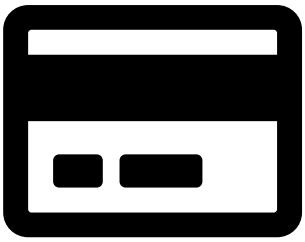
Credit risk.



Dr. Martín Lozano.

<martin.lozano@udem.edu>

https://sites.google.com/site/mlozanoqf/

https://github.com/mlozanoqf/

Last compiled on: 18/07/2021, 10:14:33.

Abstract

This material relies on John C. Hull [Hull, 2015] credit risk chapters and Lore Dirick credit risk DataCamp course. Some mathematical background is skipped to emphasize the data analysis, model logic, discussion, graphical approach and R coding. As in the philosophy of Donald Knuth [Knuth, 1984], the objective of this document is to explain to human beings what we want a computer to do as literate programming. This is a work in progress and it is under revision.

Index.

```
# Logistic models
library(gmodels)
library(ggplot2)
library(tidyr)
library(dplyr)
library(pROC)
# Merton model
library(knitr)
library(Sim.DiffProc)
library(bazar) # para la función almost.equal
library(scatterplot3d)
# Copula models
library(MASS)
library(viridis)
library(rayshader)
```

1 Loan analysis.

This section relies on the DataCamp course *Credit Risk Modeling in R* by Lore Dirick. However, we incorporate a slightly different database and an extended analysis.

1.1 Explore the database.

Let's load the data called `loan_data_ARF.rds` and then understand its structure. This database is available upon request.

```
loan_data <- readRDS("loan_data_ARF.rds")
str(loan_data)
```

```
## 'data.frame':    29092 obs. of  10 variables:
##  $ loan_status   : int  0 0 0 0 0 0 1 0 1 0 ...
##  $ loan_amnt     : int  5000 2400 10000 5000 3000 12000 9000 3000 10000 1000 ...
##  $ int_rate      : num  10.6 11 13.5 11 11 ...
##  $ grade         : Factor w/ 7 levels "A","B","C","D",..: 2 3 3 1 5 2 3 2 2 4 ...
##  $ emp_length    : int  10 25 13 3 9 11 0 3 3 0 ...
##  $ home_ownership: Factor w/ 4 levels "MORTGAGE","OTHER",..: 4 4 4 4 4 3 4 4 4 4 ...
##  $ annual_inc    : num  24000 12252 49200 36000 48000 ...
```

```
##  $ age           : int  33 31 24 39 24 28 22 22 28 22 ...
##  $ sex           : Factor w/ 2 levels "0","1": 1 1 1 1 2 2 2 2 1 2 ...
##  $ region        : Factor w/ 4 levels "E","N","S","W": 1 1 3 3 2 2 2 2 4 1 ...
```

This could be a typical database taken from any given financial institution like a bank or a firm that uses credit channels to sell their products or services. Here, we have 29,092 observations of 10 variables. Each observation corresponds to one individual loan and each variable allow us to understand the individual and the loan characteristics. One important variable, our dependent variable, is `loan_status` the value of 0 is no default and the value of 1 is default. A default occurs when a borrower is unable to make timely payments, misses payments, or avoids or stops making payments on interest or principal owed. Then, the definition of default depends on the interests and objectives of the analysis.

Clearly, this is past information as we know with certainty whether the individual defaulted (1) or not (0). Past information is helpful to better understand how likely is that one individual may default according to the rest of their variable values. This kind of data could be easily found in most financial firms as they store details about the applicant and its corresponding loan. Past information is useful to train our quantitative models and eventually make predictions of new applicants, and even evaluate our predictions.

We can look at the information in different ways. For example, look at the first 10 rows (out of 29,092) and their corresponding 10 variables.

```
head(loan_data, 10)
```

```
##    loan_status loan_amnt int_rate grade emp_length home_ownership annual_inc
## 1            0      5000    10.65     B         10           RENT      24000
## 2            0      2400    10.99     C         25           RENT      12252
## 3            0     10000    13.49     C         13           RENT      49200
## 4            0      5000    10.99     A          3           RENT      36000
## 5            0      3000    10.99     E          9           RENT      48000
## 6            0     12000    12.69     B         11            OWN      75000
## 7            1      9000    13.49     C          0           RENT      30000
## 8            0      3000     9.91     B          3           RENT      15000
## 9            1     10000    10.65     B          3           RENT     100000
## 10           0      1000    16.29     D          0           RENT      28000
##    age sex region
## 1   33   0      E
## 2   31   0      E
```

```
## 3   24   0      S
## 4   39   0      S
## 5   24   1      N
## 6   28   1      N
## 7   22   1      N
## 8   22   1      N
## 9   28   0      W
## 10  22   1      E
```

The `CrossTable` function is used in different contexts. Generating tables like this is only one way we can use it. Here, instead of looking the details of the first 10 rows, we summarize with respect to `home_ownership`.

```
CrossTable(loan_data$home_ownership)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  29092
##
##
##            |  MORTGAGE |     OTHER |       OWN |      RENT |
##            |-----------|-----------|-----------|-----------|
##            |     12002 |        97 |      2301 |     14692 |
##            |     0.413 |     0.003 |     0.079 |     0.505 |
##            |-----------|-----------|-----------|-----------|
##
##
##
##
```

These tables illustrate the data structure and contents. We can also use two variables instead

of one. In particular, instead of counting for home ownership we can add a second dimension like `loan_status`. This allows us to create more informative tables.

```
CrossTable(loan_data$home_ownership, loan_data$loan_status, prop.r = TRUE,
           prop.c = FALSE, prop.t = FALSE, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |-------------------------|
##
##
## Total Observations in Table:  29092
##
##
##                            | loan_data$loan_status
## loan_data$home_ownership |         0 |         1 | Row Total |
## -------------------------|-----------|-----------|-----------|
##                 MORTGAGE |     10821 |      1181 |     12002 |
##                          |     0.902 |     0.098 |     0.413 |
## -------------------------|-----------|-----------|-----------|
##                    OTHER |        80 |        17 |         97 |
##                          |     0.825 |     0.175 |     0.003 |
## -------------------------|-----------|-----------|-----------|
##                      OWN |      2049 |       252 |      2301 |
##                          |     0.890 |     0.110 |     0.079 |
## -------------------------|-----------|-----------|-----------|
##                     RENT |     12915 |      1777 |     14692 |
##                          |     0.879 |     0.121 |     0.505 |
## -------------------------|-----------|-----------|-----------|
##             Column Total |     25865 |      3227 |     29092 |
## -------------------------|-----------|-----------|-----------|
##
##
```

This table reveals defaults by home ownership. We can use histograms to see one variable distribution. In this case we have the interest rate distribution.

```
ggplot(loan_data, aes(x = int_rate)) +
  geom_histogram(aes(y=..density..), binwidth = 0.5, colour = "black",
                 fill = "white") +
  labs(y = "Density",
       x = "Interest rate",
       title = "Interest rate histogram",
       subtitle = NULL) +
  theme(legend.position = "bottom", legend.title = element_blank())
```

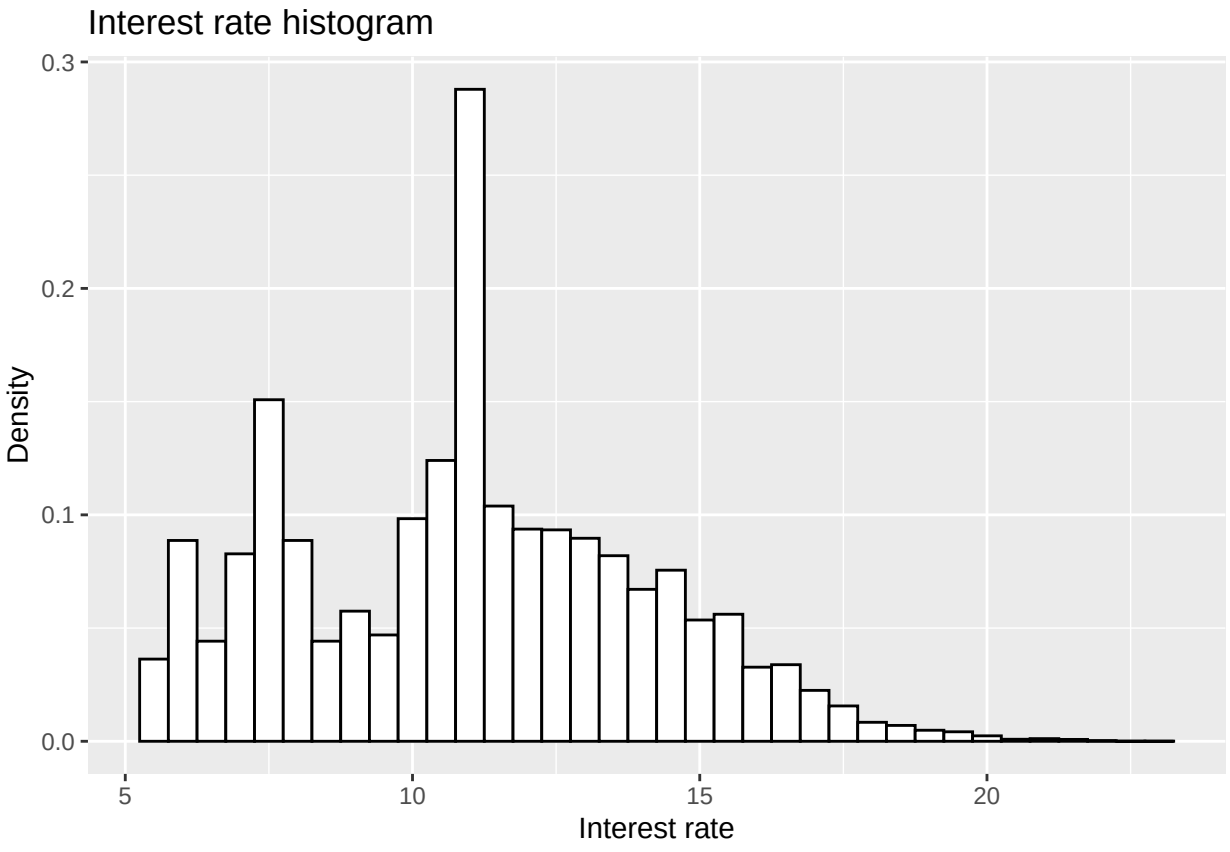


Figure 1.1.1: Interest rate histogram.

The following is a similar figure for the annual income.

```
ggplot(loan_data, aes(x = annual_inc)) +
  geom_histogram(aes(y=..density..), colour = "black", fill = "red") +
  labs(y = "Density",
       x = "Annual income",
       title = "Annual income histogram",
       subtitle = NULL) +
  theme(legend.position = "bottom", legend.title = element_blank())
```

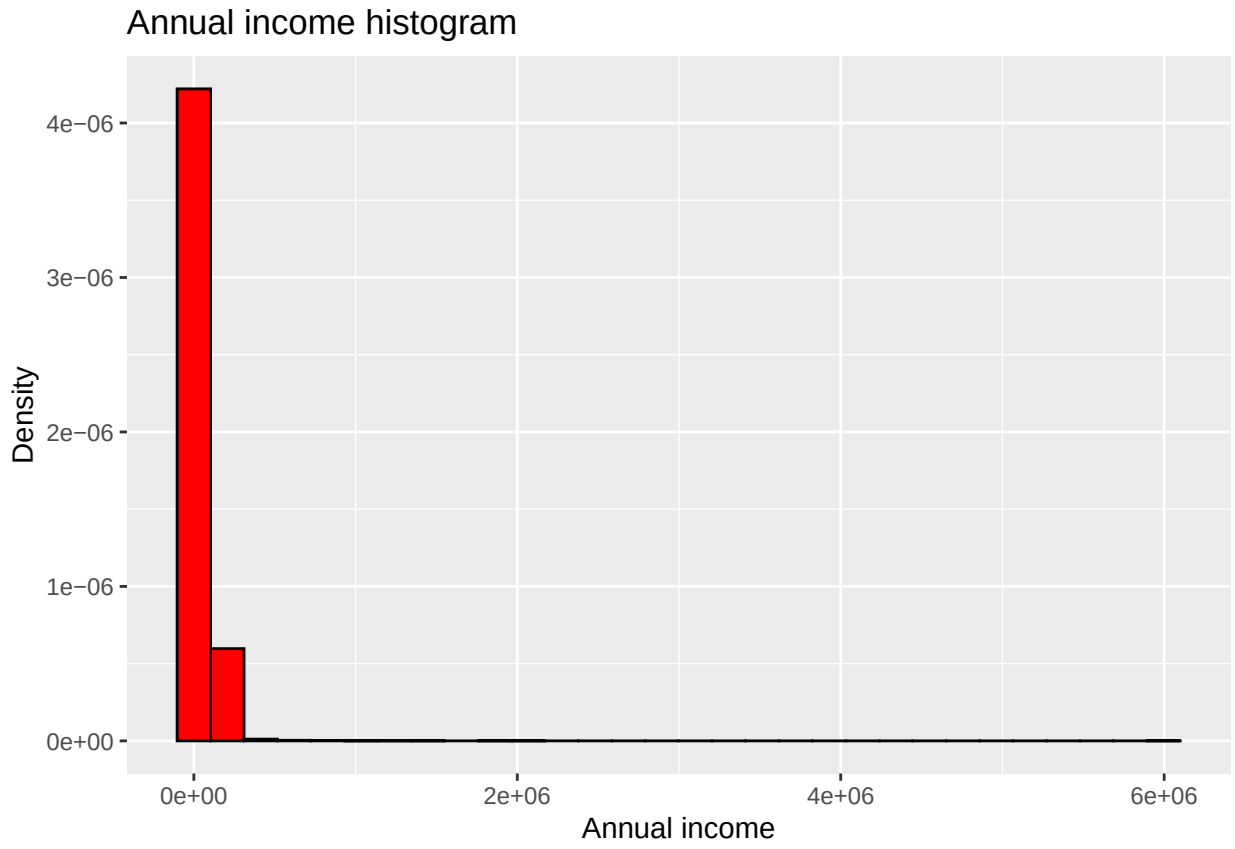


Figure 1.1.2: Annual income histogram.

