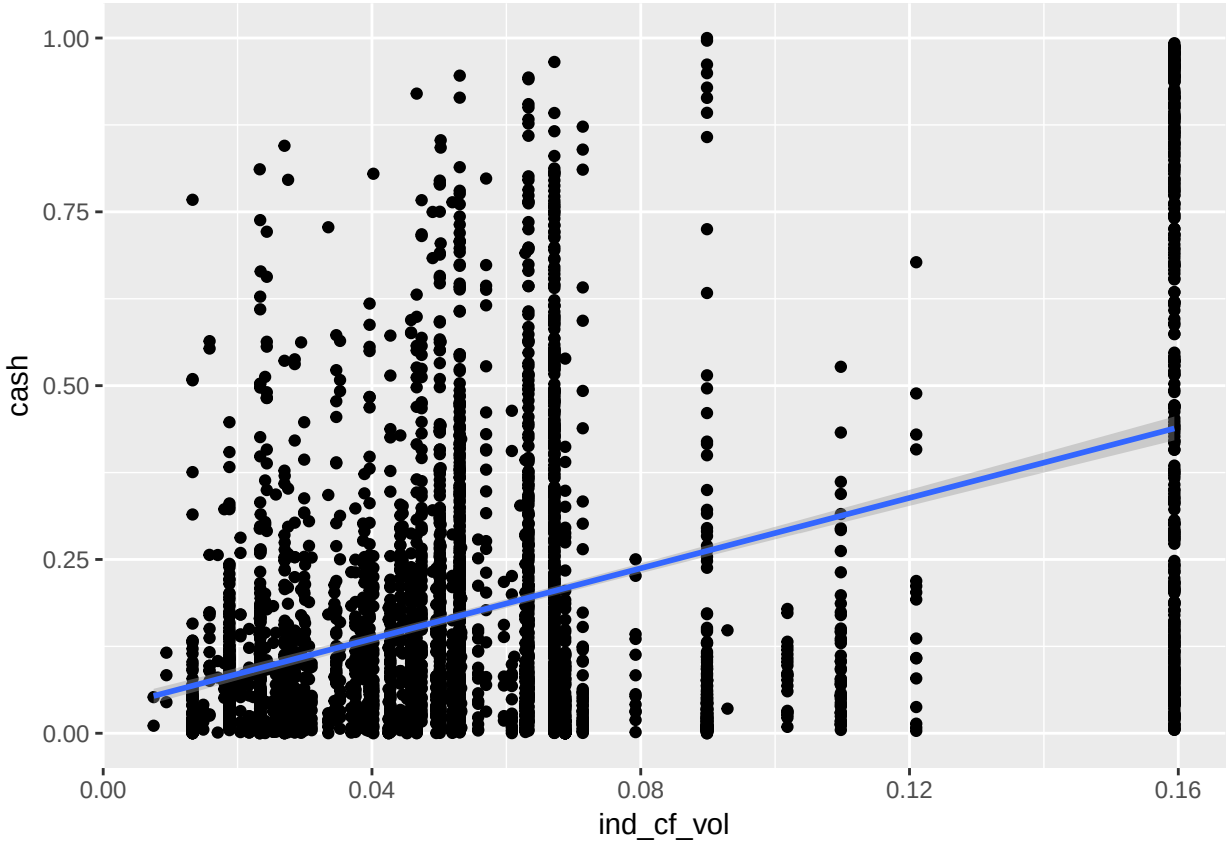
and add confidence interval

```
predict(model, newdata = data.frame(ind_cf_vol = c(0.05, 0.06, 0.07, 0.08)), interval = "confidence")
```

```
##         fit       lwr       upr
## 1 0.1612904 0.1548442 0.1677365
## 2 0.1866214 0.1803375 0.1929053
## 3 0.2119525 0.2054358 0.2184692
## 4 0.2372835 0.2301778 0.2443893
```

Let's plot our results using ggplot (in more details it will be explained in the next part):

```
library("ggplot2")
ggplot(data = data, aes(x = ind_cf_vol, y = cash)) +
  geom_point() +
  geom_smooth(method = "lm", se = TRUE) # fitting a line
```



Fitting multivariate regression and prediction is just as simple:

```
model2 <- lm(data = data, cash ~ ind_cf_vol + rd_to_sales)
predict(model2,
        newdata = data.frame(ind_cf_vol = c(0.05, 0.06),
                             rd_to_sales = c(0.3, 0.45)),
        interval = "confidence")
```

```
##         fit       lwr       upr
## 1 0.1710242 0.1651791 0.1768694
## 2 0.1950629 0.1892911 0.2008347
```

If you want to calculate $RMSE$, you can do it just manually since it is

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} \hat{u}^2}{n}}$$

```
sqrt(mean(model$residuals))
```

```
## Warning in sqrt(mean(model$residuals)): NaNs produced
```

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

Next we are going to export summary for the second to LaTeX format using package `stargazer`:

```
library("stargazer")
stargazer(model2, type = "latex", df=FALSE, dep.var.labels = NULL, report = "vcp")
```

```
##
## % Table created by stargazer v.5.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harvar
## % Date and time: Fri, Jun 30, 2017 - 12:26:08
```

```
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}}lc}
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
##  & \multicolumn{1}{c}{\textit{Dependent variable:}} \\
## \cline{2-2}
## \\[-1.8ex] & cash \\
## \hline \\[-1.8ex]
##  ind\_cf\_vol & 1.310 \\
##   & p = 0.000 \\
##   & \\
##  rd\_to\_sales & 0.073 \\
##   & p = 0.000 \\
##   & \\
##  Constant & 0.084 \\
##   & p = 0.000 \\
##   & \\
## \hline \\[-1.8ex]
## Observations & 4,018 \\
## R$^{2}$ & 0.273 \\
## Adjusted R$^{2}$ & 0.273 \\
## Residual Std. Error & 0.183 \\
## F Statistic & 755.339$^{***}$ \\
## \hline
## \hline \\[-1.8ex]
## \textit{Note:}  & \multicolumn{1}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.01} \\
## \end{tabular}
## \end{table}
```

Now you can copy and paste this code into latex and you will get a nice table. You can include different model into the same output latex-code and specify errors (standard errors will be covered in a great details later) that you want to report:

```
library("sandwich")
se_model <- sqrt(diag(vcovHC(model)))
se_model2 <- sqrt(diag(vcovHC(model2)))

knitr::kable(stargazer(model, model2, type = "html", report = "vcp",
          se = list(se_model, se_model2)))
```

```
##
## <table style="text-align:center"><tr><td colspan="3" style="border-bottom: 1px solid black"></td></t
## <tr><td></td><td colspan="2" style="border-bottom: 1px solid black"></td></tr>
## <tr><td style="text-align:left"></td><td colspan="2">cash</td></tr>
## <tr><td style="text-align:left"></td><td>(1)</td><td>(2)</td></tr>
## <tr><td colspan="3" style="border-bottom: 1px solid black"></td></tr><tr><td style="text-align:left"
## <tr><td style="text-align:left"></td><td>p = 0.000</td><td>p = 0.000</td></tr>
## <tr><td style="text-align:left"></td><td></td><td></td></tr>
## <tr><td style="text-align:left">rd_to_sales</td><td></td><td>0.073</td></tr>
## <tr><td style="text-align:left"></td><td></td><td>p = 0.000</td></tr>
## <tr><td style="text-align:left"></td><td></td><td></td></tr>
## <tr><td style="text-align:left">Constant</td><td>0.035</td><td>0.084</td></tr>
## <tr><td style="text-align:left"></td><td>p = 0.000</td><td>p = 0.000</td></tr>
```

```
## <tr><td style="text-align:left"></td><td></td><td></td></tr>
## <tr><td colspan="3" style="border-bottom: 1px solid black"></td></tr><tr><td style="text-align:left"
## <tr><td style="text-align:left">R<sup>2</sup></td><td>0.194</td><td>0.273</td></tr>
## <tr><td style="text-align:left">Adjusted R<sup>2</sup></td><td>0.193</td><td>0.273</td></tr>
## <tr><td style="text-align:left">Residual Std. Error</td><td>0.205 (df = 4071)</td><td>0.183 (df = 401
## <tr><td style="text-align:left">F Statistic</td><td>977.808<sup>***</sup> (df = 1; 4071)</td><td>755
## <tr><td colspan="3" style="border-bottom: 1px solid black"></td></tr><tr><td style="text-align:left"
## </table>
```