The histogram looks suspicious. We have a very large values of the annual income in the horizontal axis (6,000,000) and apparently no observations. We can plot the values and explore the data further.

```
ggplot(loan_data, aes(int_rate, annual_inc)) +
  geom_point() +
```

```
  labs(y = "Annual income",
       x = "Interest rate",
       title = "Annual income inspection.",
       subtitle = NULL) +
  theme(legend.position = "bottom", legend.title = element_blank())
```

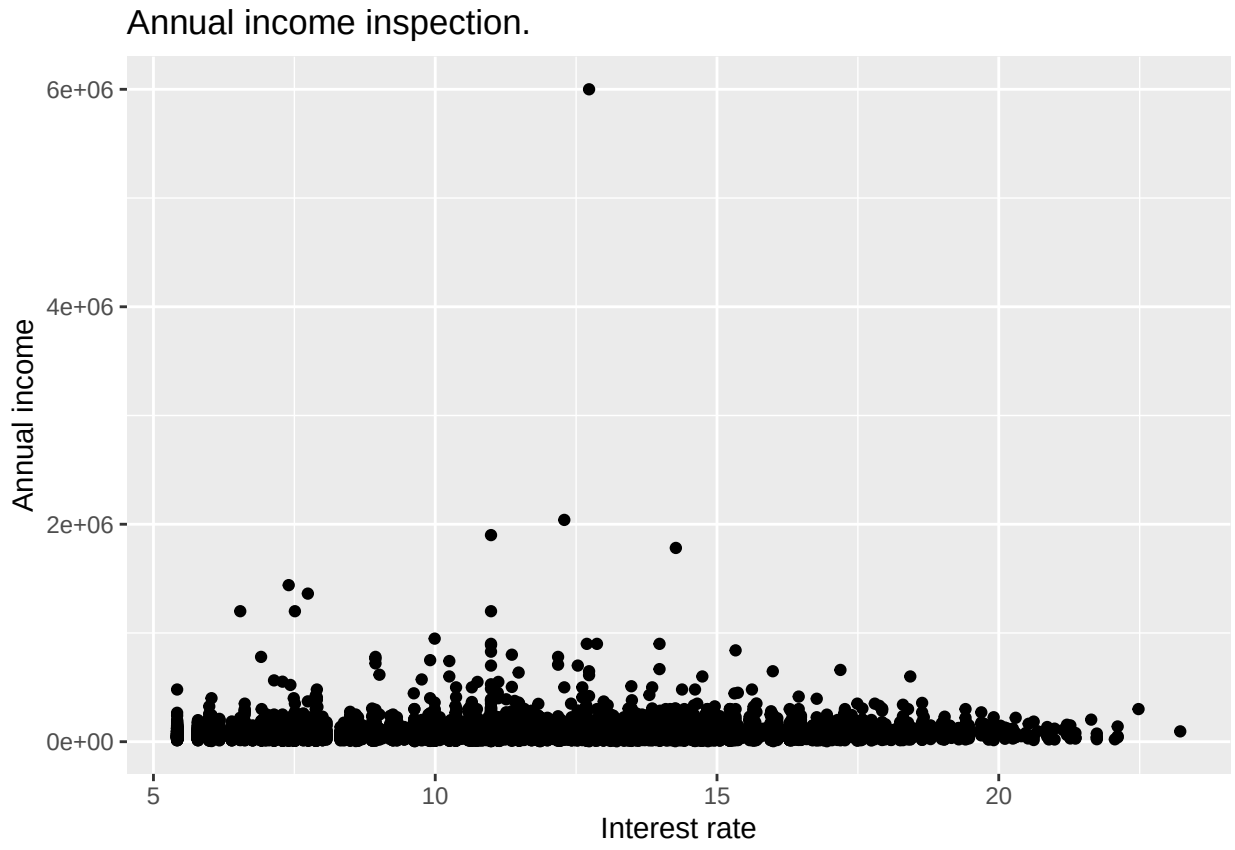


Figure 1.1.3: Annual income inspection.

There are some individuals with a very high income. We should explore further and investigate if this are valid observations or simply a mistake in the database.

```
high_income <- loan_data[(loan_data$annual_inc > 1000000), ]
high_income
```

```
##       loan_status loan_amnt int_rate grade emp_length home_ownership annual_inc
## 4861            0     12025    14.27     C         13           RENT    1782000
## 13931           0     10000     6.54     A         16            OWN    1200000
```

```
## 15386           0      1500    10.99      A           5       MORTGAGE    1900000
## 16713           0     12000     7.51      A           1       MORTGAGE    1200000
## 19486           0      5000    12.73      C          12       MORTGAGE    6000000
## 22811           0     10000    10.99      A           1       MORTGAGE    1200000
## 23361           0      6400     7.40      A           7       MORTGAGE    1440000
## 23683           0      6600     7.74      A           9       MORTGAGE    1362000
## 28468           0      8450    12.29      C           0           RENT    2039784
##       age sex region
## 4861   63   0      E
## 13931  36   0      W
## 15386  60   1      N
## 16713  32   0      E
## 19486 144   1      E
## 22811  40   1      E
## 23361  44   1      E
## 23683  47   0      E
## 28468  42   0      E
```

One guy is not only rich, he is 144 years old. So, my decision is to drop these 9 observations.

```
high_income_index <- data.frame(value = as.integer(rownames(high_income)))
loan_data <- loan_data[-high_income_index$value,]
ggplot(loan_data, aes(int_rate, annual_inc)) +
  geom_point() +
  labs(y = "Annual income",
       x = "Interest rate",
       title = "Annual income without extreme values.",
       subtitle = NULL) +
  theme(legend.position = "bottom", legend.title = element_blank())
```

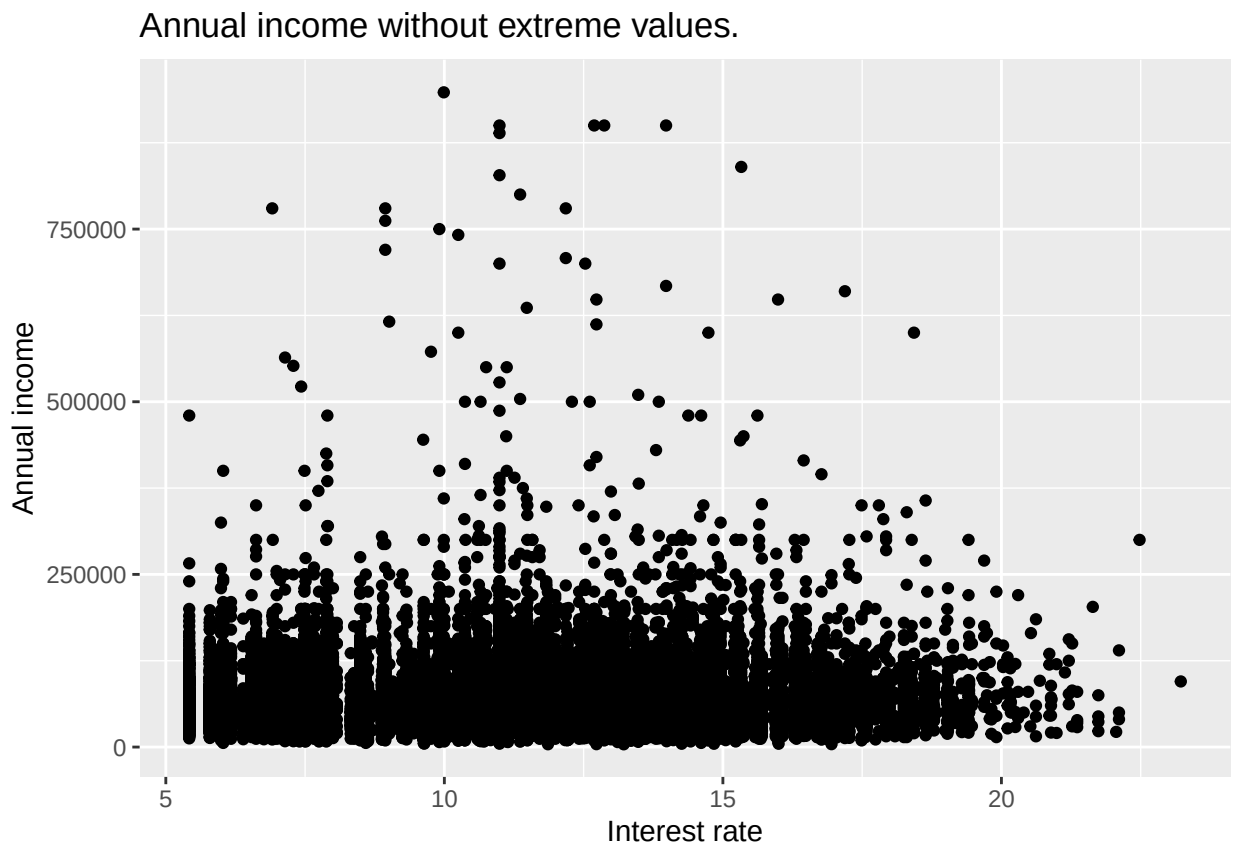


Figure 1.1.4: Annual income without extreme values.