Dependent variable:

cash

(1)

(2)

ind_cf_vol

2.533

1.310

p = 0.000

p = 0.000

rd_to_sales

0.073

p = 0.000

Constant

0.035

0.084

p = 0.000

p = 0.000

Observations

4,073

4,018

R2

0.194

0.273

Adjusted R2

0.193

0.273

Residual Std. Error

0.205 (df = 4071)

0.183 (df = 4015)

F Statistic

977.808*** (df = 1; 4071)

755.339*** (df = 2; 4015)

Note:

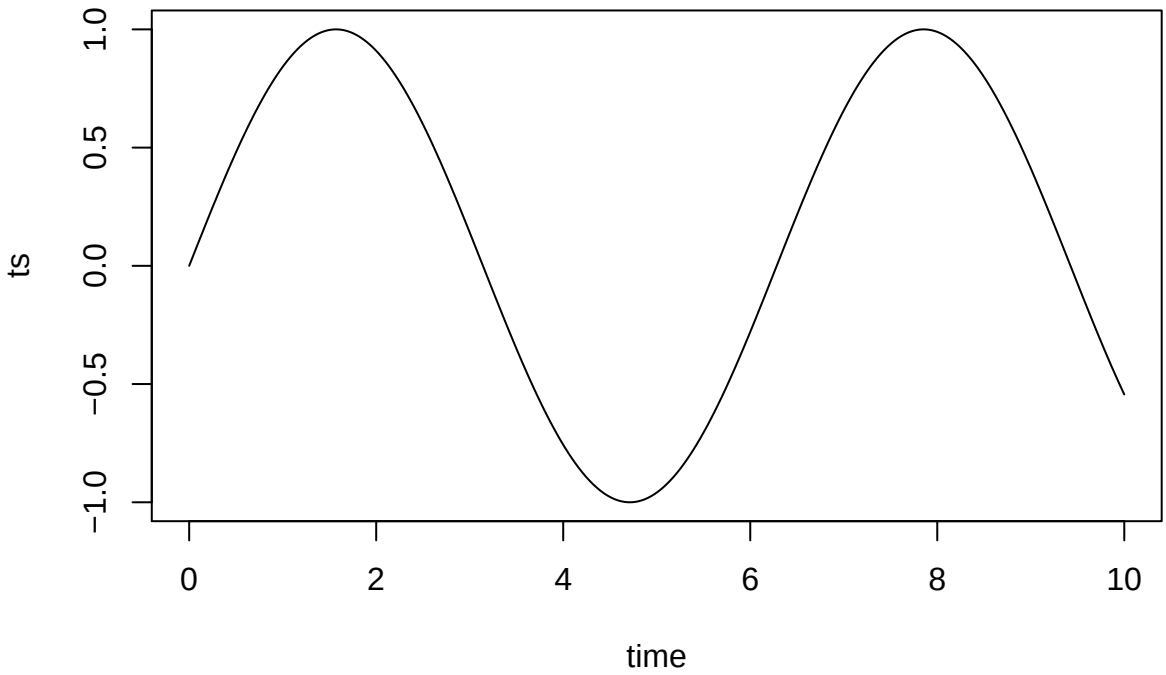
p<0.1; ***p<0.05;*** p<0.01

Here I used `knitr::kable()` to directly put a table into html file. Stargazer has a lot of capabilities, for reference use this link http://jakeruss.com/cheatsheets/stargazer.html

Plotting. Introduction to ggplot2.

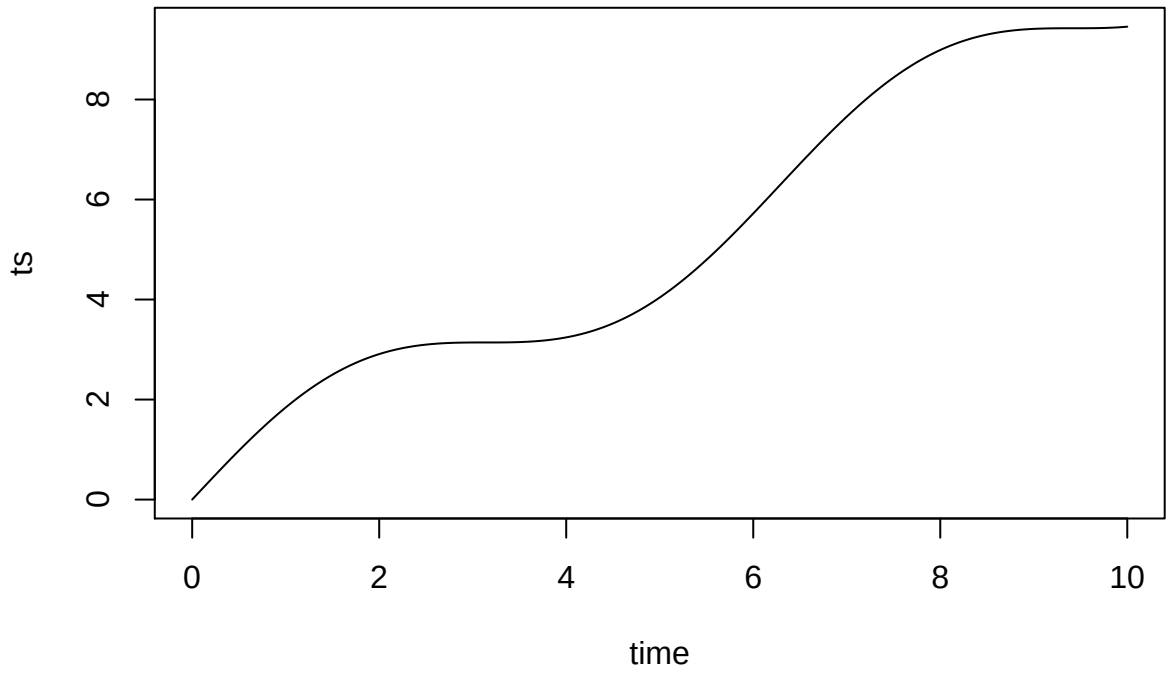
`ggplot2` is a wonderful package which will help you to create beautiful graphs. It can be tricky at first but as your demands will grow it will become indisensible. First let's create simple graph without ggplot, because it is easier to do it this way.

```
time <- seq(0,10,0.01)
ts <- sapply(time, sin)
plot(x = time, y = ts, type = "l")
```



`sapply` is a very powerful function, it takes vector (in our case `time`) and applies function `sin` elementwise. If you will create your own function you can use it as well. For example:

```
time <- seq(0,10,0.01)
func <- function(x) {sin(x) + x}
ts <- sapply(time, func)
plot(x = time, y = ts, type = "l")
```



ggplot is definitely worse in plotting such data. Hence, I will show how to use it for more complicated datasets with a lot of variables that we want to somehow represent.