Remember the database has originally 29,092 rows and now we are dropping 9 so we end up with 29,083. See the result.

```
ggplot(loan_data, aes(x = annual_inc)) +
  geom_histogram(aes(y=..density..), colour = "black", fill = "red") +
    labs(y = "Density",
        x = "Annual income",
        title = "Annual income histogram second version.",
        subtitle = NULL) +
  theme(legend.position = "bottom", legend.title = element_blank())
```

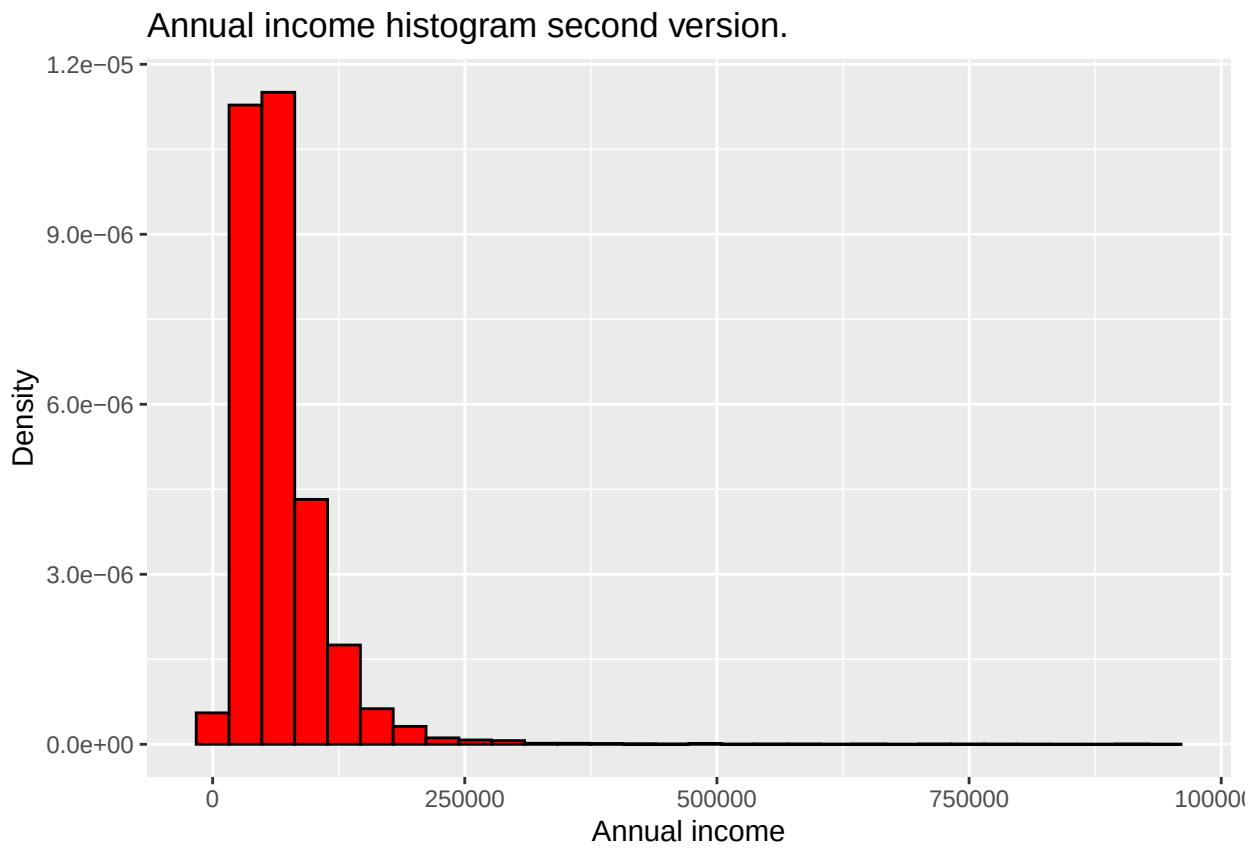


Figure 1.1.5: Annual income histogram second version.

1.2 Logistic models.

Logistic models allows us to make predictions about loan defaults. Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable. In this case, the binary dependent variable is default or no default (loan status). A good reference for this section is Hull [2020].

First, load the data and split it into two sets: (1) training and (2) test. The training set is for building and estimate models and the test set is used to evaluate our model predictions. This is, when estimating models, it is common practice to separate the available data into two portions, *training* and *test* data, where the training data is used to estimate parameters and the test data is used to evaluate its accuracy. Because the test data is not used in determining the estimation, it should provide a reliable indication of how well the model is likely to estimate or forecast on new data. In sum, we train the model, we test the model, and once we are OK with the model performance on new data, we are ready to use it in

real-life applications.

```
# It is convenient to set the loan_status as factor.
loan_data$loan_status <- as.factor(loan_data$loan_status)
set.seed(567)
index_train <- cbind(runif(1 : nrow(loan_data), 0 , 1),
                     c(1 : nrow(loan_data)))
index_train <- order(index_train[, 1])
index_train <- index_train[1: (2/3 * nrow(loan_data))]
# Create training_set
training_set <- loan_data[index_train, ]
# Create test_set
test_set <- loan_data[-index_train, ]
```

We have 29,083 observations in `loan_data`. The code above randomly selects $29083 \times (2/3) = 19388$ rows to form the `training_set`. The `test_set` are the remaining $29083 - 19388 = 9695$ rows. The random selection is highly recommended as the `loan_data` may have some structure or sorting that could bias our model estimation and negatively impact our model test.

Take a look of the training set structure.

```
# See the data structure.
str(training_set)
```

```
## 'data.frame':    19388 obs. of  10 variables:
##  $ loan_status   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
##  $ loan_amnt     : int  4000 5000 18000 5600 2700 15000 14500 12000 4800 4050 ...
##  $ int_rate      : num  10.2 14.3 18.3 7.9 14.3 ...
##  $ grade         : Factor w/ 7 levels "A","B","C","D",..: 2 3 6 1 3 2 1 4 1 1 ...
##  $ emp_length    : int  6 3 10 3 5 10 11 9 1 17 ...
##  $ home_ownership: Factor w/ 4 levels "MORTGAGE","OTHER",..: 4 4 4 4 1 1 1 1 4 1 ...
##  $ annual_inc    : num  26000 280000 121000 32000 88500 75000 97600 57600 92000 75000
##  $ age           : int  27 36 24 22 32 23 23 28 48 31 ...
##  $ sex           : Factor w/ 2 levels "0","1": 2 2 2 1 2 1 1 1 2 1 ...
##  $ region        : Factor w/ 4 levels "E","N","S","W": 2 2 2 4 2 4 3 4 2 3 ...
```

Variables as factors are useful for model estimation and data visualization. Factors are variables in R which take on a limited number of different values; such variables are often refered to as categorical variables.

Assume we think that the `loan_status` depends on the age of the individual. We can estimate a simple logistic model to learn about the relationship between age and loan status.

```
# Fitting a simple logistic model.
log_model_age <- glm(loan_status ~ age , family = "binomial",
                     data = training_set)
log_model_age
```

```
##
## Call:  glm(formula = loan_status ~ age, family = "binomial", data = training_set)
##
## Coefficients:
## (Intercept)          age
##    -1.90097     -0.00623
##
## Degrees of Freedom: 19387 Total (i.e. Null);  19386 Residual
## Null Deviance:       13580
## Residual Deviance: 13580     AIC: 13580
```

Apparently, there is a negative relationship between age and loan status. The AIC value (13580) is useful when comparing models. The Akaike information criterion (AIC) is a mathematical method for evaluating how well a model fits the data it was generated from. In statistics, AIC is used to compare different possible models and determine which one is the best fit for the data.

Let's estimate another simple model where the interest rate category is used as a predictor of the `loan_status`.

```
# Build a glm model with variable int_rate as a predictor.
log_model_ir <- glm(formula = loan_status ~ int_rate, family = "binomial",
                     data = training_set)
# Print the parameter estimates.
log_model_ir
```

```
##
## Call:  glm(formula = loan_status ~ int_rate, family = "binomial", data = training_set
##
## Coefficients:
## (Intercept)     int_rate
```

```
##      -3.710        0.142
##
## Degrees of Freedom: 19387 Total (i.e. Null);  19386 Residual
## Null Deviance:       13580
## Residual Deviance: 13210     AIC: 13220
```

The AIC is a lower (13220), so we have a better model now.

Using one single predictor as age or interest rate could be a limited approach. Let's add some more. Also, let's introduce the `summary` function to extract more information about the model estimation. The `log_model_multi` below assumes that the loan status depend on the age, interest rate, grade, loan amount, and annual income.

```
# Multiple variables in a logistic regression model.
log_model_multi <- glm(loan_status ~ age + int_rate + grade + log(loan_amnt) +
                         log(annual_inc) , family = "binomial",
                       data = training_set)
# Obtain significance levels using summary().
summary(log_model_multi)
```

```
##
## Call:
## glm(formula = loan_status ~ age + int_rate + grade + log(loan_amnt) +
##     log(annual_inc), family = "binomial", data = training_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.1545  -0.5351  -0.4397  -0.3382   2.6101
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      1.996240   0.477911   4.177 2.95e-05 ***
## age             -0.002302   0.003825  -0.602   0.5473
## int_rate         0.038767   0.017249   2.247   0.0246 *
## gradeB           0.503409   0.087435   5.758 8.54e-09 ***
## gradeC           0.748229   0.117765   6.354 2.10e-10 ***
## gradeD           0.964343   0.147283   6.548 5.85e-11 ***
## gradeE           1.033442   0.190817   5.416 6.10e-08 ***
```

```
## gradeF           1.619470   0.257900   6.279 3.40e-10 ***
## gradeG           1.867494   0.440232   4.242 2.21e-05 ***
## log(loan_amnt)   0.015718   0.036341   0.433   0.6654
## log(annual_inc) -0.470748   0.046423 -10.140  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13579  on 19387  degrees of freedom
## Residual deviance: 13028  on 19377  degrees of freedom
## AIC: 13050
##
## Number of Fisher Scoring iterations: 5
```

Our multi-factor model works well. In `log_model_multi`, the AIC value is the lowest so far (13050), so this should be considered as the best model at the moment. The `summary` function shows the significance levels of the estimators, but we are currently more interested in the goodness of fit of the models because we want to conduct predictions about the `loan_status`. This is, we are interested to use a model to find out whether new applicants in the test set are expected to default or not, rather than in the applicants' credit risk factors.

When someone fill out a credit application form, we collect information but we do not know for sure whether she or he will eventually default. A credit risk model can help us in this task.

1.3 Prediction and model evaluation.

Let's take our three models: `log_model_age`, `log_model_ir` and `log_model_multi` from the previous subsection to carry out a simple prediction exercise. We start by identifying one observation in the test set and ask the models to estimate the `loan_status`. Every model is expected to produce different estimates.

```
# Define one single observation in test_set.
test_case <- as.data.frame(test_set[1, ])
test_case
```

```
##   loan_status loan_amnt int_rate grade emp_length home_ownership annual_inc age
## 1           0      5000    10.65     B         10           RENT      24000  33
```

```
##   sex region
## 1   0      E
```

We know in advance that the `loan_status` of this observation taken from the test set is 0. However, the models cannot know this simply because we did not use the test set to estimate the logistic models. A good credit risk model should estimate a no default given this new applicant.

The values of `loan_status` in the test set is either 0 or 1. However, the logistic models estimate the `loan_status` as values in the range of 0 to 1. This mean that we would expect the estimated `loan_status` to be close to 0. But, how close? We will deal with this issue later.

```
# Estimate the loan status with log_model_age and log_model_ir models.
log_model_age_pred <- as.numeric(predict(log_model_age, newdata = test_case,
                                 type = "response"))
# Remember the test_case is only one observation (one row).
log_model_ir_pred <- as.numeric(predict(log_model_ir, newdata = test_case,
                                 type = "response"))
log_model_multi_pred <- as.numeric(predict(log_model_multi, newdata = test_case,
                                 type = "response"))
predictions <- rbind("log_model_age"   = log_model_age_pred,
                     "log_model_ir"    = log_model_ir_pred,
                     "log_model_multi" = log_model_multi_pred)
colnames(predictions) <- "predictions of a known no-default"
predictions
```

```
##                 predictions of a known no-default
## log_model_age                           0.10846236
## log_model_ir                            0.09996702
## log_model_multi                         0.14461840
```

These values are low as they are close to 0. We could interpret this as a certain ability of the models to predict this single case from the test set. However, several questions remains unanswered and requires further analysis. For example: How can we determine if the prediction is low enough to consider it as a non-default? We may need a cut-off to decide. We will explore this issue later.

Another aspect is: What about the rest of the cases in the test set? We have 9,695 observations

in the test set and in the example above we only test for the first one. We are interested in test for the entire test set, not only for one observation. Fortunately, this issue is easy to address as we only need to change the `newdata` parameter in the `predict` function. In particular, instead of `newdata = test_case` (which is one observation) we can change it to `newdata = test_set` (which is the entire 9,695 test set).

```
# Estimate the loan status with the three models.
pred_log_model_age <- predict(log_model_age, newdata = test_set,
                              type = "response")
# Now newdata = test_set so we are testing all the test set observations.
pred_log_model_ir <- predict(log_model_ir, newdata = test_set,
                             type = "response")
pred_log_model_multi <- predict(log_model_multi, newdata = test_set,
                                type = "response")
# The range of the estimated loan status.
log_model_age_predall <- range(pred_log_model_age)
log_model_ir_predall <- range(pred_log_model_ir)
log_model_multi_predall <- range(pred_log_model_multi)

predictions_2 <- rbind("log_model_age"   = log_model_age_predall,
                       "log_model_ir"    = log_model_ir_predall,
                       "log_model_multi" = log_model_multi_predall)
aic <- rbind(log_model_age$aic, log_model_ir$aic, log_model_multi$aic)
predictions_2 <- cbind(predictions_2, aic)
colnames(predictions_2) <- c("lower value", "higer value", "AIC")
predictions_2
```

```
##                 lower value higer value      AIC
## log_model_age    0.08417892   0.1165453 13580.65
## log_model_ir     0.05019756   0.3982977 13215.61
## log_model_multi  0.01739107   0.4668004 13050.30
```

Narrow ranges could be problematic because the model could not be able to differentiate between defaults (predictions closer to 1) and no-defaults (predictions closer to 0). The higher AIC corresponds to the worst model and the lower AIC to the best model. Here, we can see some consistency because the best model according to the AIC, corresponds to the model with the higher prediction range.

Let's explore the `log_model_age`:

```
pred_log_model_age_plot <- data.frame(pred_log_model_age)
pred_log_model_age_plot <- gather(pred_log_model_age_plot)

ggplot(pred_log_model_age_plot, aes(x = value, fill = key)) +
  geom_density(alpha = 0.4) +
  labs(y = "Density",
       x = "Default prediction",
       title = "Age model prediction histogram.",
       subtitle = "Very limited prediction range.") +
  theme(legend.position = "bottom", legend.title = element_blank())
```

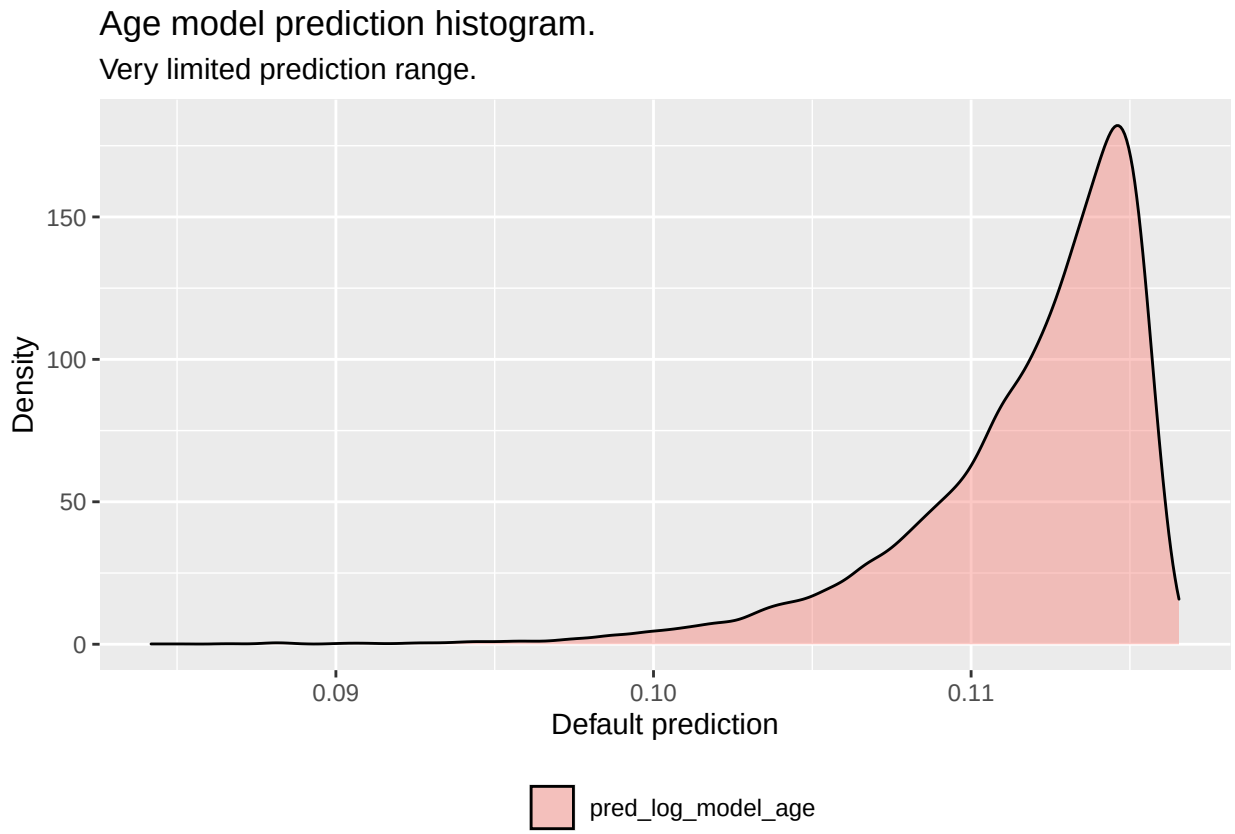


Figure 1.3.1: Age model prediction histogram.

The `log_model_age` fails to predict values ranging from 0 to 1. In fact, these values are quite concentrated in a very small range of values. As a consequence, this model fails to

differentiate between default and no default predictions.

Let's visualize the predictions of the `log_model_ir` and `log_model_multi`.

```
pred_ir_multi <- as.data.frame(cbind(pred_log_model_ir,
                                     pred_log_model_multi))
pred_ir_multi <- gather(pred_ir_multi)

ggplot(pred_ir_multi, aes(x = value, fill = key)) +
  geom_density(alpha = 0.4) +
  labs(y = "Density",
       x = "Default prediction",
       title = "Interest rate and multi models predictions histograms.",
       subtitle = "Multi model performs somewhat better.") +
  theme(legend.position = "bottom", legend.title = element_blank())
```

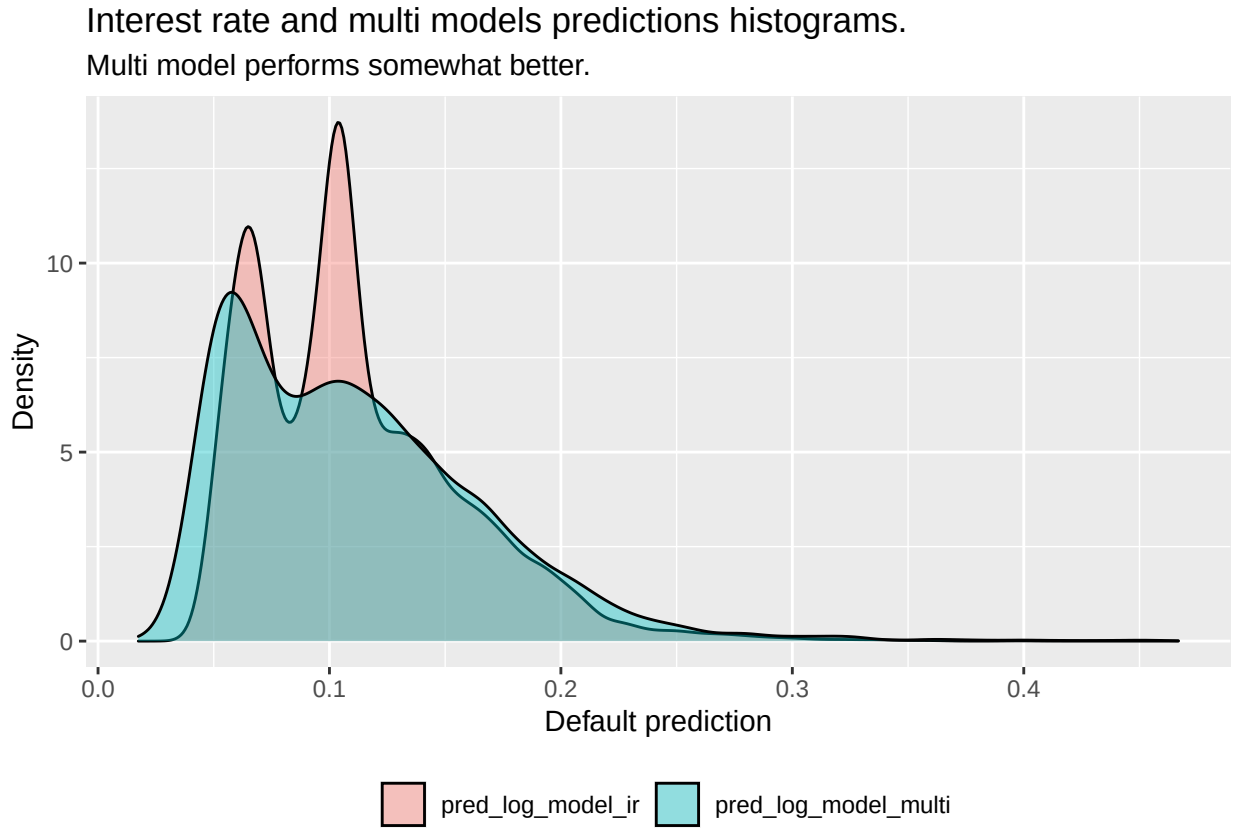


Figure 1.3.2: Interest rate and multi models predictions histograms.

Presumably, a model which considers all available predictors could be better for predicting the `loan_status`.

```
# Logistic regression model using all available predictors in the data set.
log_model_full <- glm(loan_status ~ age + int_rate + grade + log(loan_amnt) +
                        log(annual_inc) + emp_length + home_ownership + sex +
                      region, family = "binomial", data = training_set)
#glm(loan_status ~ ., family = "binomial", data = training_set)

# Loan status predictions for all test set elements.
predictions_all_full <- predict(log_model_full, newdata = test_set,
                                type = "response")
# Look at the predictions range.
range(predictions_all_full)
```

```
## [1] 1.422469e-09 8.544424e-01
```

Now, the range is wider. A wider range means that the estimates are now closer to 1. This is good because we need the model to be able to predict both no-defaults (0) and defaults (1). Let's see a prediction comparisons of `log_model_multi` and `log_model_full`.

```
predictions_multi_full <-as.data.frame(cbind(pred_log_model_multi,
                                             predictions_all_full))
predictions_multi_full <- gather(predictions_multi_full)

ggplot(predictions_multi_full, aes(x = value, fill = key)) +
  geom_density(alpha = 0.4) +
  labs(y = "Density",
       x = "Default prediction",
       title = "Multi and full models predictions histograms.",
       subtitle = "Full model performs somewhat better.") +
  theme(legend.position = "bottom", legend.title = element_blank())
```

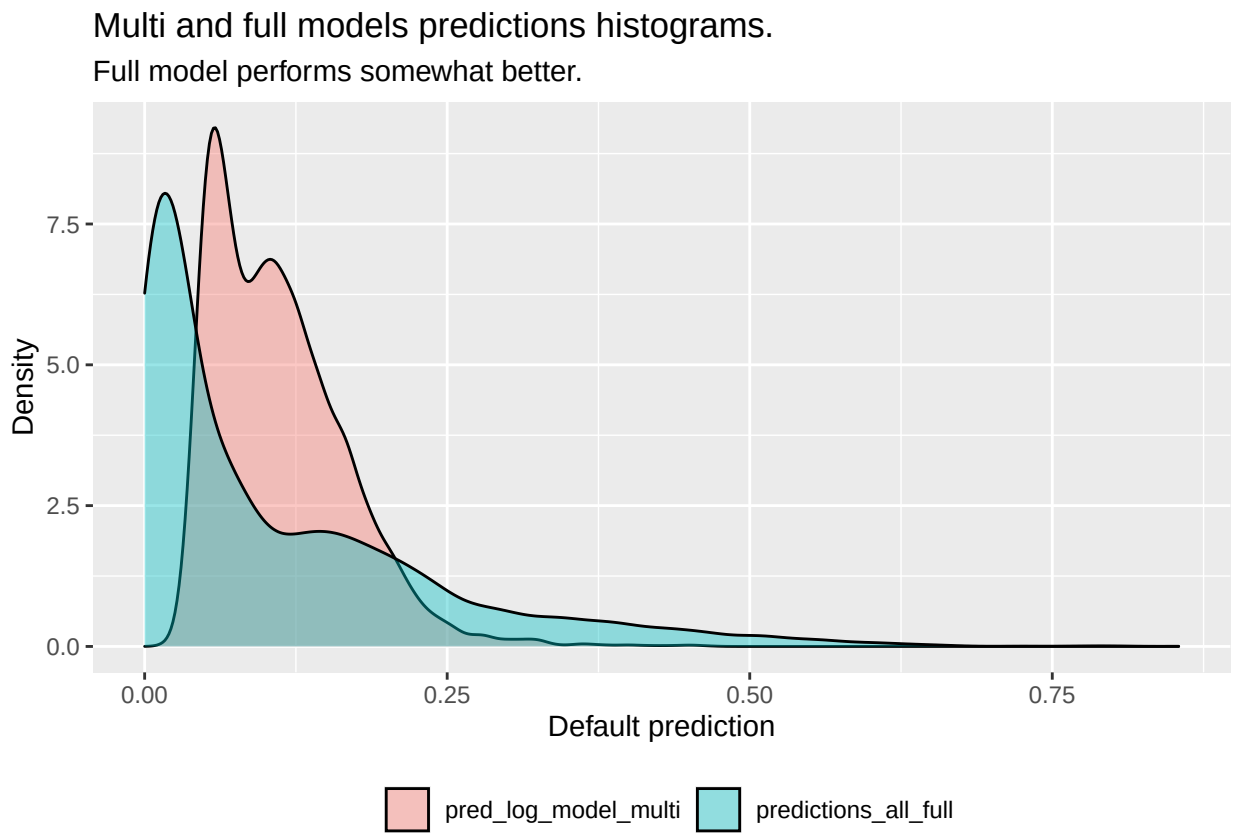


Figure 1.3.3: Multi and full models predictions histograms.