I have two datasets which represent results of backtesting of a trading strategy. First file contains parameters on the beginning of the backtest, second file on the end.

```
setwd("~")
first <- read.csv("/Users/rsigalov/Documents/HSENES/R\ Econometrics\ Handbook/EURUSD_first.csv", sep = "
last <- read.csv("/Users/rsigalov/Documents/HSENES/R\ Econometrics\ Handbook/EURUSD_last.csv", sep = ";"
colnames(first)
```

```
##  [1] "Day"              "Date"             "Spot"
##  [4] "Vol"              "Rate"             "Hedge"
##  [7] "Rehedge_Value"    "Option_Value"     "Interpolated_Vol"
## [10] "Delta"            "Maturity"         "Forward"
## [13] "Hedge_Position"   "Stock_Position"   "Trading_Gain"
## [16] "Position_Value"   "Acc_Trading_Gain" "Trades"
## [19] "Hedge_name"       "Scenario"         "Scenario_Group"
## [22] "Hedge_Type"       "Day_Type"
```

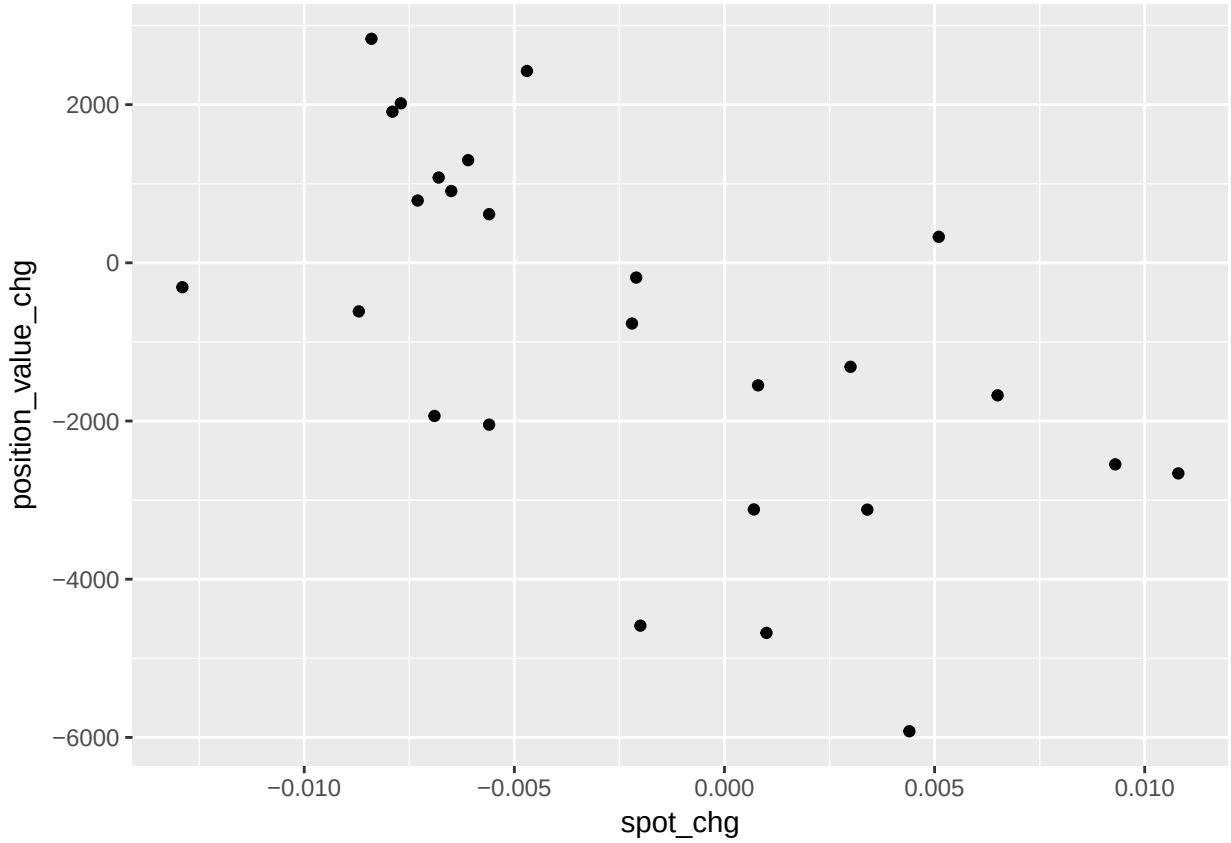
I will do some manipulations with these files and show which columns do we have in the end:

```
stats <- merge(x = first, y = last, by = c("Scenario", "Scenario_Group"))
stats$scenario_group <- stats$Scenario_Group
stats$scenario <- stats$Scenario
stats$spot_chg <- stats$Spot.y - stats$Spot.x
stats$position_value_chg <- stats$Position_Value.y + stats$Acc_Trading_Gain.y - stats$Option_Value.x
stats$trades <- stats$Trades
stats$option_chg <- stats$Option_Value.y - stats$Option_Value.x

stats <- stats[names(stats) %in% c("scenario_group", "scenario", "spot_chg", "position_value_chg", "trad

colnames(stats)
```

```
## [1] "scenario_group"     "scenario"           "spot_chg"
## [4] "position_value_chg" "option_chg"
```