The `log_model_full` model predictions looks better than the other models.

Another question is: How can we know these model predictions corresponds to a default or no default? The loan status predictions go from 0 to 0.854. At the end, these loan status estimations need to be interpreted as a default or no default because we are interested on that. Are they closer enough to 0 to consider a no default? This issue is addressed by setting up a cutoff rate so we can split all estimated loan status into 0 or 1.

Let's arbitrarily consider a cutoff of 0.15. This mean that every estimated loan status below 0.15 will be considered as 0 (no-default), and every estimated loan status above 0.15 will be considered as 1 (default). Graphically looks like this:

```
pred_log_model_full <- data.frame(predictions_all_full)
pred_log_model_full <- gather(pred_log_model_full)
pred_log_model_full <- mutate(pred_log_model_full,
                              def = ifelse(value < 0.15, 0, 1))
```

```
pred_log_model_full$def <- as.factor(pred_log_model_full$def)

ggplot(pred_log_model_full, aes(x = value, fill = key)) +
  geom_density(alpha = 0.4, adjust = 0.4) +
  geom_vline(xintercept = 0.15, linetype = "longdash") +
  labs(y = "Density",
       x = "Default prediction",
       title = "Full model prediction histogram.",
       subtitle = "A cutoff of 0.15.") +
  theme(legend.position = "bottom", legend.title = element_blank())
```

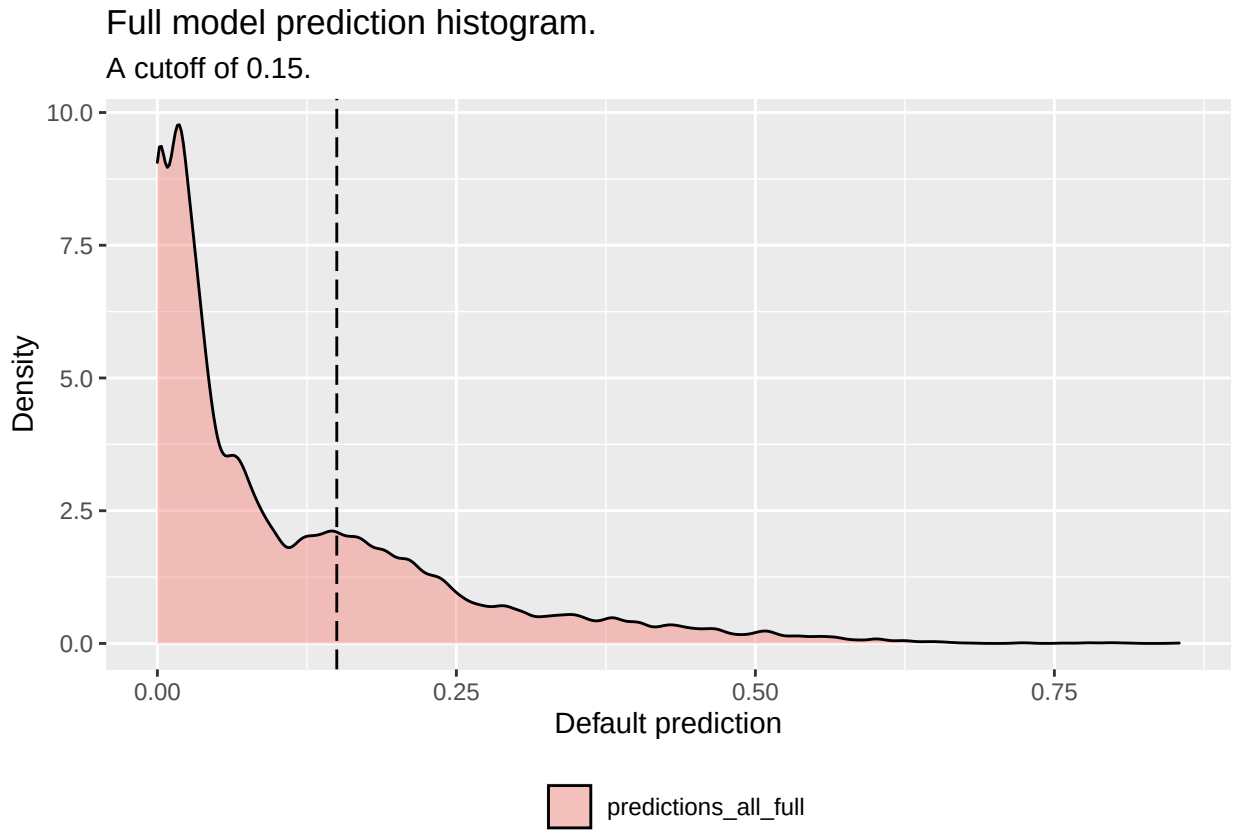


Figure 1.3.4: Full model prediction histogram.

Here, values at the left of the dashed line represent no default and values at the right of the dashed line represent default. Let's set up the rule to convert the estimated loan status into a binary variable 0 or 1. See how this transformation takes place:

```
# Make a binary predictions-vector using a cut-off of 15%
pred_cutoff_15 <- ifelse(predictions_all_full > 0.15, 1, 0)
head(cbind(predictions_all_full, pred_cutoff_15))
```

```
##    predictions_all_full pred_cutoff_15
## 1          1.357995e-08              0
## 2          2.986278e-08              0
## 18         2.299645e-02              0
## 26         2.403879e-01              1
## 27         1.778986e-01              1
## 28         2.600172e-02              0
```

These are only the first 6 rows not all of them. We can see that the rule works well as every estimated loan status below 0.15 is now considered as 0 (no-default), and every estimated loan status above 0.15 is considered as 1 (default). The table above show how we can create this binary variable given the logistic model prediction.

Note that the rows numbers in the table above are 1, 2, 18, 26, 27 and 28. These are not 1, 2, 3, 4, 5 and 6 because the `test_set` rows were selected randomly out of the `loan_data`. So, the row numbers in the table above correspond to the original place in `loan_data`.

Is `log_model_full` a good model after all? We can add a new column to the previous table. This new variable represents what really happened with the loan. Then, the first column is the logistic model prediction, the second column the transformed binary variable given a cutoff of 0.15 and the third column is what actually happened (default or no-default).

```
# Let's take from rows 101 to 110.
(cbind(predictions_all_full, pred_cutoff_15,
          as.data.frame.numeric(test_set$loan_status)))[101:110,]
```

```
##     predictions_all_full pred_cutoff_15 test_set$loan_status
## 308         1.537106e-01              1                    0
## 309         4.137654e-02              0                    0
## 310         3.113926e-02              0                    0
## 312         5.186962e-09              0                    0
## 318         5.337092e-02              0                    0
## 319         8.402239e-02              0                    0
## 323         2.795535e-01              1                    1
## 328         5.384112e-09              0                    0
```

```
## 329          1.893745e-01                1                     0
## 330          9.012136e-03                0                     0
```

Note that the model correctly predict a no default in most cases. Rows 308 and 329 predict a default incorrectly and row 323 predict a default correctly. There is an easy way to evaluate the rest of the cases in a simple table called confusion matrix.

```
# Construct a confusion matrix.
table(test_set$loan_status, pred_cutoff_15)
```

```
##    pred_cutoff_15
##        0    1
##   0 6554 2081
##   1  308  752
```

The model predicts 6,554 of no-defaults correctly and 752 defaults correctly. However, the model predicts 2,081 defaults that are in fact no-defaults and 308 no-defaults that are in fact defaults. How good are these results? Which of these four values is more important? These are questions we address later. For now, we can say that different models and different cutoff rates lead to different confusion matrix results.

You may want to see relative and not absolute values. Let's try the `CrossTable` function instead.

```
CrossTable(test_set$loan_status, pred_cutoff_15, prop.r = TRUE,
           prop.c = FALSE, prop.t = FALSE, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |-------------------------|
##
##
## Total Observations in Table:  9695
##
##
##                      | pred_cutoff_15
```

```
## test_set$loan_status |         0 |         1 | Row Total |
## ---------------------|-----------|-----------|-----------|
##                    0 |      6554 |      2081 |      8635 |
##                      |     0.759 |     0.241 |     0.891 |
## ---------------------|-----------|-----------|-----------|
##                    1 |       308 |       752 |      1060 |
##                      |     0.291 |     0.709 |     0.109 |
## ---------------------|-----------|-----------|-----------|
##         Column Total |      6862 |      2833 |      9695 |
## ---------------------|-----------|-----------|-----------|
##
##
```

This table is more informative. The model correctly predicts no-defaults in 75.9% of all the observed no-default cases. In other words, the model fails to predict no-default in 24.1% of the total no-default cases. Now the default. The model correctly predicts the default in 70.9% of all the observed default cases (not bad), and fails to predict default in 29.1% of the total default cases.

Instead of arbitrarily consider a cutoff of 0.15, we can follow a different approach. Now consider that we are interested in taking the 20% highest estimates (closer to 1) of `predictions_all_full` as default. Equivalently, this is to take the lowest 80% `predictions_all_full` estimates (closer to 0) as no-default.

```
# Cutoff definition.
cutoff <- quantile(predictions_all_full, 0.8)
cutoff
```

```
##       80%
## 0.1994621
```

This new approach (taking the 20% highest estimates of `predictions_all_full` as default) represent a cutoff of 0.1994621. Graphically:

```
ggplot(pred_log_model_full, aes(x = value, fill = key)) +
  geom_density(alpha = 0.4, adjust = 0.4) +
  geom_vline(xintercept = cutoff, linetype = "longdash") +
  labs(y = "Density",
       x = "Default prediction",
       title = "Full model prediction histogram.",
```

```
       subtitle = "A cutoff of 0.1994621.") +
  theme(legend.position = "bottom", legend.title = element_blank())
```

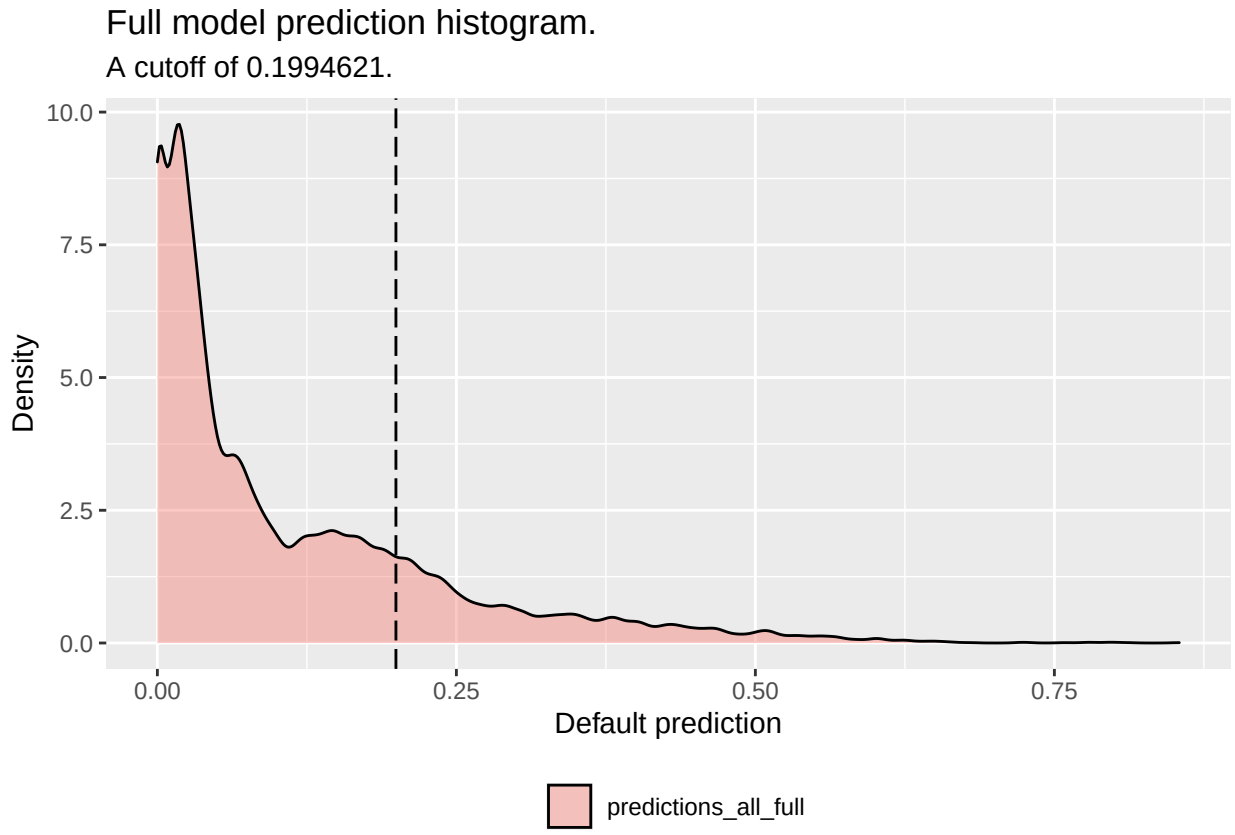


Figure 1.3.5: Full model prediction histogram new cutoff.