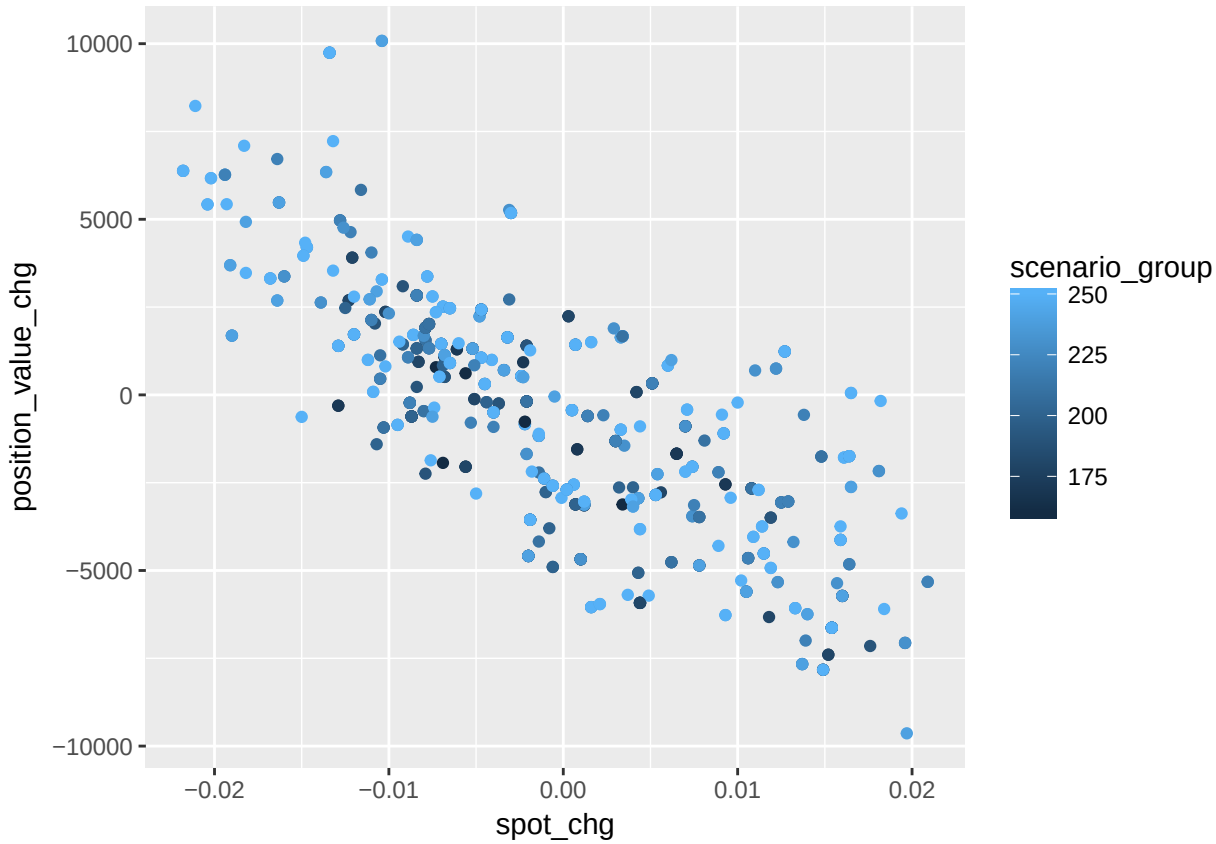
We have different scenario groups and different scenarios. First, let's plot change in position value against change in spot price of asset:

```
library("ggplot2")
ggplot(stats[stats$scenario_group == 160,], aes(x = spot_chg, y = position_value_chg)) +
  geom_point()
```



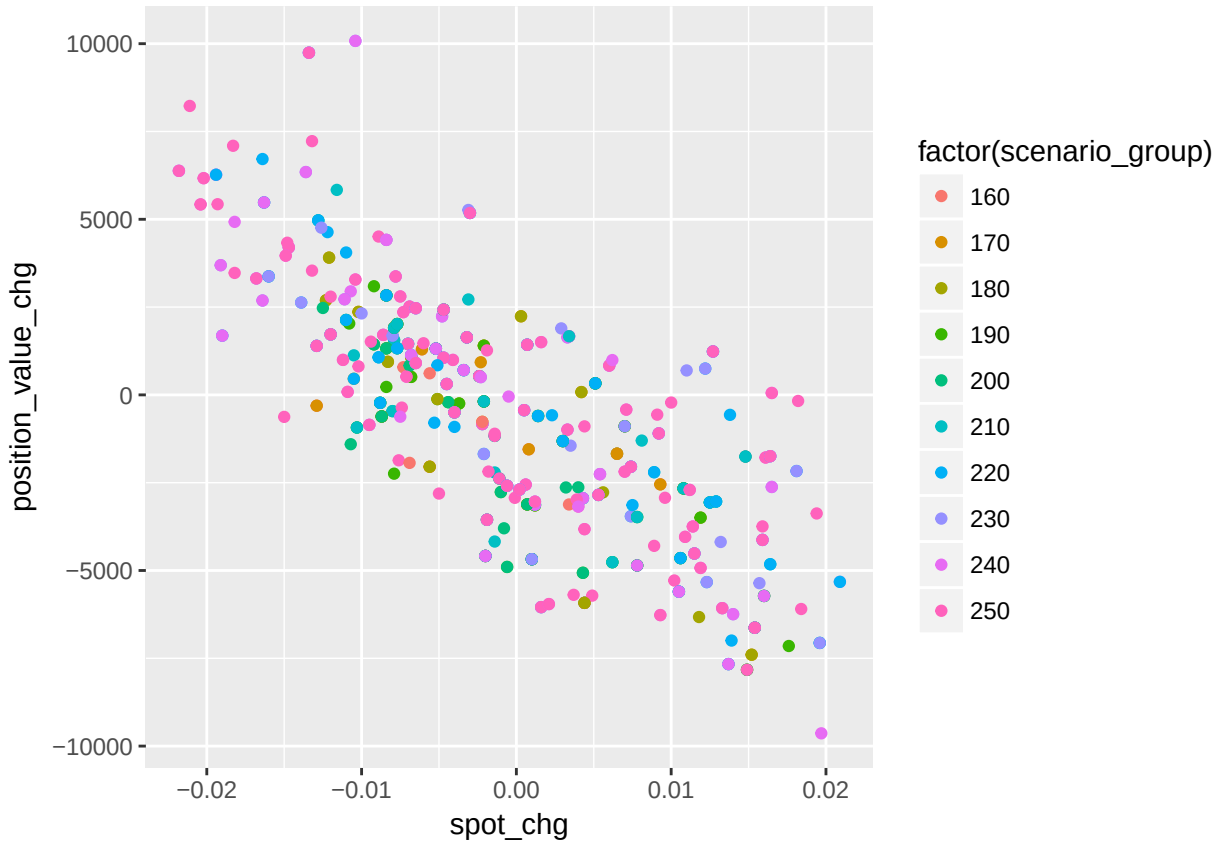
However, we might want to show different groups we can do it by adding color into aes with which will depend on group:

```
ggplot(stats, aes(x = spot_chg, y = position_value_chg, color = scenario_group)) +
  geom_point()
```



ggplot suggested to use continuous scale for our groups, however, they are discrete and we can specify it by adding `factor()` around `scenario_group`:

```
ggplot(stats, aes(x = spot_chg, y = position_value_chg, color = factor(scenario_group))) +
  geom_point()
```

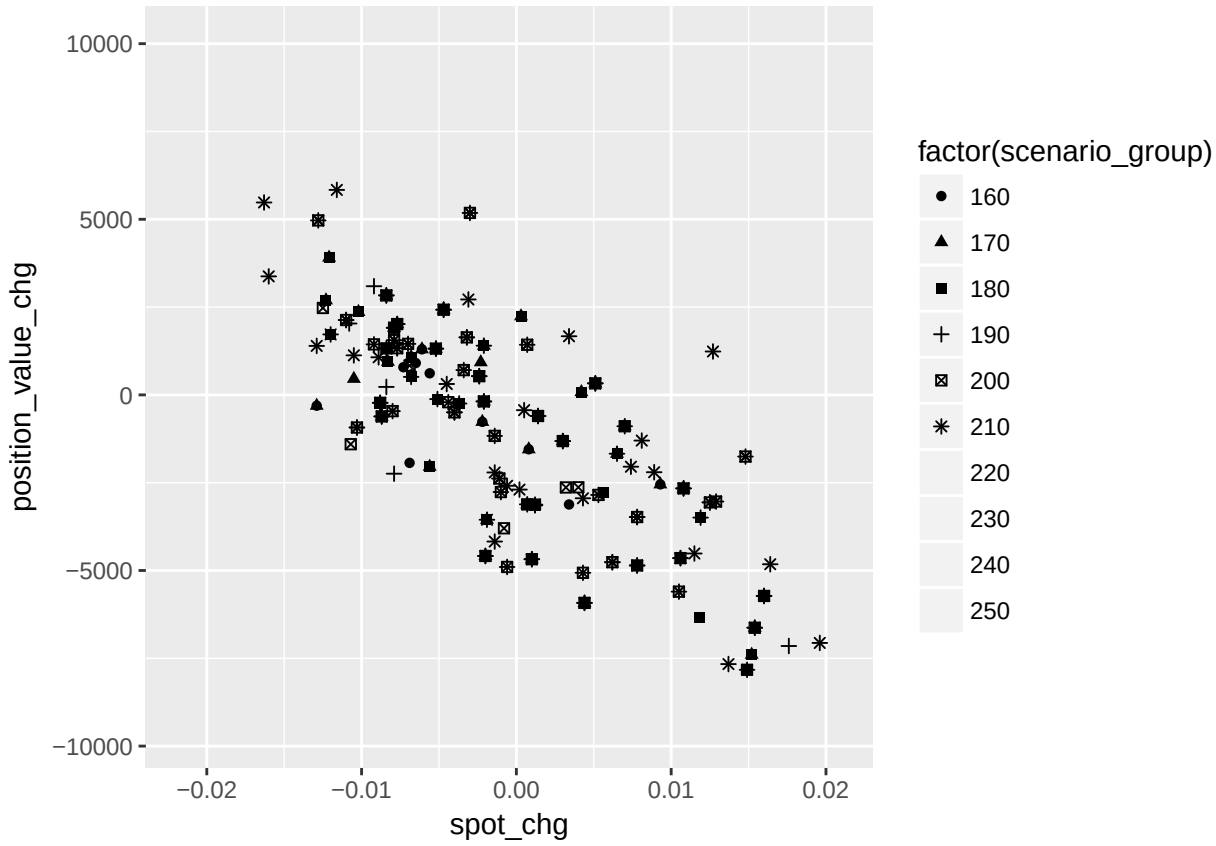


It looks like a mess, maybe it is better give each group a separate point type:

```
ggplot(stats, aes(x = spot_chg, y = position_value_chg, shape = factor(scenario_group))) +
  geom_point()
```

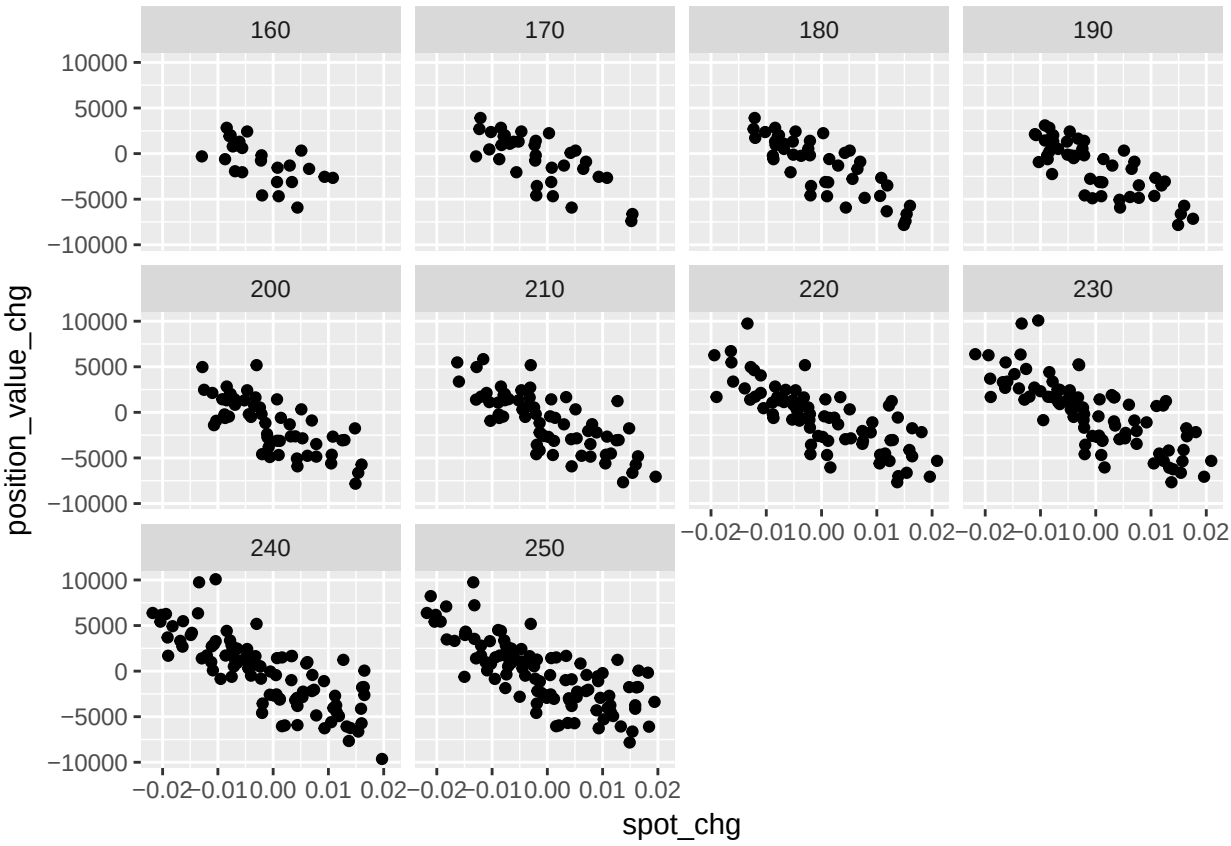
```
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have
## 10. Consider specifying shapes manually if you must have them.
```

```
## Warning: Removed 363 rows containing missing values (geom_point).
```



Well, there are no such many point types. We can separate each group into seprate graph:

```
stats$scenario_group <- as.character(stats$scenario_group)
ggplot(stats, aes(x = spot_chg, y = position_value_chg)) +
  geom_point() +
  facet_wrap(~scenario_group)
```



Above I converted scenario_group column in character format from numerical, to avoid using `factor()`, for some reasons it doesn't work in facet. Of course, we can do not only points, but also lines. Below, I will show several examples. I have the following data:

```
price_data <- read.csv("/Users/rsigalov/Documents/HSENES/R\ Econometrics\ Handbook/price_dynamics.csv",
colnames(price_data)
```

```
##  [1] "X"           "sigma_s"     "sigma_e"     "theta"       "k_d"
##  [6] "lambda"      "lambda_real" "v_bar"       "time"        "price"
## [11] "price_cond"  "k_tilde"
```

```
unique(price_data$sigma_e)
```

```
## [1] 0.50 0.75 1.00 1.25 1.50
```

```
unique(price_data$k_d)
```