Now the cutoff is 0.1994621. This splits the loan status predictions into two parts: higher than the cutoff is a default, and lower than the cutoff is a no-default. Taking the lowest 80% estimates (closer to 0) as no-default is an arbitrary decision. Here are the cutoff values depending on this arbitrary decision.

```
cutoff_all <- quantile(predictions_all_full, seq(0.1, 1, 0.1))
data.frame(cutoff_all)
```

```
##         cutoff_all
## 10%  1.168657e-08
## 20%  1.468925e-02
## 30%  2.410243e-02
```

```
## 40%  3.750625e-02
## 50%  6.183830e-02
## 60%  9.617009e-02
## 70%  1.467331e-01
## 80%  1.994621e-01
## 90%  2.922731e-01
## 100% 8.544424e-01
```

We can show a similar summary table as we did with the cutoff of 0.15. Here, we show the predictive ability of the `log_model_full` model and new cutoff of 0.1994621.

```
# Calculate the predictions with the same model and new cutoff.
pred_full_20 <- ifelse(predictions_all_full > cutoff, 1, 0)
# Show results in a confusion matrix.
CrossTable(test_set$loan_status, pred_full_20, prop.r = TRUE,
           prop.c = FALSE, prop.t = FALSE, prop.chisq = FALSE)
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |-------------------------|
##
##
## Total Observations in Table:  9695
##
##
##                      | pred_full_20
## test_set$loan_status |         0 |         1 | Row Total |
## ---------------------|-----------|-----------|-----------|
##                    0 |      7309 |      1326 |      8635 |
##                      |     0.846 |     0.154 |     0.891 |
## ---------------------|-----------|-----------|-----------|
##                    1 |       447 |       613 |      1060 |
##                      |     0.422 |     0.578 |     0.109 |
## ---------------------|-----------|-----------|-----------|
```

```
##         Column Total |      7756 |      1939 |      9695 |
## ---------------------|-----------|-----------|-----------|
##
##
```

With a cutoff of 0.1994621 we accept 7,309 applications as those are the ones that the model predicts a no default. We can compare both confusion matrix:

```
cat <- c("correct no-default", "false default",
         "false no-default", "correct default")
cut_15 <- c(0.759, 0.241, 0.291, 0.709)
cut_19 <- c(0.846, 0.154, 0.422, 0.578)
cbind(cat, cut_15, cut_19)
```

```
##      cat                  cut_15  cut_19
## [1,] "correct no-default" "0.759" "0.846"
## [2,] "false default"      "0.241" "0.154"
## [3,] "false no-default"   "0.291" "0.422"
## [4,] "correct default"    "0.709" "0.578"
```

The new cutoff of 0.1994621 improves the identification of no-defaults but worsen the identification of default. Also, the new cutoff fails less in the default and fails more in the no-default. Apparently there is some sort of trade-off.

We can also look the detail of 0.1994621 cutoff. Comparing two columns, the one in the left with the actual loan status, and the right column with the estimated loan status.

```
# Comparative table in detail.
true_and_predval <- cbind.data.frame(test_set$loan_status, pred_full_20,
                                     test_set$loan_status==pred_full_20)
# Show some values.
true_and_predval[131:140,]
```

```
##     test_set$loan_status pred_full_20 test_set$loan_status == pred_full_20
## 398                    0            0                                 TRUE
## 399                    1            1                                 TRUE
## 404                    0            1                                FALSE
## 414                    0            0                                 TRUE
## 417                    1            1                                 TRUE
## 419                    0            0                                 TRUE
```

```
## 420                  0             0                    TRUE
## 425                  1             1                    TRUE
## 431                  0             0                    TRUE
## 435                  0             0                    TRUE
```

In this sample the model fails in 1 out of 10 individuals. Not bad at all. Let's imagine we are a bank. We have a total of 9,695 applications for a loan. Assume we use the predictions from `pred_full_20` to decide whether we accept a loan application or not. Our model-based acceptance rule is the following: if `pred_full_20 = 0` then the model estimates a no-default and we accept the loan application. According to the extract of table above, we fortunately reject application 399, 417 and 425 because that was indeed a default. However, we reject application 404 incorrectly because it did not default.

Let's count how many applications are accepted and rejected according to our rule.

```
# Accepted.
accept_20 <- sum(pred_full_20 == 0)
# Rejected.
reject_20 <- sum(pred_full_20 == 1)
data.frame(accept_20, reject_20)
```

```
##   accept_20 reject_20
## 1      7756      1939
```

Taking the 20% highest estimates of `predictions_all_full` as default and the lowest 80% `predictions_all_full` as no-default mean that by construction, we accept 7,756 loan applications (80% of the total) and reject 1,939 (20% of the total). Then, the criterion determines the number of accepted applications and the model determines which applications to accept/reject.

We can evaluate our loan application rule as we have the corresponding real values in `loan_status`.

First, let's illustrate the decision making process given the model estimates.

```
# First 10 accept decisions.
head(data.frame(true_and_predval[,1:2],
                decision =
                  ifelse(true_and_predval$pred_full_20 == 0,
                         "accept", "reject")), 12)
```

```
##    test_set.loan_status pred_full_20 decision
## 1                     0            0   accept
## 2                     0            0   accept
## 18                    1            0   accept
## 26                    0            1   reject
## 27                    0            0   accept
## 28                    0            0   accept
## 30                    0            0   accept
## 32                    0            0   accept
## 34                    0            1   reject
## 35                    0            0   accept
## 36                    0            0   accept
## 37                    1            0   accept
```

We can add an evaluation column.

```
# First 10 accept decisions.
head(data.frame(true_and_predval[,1:2], decision =
                  ifelse(true_and_predval$pred_full_20 == 0, "accept", "reject"),
                evaluation = ifelse(true_and_predval$pred_full_20 == 1,
                                    "it does not matter",
                                    ifelse(true_and_predval$pred_full_20 == 0 &
  true_and_predval$`test_set$loan_status` == 0,
                         "good decision", "bad decision"))), 12)
```

```
##    test_set.loan_status pred_full_20 decision         evaluation
## 1                     0            0   accept      good decision
## 2                     0            0   accept      good decision
## 18                    1            0   accept       bad decision
## 26                    0            1   reject it does not matter
## 27                    0            0   accept      good decision
## 28                    0            0   accept      good decision
## 30                    0            0   accept      good decision
## 32                    0            0   accept      good decision
## 34                    0            1   reject it does not matter
## 35                    0            0   accept      good decision
## 36                    0            0   accept      good decision
## 37                    1            0   accept       bad decision
```

Here we have the first 10 accepted loan applications. A good decision is because we accept the loan that did not default. A bad decision is because we accept the loan application and defaulted. The evaluation does not matter when we reject a loan application.

```
# We accept loans that the model predicts a no-default (0).
# In "accepted_loans" we know whether the accepted loans are in fact
# default or no-default.
accepted_loans <- true_and_predval[pred_full_20 == 0, 1]
# The code above says: if we accept the application, tell me what happened.
head(accepted_loans, 10)
```

```
##  [1] 0 0 1 0 0 0 0 0 0 1
## Levels: 0 1
```

Here we have the first 10 accepted loans. We fail at the third (which is row 18) and the tenth (which is row 37) as the previous table indicates.

```
# bad_rate is the proportion of accepted loans that are in fact default.
bad_rate <- sum(accepted_loans == 1)/length(accepted_loans)
bad_rate
```

```
## [1] 0.0576328
```

This is, by following the model-based rule, we accepted 7,756 loan applications that represent 80% of the total applications. However, 5.76% of those accepted applications were in fact a default. Models are not perfect but we are always interested to find out a good model that leads to a lower bad rate. If we keep the same model and the test set the same, we could reduce this 5.76% by being more strict in the loan application which in simple terms mean to reduce the acceptance rate. This alternative could be controversial as a lower acceptance rate represents lower income (less customers) for the bank or financial firm. In any case, consider we reduce the acceptance rate from 80% to 65% so we can evaluate the impact over the bad rate.

```
# New cutoff value.
cutoff <- quantile(predictions_all_full, 0.65)
# Split the predictions_all_full into a binary variable.
pred_full_35 <- ifelse(predictions_all_full > cutoff, 1, 0)
# A data frame with real and predicted loan status.
true_and_predval <- cbind.data.frame(test_set$loan_status, pred_full_35)
# Loans that we accept given these new rules.
```

```r
accepted_loans <- true_and_predval[pred_full_35 == 0, 1]
# Bad rate (accepted loan applications that are defaults).
bad_rate <- sum(accepted_loans == 1)/length(accepted_loans)
# Show the bad rate.
bad_rate
```

```
## [1] 0.03713107
```

As expected, the bad rate is lower (from 5.76%% to 3.71%). This is, the lower the acceptance rate the lower the bad rate. We can create a function such that given a vector of prediction of loan status we can return the bad rate for different cutoff values. This could be useful to build the bank strategy more easily. This function will reveal the trade-off between the acceptance rate and the bad rate. In particular, the lower the acceptance rate, the lower the income (bad thing) and the lower the bad rate (good thing). So, which combination is the optimal?

1.4 The bank strategy.

A bank could be interested to understand the relationship between the acceptance rate and the bad rate given a model that predicts the loan status.

```r
# Function.
strategy_bank <- function(prob_of_def){
cutoff <- rep(NA, 21)
bad_rate <- rep(NA, 21)
accept_rate <- seq(1, 0, by = -0.05)
for (i in 1:21){
  cutoff[i] <- quantile(prob_of_def, accept_rate[i])
  pred_i <- ifelse(prob_of_def > cutoff[i], 1, 0)
  pred_as_good <- test_set$loan_status[pred_i == 0]
  bad_rate[i] <- sum(pred_as_good == 1)/length(pred_as_good)}
table <- cbind(accept_rate, cutoff = round(cutoff, 4),
               bad_rate = round(bad_rate, 4))
return(list(table = table, bad_rate = bad_rate,
            accept_rate = accept_rate, cutoff = cutoff))
}
```

We can evaluate this function for the `log_model_full` and a bad model like the

log_model_age. In principle, we expect the log_model_full model to have a more attractive relationship between the acceptance rate and the bad rate. This is, lower bad rates for a given acceptance rate. Any financial institution could be interested in increasing the acceptance rate without increasing too much the bad rate.