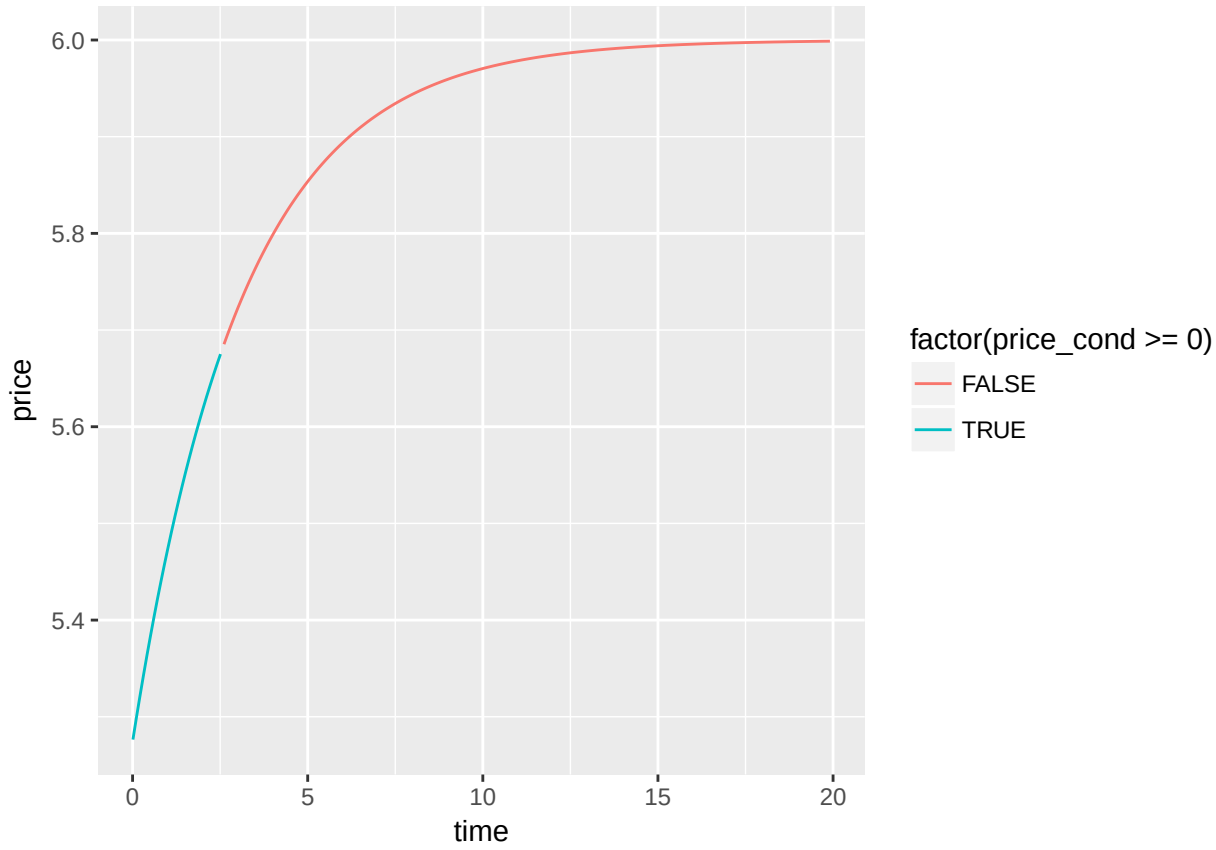
```
## [1] 0.50 0.75 1.00 1.25 1.50
```

```
unique(price_data$lambda)
```

```
## [1] 0.50 0.75 1.00 1.25 1.50
```

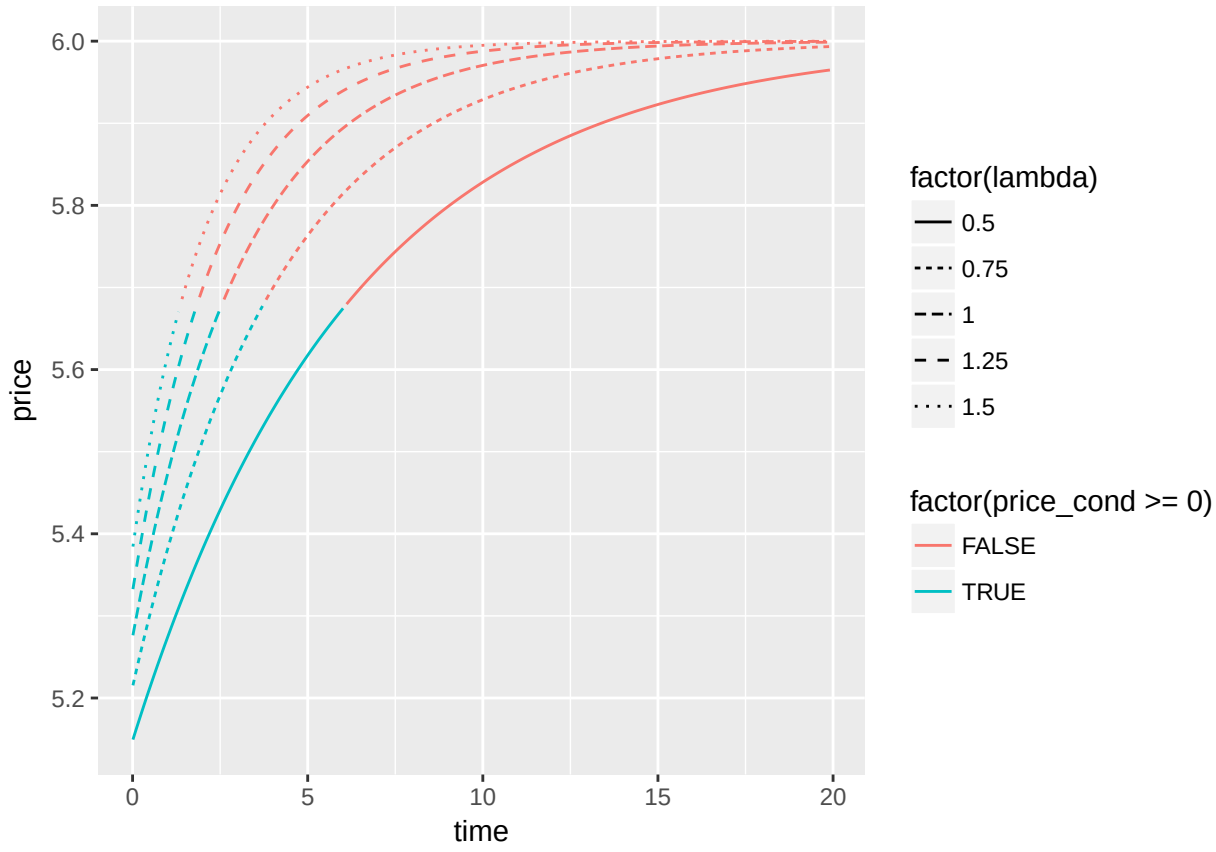
Let's first pick some values for `sigma_e`, `k_d` and `lambda`. I want to plot `price` against `time` and want to distinguish whether `price_cond` was above or below zero.

```
price_data_sub <- price_data[price_data$sigma_e==1 & price_data$k_d==1 & price_data$lambda==1,]
ggplot(price_data_sub, aes(x = time, y = price, color = factor(price_cond >= 0))) +
  geom_line()
```