Let's apply the function to the predictions_all_full and the log_model_age.

```
# Apply the strategy_bank function.
strategy_predictions_all_full <-
  strategy_bank(as.numeric(predictions_all_full))
strategy_pred_log_model <- strategy_bank(as.numeric(pred_log_model_age))
data.frame(accept_rate = strategy_pred_log_model$accept_rate,
           log_model_bad_rate = strategy_pred_log_model$bad_rate,
           full_model_bad_rate = strategy_predictions_all_full$bad_rate)
```

```
##    accept_rate log_model_bad_rate full_model_bad_rate
## 1         1.00         0.10933471         0.109334709
## 2         0.95         0.10849715         0.090662324
## 3         0.90         0.10849715         0.076790831
## 4         0.85         0.10849715         0.066626214
## 5         0.80         0.10547397         0.057632800
## 6         0.75         0.10547397         0.050612020
## 7         0.70         0.10396251         0.043619216
## 8         0.65         0.10396251         0.037131070
## 9         0.60         0.10092961         0.029396596
## 10        0.55         0.10092961         0.023443361
## 11        0.50         0.09933775         0.021039604
## 12        0.45         0.10206865         0.018565207
## 13        0.40         0.10206865         0.015471893
## 14        0.35         0.09907039         0.011788977
## 15        0.30         0.09972041         0.009281540
## 16        0.25         0.10343554         0.005363036
## 17        0.20         0.09722922         0.003610108
## 18        0.15         0.09360190         0.001374570
## 19        0.10         0.10125362         0.000000000
## 20        0.05         0.10516605         0.000000000
## 21        0.00         0.00000000         0.000000000
```

The full model is superior because at any acceptance rate we can reach a lower bad rate. A plot can reveal the main differences of these two models: `log_model_full` and `log_model_age`.

```
# Plot the strategy functions
par(mfrow = c(1, 2))
plot(strategy_predictions_all_full$accept_rate,
     strategy_predictions_all_full$bad_rate,
     type = "l", xlab = "Acceptance rate", ylab = "Bad rate",
     lwd = 2, main = "log_model_full")
abline(v = strategy_predictions_all_full[["accept_rate"]][8], lty = 2)
abline(h = strategy_predictions_all_full[["bad_rate"]][8], lty = 2)
abline(v = strategy_predictions_all_full[["accept_rate"]][5], lty = 2,
       col = "red")
abline(h = strategy_predictions_all_full[["bad_rate"]][5], lty = 2,
       col = "red")
plot(strategy_pred_log_model$accept_rate,
     strategy_pred_log_model$bad_rate,
     type = "l", xlab = "Acceptance rate",
     ylab = "Bad rate", lwd = 2, main = "log_model_age")
abline(v = strategy_pred_log_model[["accept_rate"]][8], lty = 2)
abline(h = strategy_pred_log_model[["bad_rate"]][8], lty = 2)
abline(v = strategy_pred_log_model[["accept_rate"]][5], lty = 2, col = "red")
abline(h = strategy_pred_log_model[["bad_rate"]][5], lty = 2, col = "red")
```

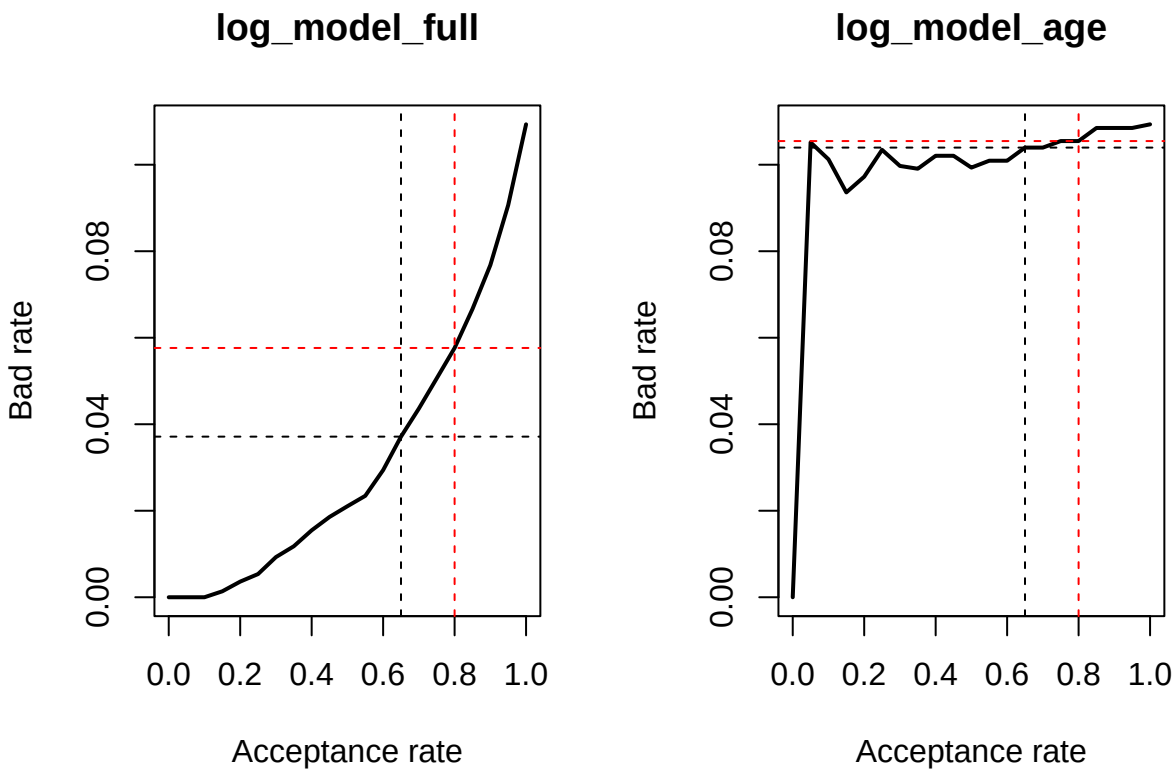


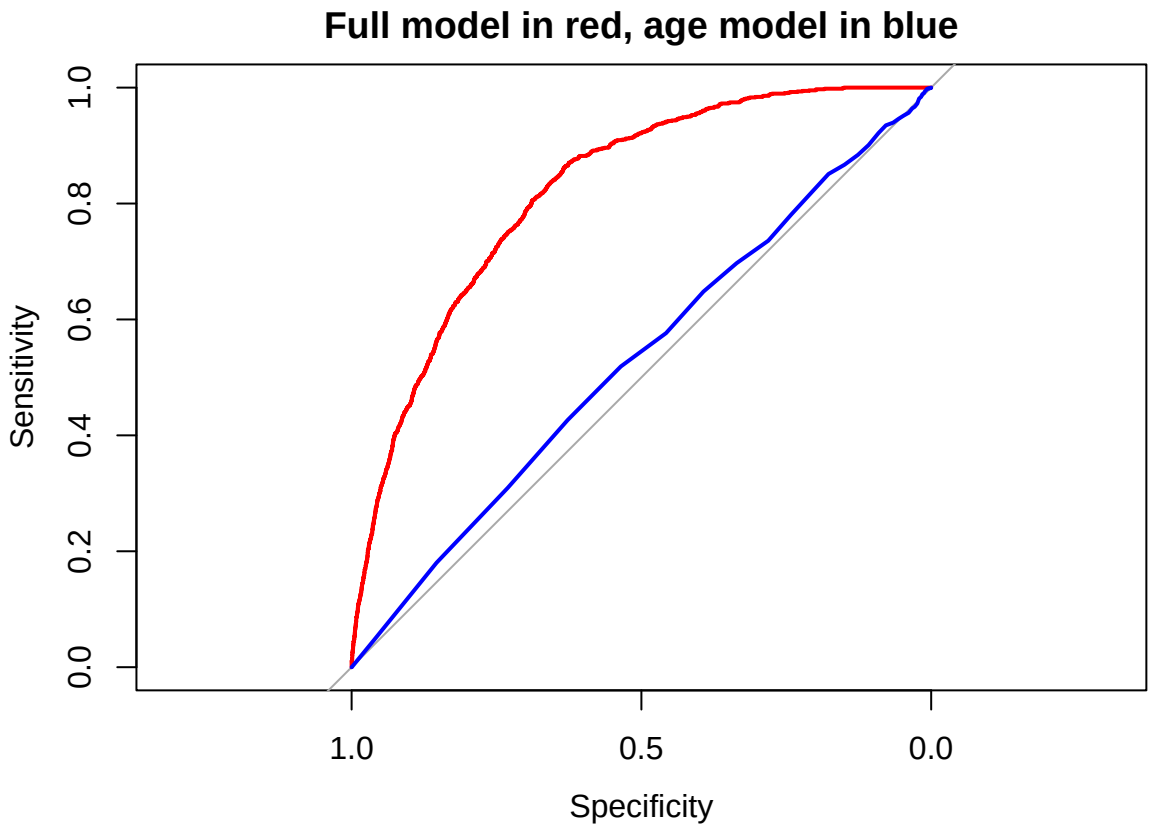
Figure 1.4.1: A good and a bad model.

The `log_model_full` model is better because for any acceptance rate we can reach a lower bad rate compared with the `log_model_age`. This is because the `log_model_full` model can identify defaults and no defaults with higher precision compared with the `log_model_age`. The value of a good model is that it can help us to make better business decisions, in this case better credit evaluation decisions.

The model ability to predict defaults and no defaults can be measured by the AUC. The AUC can be defined as the probability that the fit model will score a randomly drawn positive sample higher than a randomly drawn negative sample. AUC stands for area under the curve in the following context:

```
ROC_all_full <- roc(test_set$loan_status, predictions_all_full)
ROC_log_model <- roc(test_set$loan_status, pred_log_model_age)
# Draw the ROCs on one plot
plot(ROC_all_full, col = "red", main = "Full model in red, age model in blue")
```

```
lines(ROC_log_model, col = "blue")
```



Sensitivity is the model ability to correctly identify defaults, these are known as true positive. Specificity is the model ability to correctly identify no-default loans, these are known as true negative.

As expected, the area under the curve (AUC) is higher for the red line which corresponds to the `log_model_full` model. We can calculate the exact values:

```
# Compute the AUCs
auc(ROC_all_full)
```

```
## Area under the curve: 0.8213
```

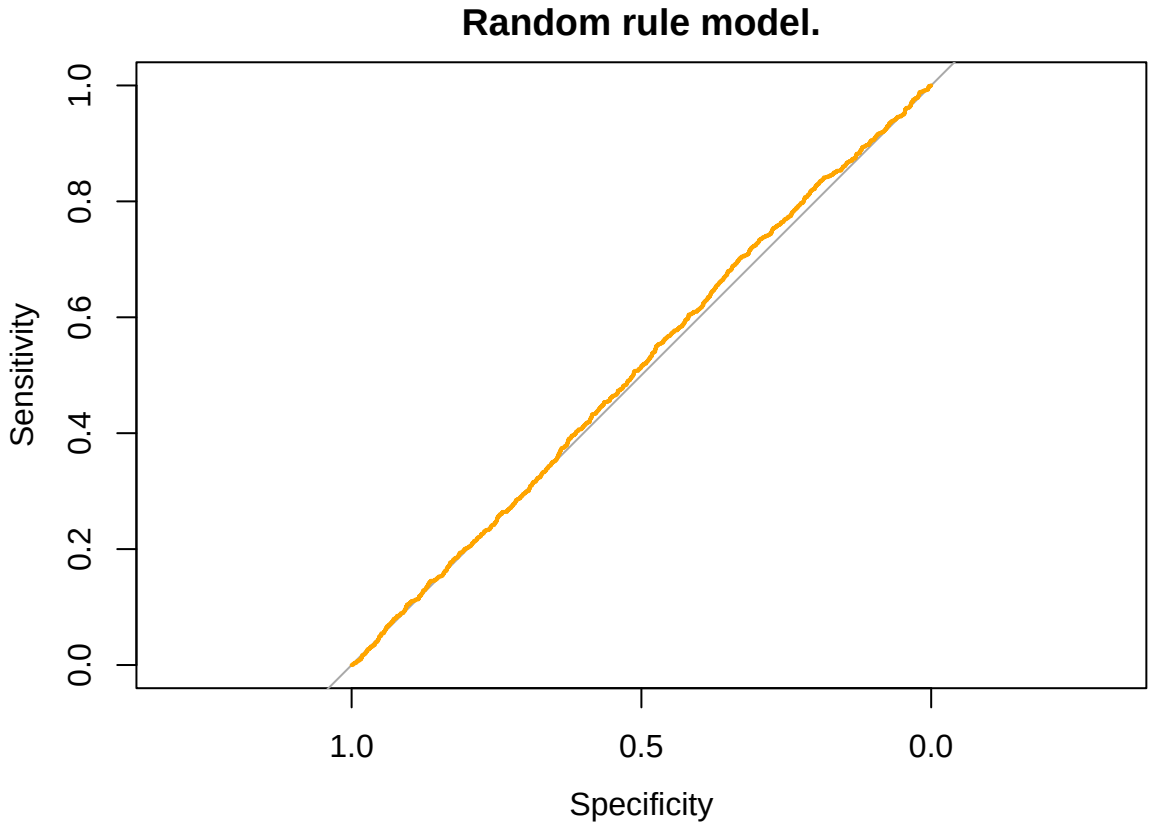
```
auc(ROC_log_model)
```

```
## Area under the curve: 0.5301
```