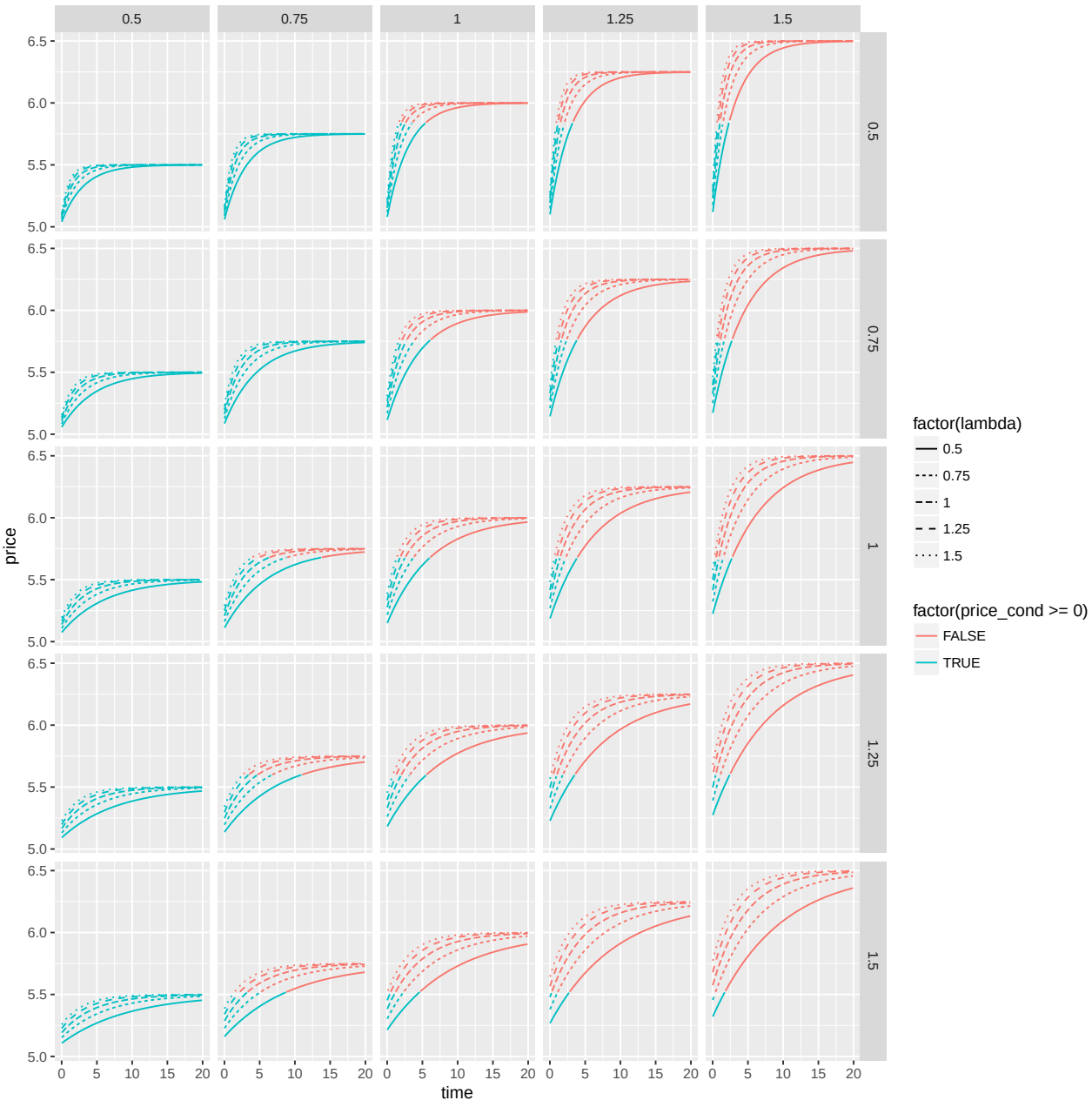
Now I want to add different lines types for values of `lambda` (here we need to use `factor()` again):

```
price_data_sub <- price_data[price_data$sigma_e==1 & price_data$k_d==1,]
ggplot(price_data_sub, aes(x = time, y = price, color = factor(price_cond >= 0), linetype = factor(lambd
  geom_line()
```



and then we can make a grid with horizontal separation by `k_d` and vertical by `sigma_e`:

```
ggplot(price_data, aes(x = time, y = price, color = factor(price_cond >= 0), linetype = factor(lambda))
  geom_line() +
  facet_grid(sigma_e ~ k_d)
```



Using combinations of facetting, linetype, linecolors you can show various proprties of a data on one single plot. This is just basics of `ggplot` I will include chapter on additional features later on. Meanwhile you can use this cheatsheet https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf which is really helpful.

Regressions II. Standard Errors

Here I will show how to change standard errors as well as how to manipulate variance-covariance matrices and other important stuff. I will give you several examples of how to build confidence intervals and make graphs clearly representing your point.

Let's continue to work with our previous example and subset data to include only one year. Use `unique()` function to select only inique values from a vector:

```r
unique(data$year) # Determining which years there are
```

```
## [1] 2015 2016
```

```r
data2016 <- subset(x = data, year == 2016)
```

Since we might expect that each error are different for each firm we might want to use White standard errors. Using package `sandwich` we can estimate Heteroskedasticity consistent variance-covariance matrix:

```r
library("sandwich")
model <- lm(data = data, cash ~ ind_cf_vol + rd_to_sales)
vcovHC(model) # Variance convariance matrix
```

```
##               (Intercept)    ind_cf_vol   rd_to_sales
## (Intercept)  2.608018e-05 -0.0004258878  4.315199e-06
## ind_cf_vol  -4.258878e-04  0.0095383263 -1.198015e-04
## rd_to_sales  4.315199e-06 -0.0001198015  1.586688e-05
```

```r
diag(vcovHC(model)) # Diagonal elements of VCOV
```

```
##  (Intercept)   ind_cf_vol  rd_to_sales
## 2.608018e-05 9.538326e-03 1.586688e-05
```

```r
sqrt(diag(vcovHC(model))) # standard errors
```

```
## (Intercept)  ind_cf_vol rd_to_sales
## 0.005106876 0.097664355 0.003983326
```

We can then test the coefficients using new errors using package `lmtest` and save these errors for subsequent use, for example, to export to LaTeX:

```r
library("lmtest")
coeftest(model, vcov = vcovHC(model))
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 0.0836461  0.0051069  16.379 < 2.2e-16 ***
## ind_cf_vol  1.3100239  0.0976644  13.414 < 2.2e-16 ***
## rd_to_sales 0.0729232  0.0039833  18.307 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
model_se <- sqrt(diag(vcovHC(model)))
```

Now we can create a LaTeX output with new standard errors **make Latex output table**.

Later you will need to use Heteroskedasticity and Autocorrelation consistent standard error (or Newey-West standard errors) which can be evaluated using the same way and function `vcovHAC()`.

Regressions III. Transformatin of variables

Here I will show you how to transform your variables to perform log-linear, linear-log, log-log, power and other regressions. First, you can always add another variable to the dataset and oridary regression:

```r
data$log_rd_to_sales <- log(data$rd_to_sales)
head(data$log_rd_to_sales)
```

```
## [1]      -Inf      -Inf      -Inf      -Inf -4.729433      -Inf
```

There are a lot of zeros, thus we need to exclude such observations in order to perform log regression:

```
data_tmp <- data[data$rd_to_sales != 0, ]
data_tmp$log_rd_to_sales <- log(data_tmp$rd_to_sales)
model <- lm(data = data_tmp, formula = cash ~ ind_cf_vol + log_rd_to_sales)
```

or you can simply specify it in the `lm()` function to create different kind of regressions:

```
model <- lm(data = data_tmp, formula = cash ~ ind_cf_vol + log(rd_to_sales)) # linear-log
model <- lm(data = data_tmp, formula = log(cash) ~ ind_cf_vol + rd_to_sales) # log-linear
model <- lm(data = data_tmp, formula = log(cash) ~ ind_cf_vol + log(rd_to_sales)) # log-log
```

In order to perform, for example, quadratic regression, we may again create additional variable:

```
data$ind_cf_vol_sqr <- (data$ind_cf_vol)^2
model <- lm(data = data, formula = cash ~ ind_cf_vol + ind_cf_vol_sqr)
```

or make the transformation inside `lm()` function using `I()`. You can add different transformation of the same variable (for example for fitting $y = \beta_0 + \beta_1 x + \beta_2 x^2$ using `I()` as well:

```
model <- lm(data = data, formula = cash ~ ind_cf_vol + I(ind_cf_vol^2))
model <- lm(data = data, formula = cash ~ ind_cf_vol + I(ind_cf_vol^2) + I(ind_cf_vol^3))
```

Tables

Here I will show you how to create LaTeX tables for your descriptive statistics and other

Regressions IV. F-statistics

In this section I will show you how to compute F-statistics for the regression and in general how to test linear hypotheses. This can be done using package `car`. Let's get back to our example with CRSP data and issue of companys' cash holdings with several explanatory variables:

```
setwd("~")
data <- read.csv("/Users/rsigalov/Documents/HSENES/R\ Econometrics\ Handbook/CRSP_data.csv", sep = ";")
model <- lm(data = data, formula = cash ~ market_to_book + ind_cf_vol + rd_to_sales)
summary(model)
```

```
##
## Call:
## lm(formula = cash ~ market_to_book + ind_cf_vol + rd_to_sales,
##     data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.77223 -0.09865 -0.04465  0.05482  0.85233
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    0.004940   0.005575   0.886    0.376
## market_to_book 0.056914   0.001953  29.144   <2e-16 ***
## ind_cf_vol     0.820816   0.076540  10.724   <2e-16 ***
## rd_to_sales    0.062436   0.002534  24.635   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1656 on 3997 degrees of freedom
##   (607 observations deleted due to missingness)
## Multiple R-squared:  0.3999, Adjusted R-squared:  0.3994
## F-statistic: 887.8 on 3 and 3997 DF,  p-value: < 2.2e-16
```

Well summary already provides you with F-statistics, however, using standard standard errors, and we might want to use HC errors. Import `car` library and test simple F-statistics:

```
library("car")
# exactly what is shown in the summary of the regression above
linearHypothesis(model, c("market_to_book = 0", "ind_cf_vol = 0", "rd_to_sales = 0"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## market_to_book = 0
## ind_cf_vol = 0
## rd_to_sales = 0
##
## Model 1: restricted model
## Model 2: cash ~ market_to_book + ind_cf_vol + rd_to_sales
##
##   Res.Df    RSS Df Sum of Sq     F    Pr(>F)
## 1   4000 182.65
## 2   3997 109.61  3    73.038 887.8 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# Some abstract hypothesis
linearHypothesis(model, c("2 * market_to_book - 5*ind_cf_vol = 0"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## 2 market_to_book - 5 ind_cf_vol = 0
##
## Model 1: restricted model
## Model 2: cash ~ market_to_book + ind_cf_vol + rd_to_sales
##
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1   3998 112.58
## 2   3997 109.61  1    2.9674 108.21 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If you want to include intercept into linear hypothesis use `(Intercept)` variable:

```
linearHypothesis(model, c("2 * (Intercept) - 5*ind_cf_vol = 0"))
```

```
## Linear hypothesis test
##
## Hypothesis:
## 2 ((Intercept) - 5 ind_cf_vol = 0
##
## Model 1: restricted model
```

```
## Model 2: cash ~ market_to_book + ind_cf_vol + rd_to_sales
##
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1   3998 112.64
## 2   3997 109.61  1    3.0283 110.43 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Of course, you may want to use non-standard standard errors. In order to do it you can supply variance-covariance matrix into hypothesis test. In the example below I supply heteroskedasticity-consistent standard errors:

```
linearHypothesis(model, c("market_to_book = 0", "ind_cf_vol = 0", "rd_to_sales = 0"),
                 vcov = vcovHC(model))
```

```
## Linear hypothesis test
##
## Hypothesis:
## market_to_book = 0
## ind_cf_vol = 0
## rd_to_sales = 0
##
## Model 1: restricted model
## Model 2: cash ~ market_to_book + ind_cf_vol + rd_to_sales
##
## Note: Coefficient covariance matrix supplied.
##
##   Res.Df Df      F    Pr(>F)
## 1   4000
## 2   3997  3 406.27 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Regressions V. Probit and Logit

Have you heard of neural networks? If you did and want to understand how they are working you first need to get Probit and Logit.

Plotting. More advanced ggplot2.

Dates and time

It may be combersome to deal with dates and especially times. Here I will show how we can easily work with dates and times using R. First we will deal with importing them properly. Suppose that we have date in Russian format `05/06/2017` which is 5th of June, 2017. In order to import this string into format compatible with R we need to do the following:

```
date1 <- "05/06/2017"
date_formatted <- as.Date(date1, format = "%d/%m/%Y")
date_formatted
```

```
## [1] "2017-06-05"
```

where `%d` means day in double digit format, `%m` month in two-digit format and `%Y` year in FOUR-digit format. Small y (i.e. `%y`) is used for two-digit year formatting. R treats date fromat as number of days since January

1st, 1970 and after some manipulations it can convert it to this number, for example:

```
as.numeric(date_formatted)
```

```
## [1] 17322
```

The next thing is to deal with time which is a littled bit more tricky. Time in R follow the standard `POSIXct` and has this type. Suppose that you have a really trashy respresentation of time. Date is in format 28/06/2017, time is in format 101122 meaning that it is 10:11:22 and you also given microseconds (10^-6 seconds) 567 and you want to put it all into a one variable to represent time stamp. `POSIXct` is a format which show ho many seconds elapsed since 00:00:00 of January 1st, 1970 and it supports fractional seconds.

```
date <- "28/06/2017"
time <- "101122"
microsec <- 123456

# First, we deal with complete seconds:
tstamp <- paste(date, " ", time, sep = "")
tstamp <- as.POSIXct(tstamp, format = "%d/%m/%Y %H%M%S")
tstamp
```

```
## [1] "2017-06-28 10:11:22 MSK"
```

```
# Seconds, we convert it into numeric representation:
tstamp <- as.numeric(tstamp)

# Third, we add microseconds as fractions of seconds and convert back to POSIXct
tstamp <- tstamp + microsec/1e6
tstamp <- as.POSIXct(tstamp, origin = "1970-01-01")
tstamp
```

```
## [1] "2017-06-28 10:11:22 MSK"
```

Even though you don't see fractional seconds in the representation above they are there and we will check it now and make an operation on date and times. Introduce another time:

```
date2 <- "28/06/2017"
time2 <- "101122"
microsec2 <- 987654
tstamp2 <- paste(date2, " ", time2, sep = "")
tstamp2 <- as.POSIXct(tstamp2, format = "%d/%m/%Y %H%M%S")
tstamp2 <- as.numeric(tstamp2)
tstamp2 <- tstamp2 + microsec2/1e6
tstamp2 <- as.POSIXct(tstamp2, origin = "1970-01-01")
tstamp2
```

```
## [1] "2017-06-28 10:11:22 MSK"
```

```
# substract tstamp from tstamp2:
as.numeric(tstamp2 - tstamp)
```

```
## [1] 0.864198
```

We can see that the difference is a fraction `0.864198` of a second. This way you can do algebraic operations on times and dates.

Regressions VI. Panel data

Two main thins that we need to learn in order to deal with panel data are fixed effects (possibly multiple) and clustered standard errors.

Regressions VII. Instrumental Variables

Regressions VIII. Time Series

Some Machine Learning for Prediction