Note that the `log_model_age` has an AUC of 0.5301. This is close to a loan approval process in which we randomly accept and reject with no further analysis. In other words, the `log_model_age` is so bad that it is almost equivalent as using no model at all and accept

and reject loan applications based on a random rule. A pure-random approval rule would look like this:

```
set.seed(2020)
pred_rand_model <- runif(length(pred_log_model_age))
ROC_rand <- roc(test_set$loan_status, pred_rand_model)
# Draw the ROCs on one plot
plot(ROC_rand, col = "orange", main = "Random rule model.")
```

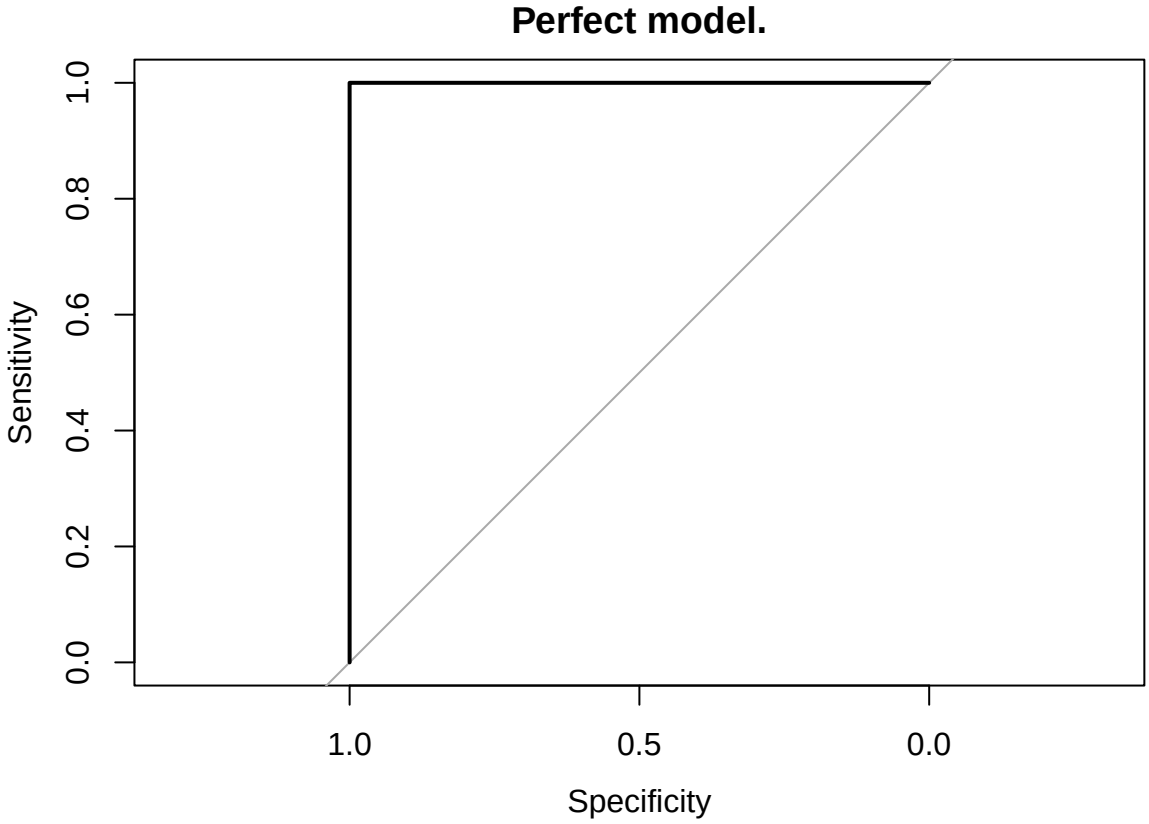



```
auc(ROC_rand)
```

```
## Area under the curve: 0.5105
```

In theory, this random evaluation process would lead to an AUC of $0.5 = (1 \times 1)/2$. In contrast, now imagine we have a perfect model:

```
pred_perfect_model <- as.numeric(test_set$loan_status)
ROC_perfect <- roc(test_set$loan_status, pred_perfect_model)
plot(ROC_perfect, main = "Perfect model.")
```




```
auc(ROC_perfect)
```

```
## Area under the curve: 1
```

In a perfect model, the AUC is equal to 1 (1×1). The model correctly identify defaults (100% sensitivity) and at the same time the model correctly identify no-defaults (100% specificity)

2 The Merton model.

The Merton model is useful when we are interested to evaluate the credit risk of public firms. In short, it evaluates how likely is that the value of the firm's assets fall in the future below a certain threshold represented by the firm's debt. By doing that, the model is able to estimate a probability of default among other results. The model can be used as a tool to propose changes (for example) in the balance sheet to reduce the credit risk exposure of a firm.

2.1 Minimize a function in R.

The estimation of the credit risk Merton's model requires a minimization of a function. In this section, we show a simple minimization example. Here are some useful online resources for further references.

- FRM: How d2 in Black-Scholes becomes PD in Merton model.
- Normal Probability Distribution Graph. Interactive.
- How to use optim in R.

The objective function to be minimized is $f(a, b) = \sum(a + bx_i - y_i)^2$. We have six values for x and y, and we are expected to minimize the function f by finding the parameter values a and b. The x_i and y_i data looks like this:

```
# Data.
dat <- data.frame(x = c(1, 2, 3, 4, 5, 6), y = c(1, 3, 5, 6, 8, 12))
kable(dat, caption = "Observations.", row.names = TRUE) # The table.
```

Table 2.1.1: Observations.

	x	y
1	1	1
2	2	3
3	3	5
4	4	6
5	5	8
6	6	12

Graphically:

```r
# A scatter plot.
plot(y ~ x, data = dat, pch = 19, cex = 2)
```

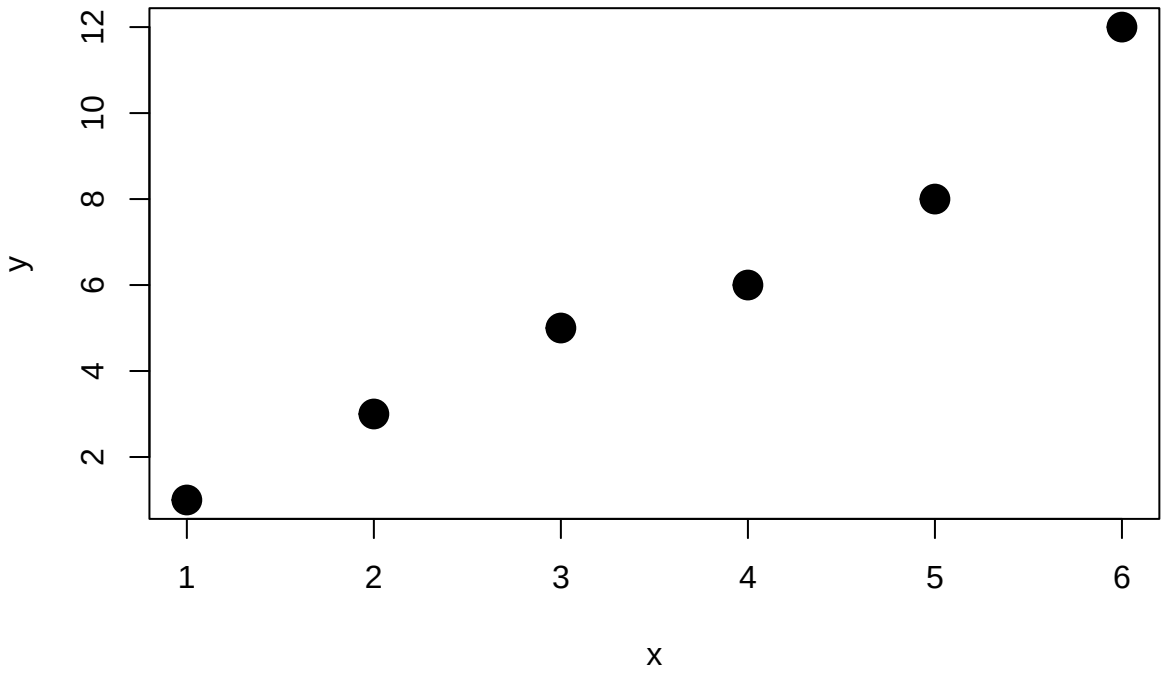


Figure 2.1.1: Original data.

Let's implement the minimization.

```r
# Function that calculates the residual sum of squares.
min.RSS <- function(data, par) {
  with(data, sum((par[1] + par[2] * x - y)^2)) }
# Function minimization.
result <- optim(par = c(0, 1), min.RSS, data = dat)
a <- result$par[1]
b <- result$par[2]
f <- sum((a + b * dat$x - dat$y)^2)
# Minimization output.
ans <- data.frame(a, b, f)
```

```
kable(ans, caption = "Minimization results f(a,b).")
```

Table 2.1.2: Minimization results f(a,b).

a	b	f
-1.266846	2.02862	2.819048

The minimization result is represented by a line, and the line is characterized by an intercept equal to -1.266846 and a slope equal to 2.02862.

We can plot it and verify that this is indeed the right answer.

```
# View the results.
plot(y ~ x, data = dat, pch = 19, ylim = c(-2, 12), xlim = c(0, 6), cex = 2)
abline(a, b, col = "red", lwd = 3)
abline(h = 0, lty = 2)
abline(v = 0, lty = 2)
```

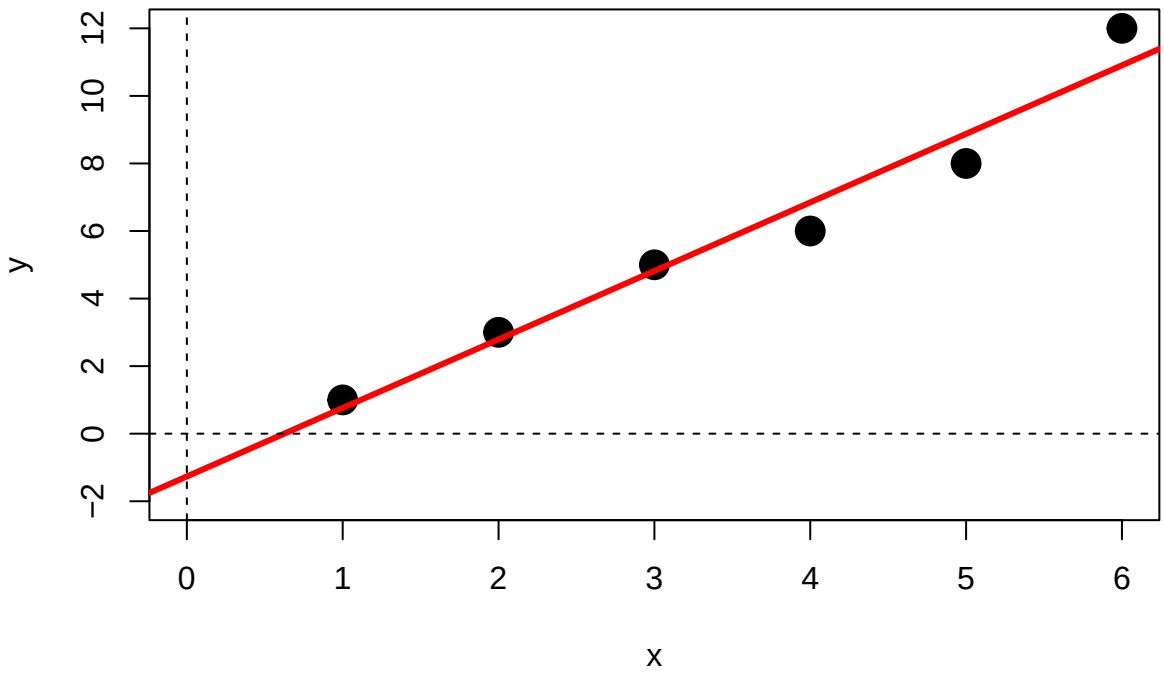


Figure 2.1.2: Line that minimize the residual sum of squares.

The function that we minimize here is basically the OLS criterion. This is why the regression model leads to the same results.

```
# The previous example is equivalent as a linear regression model.
reg <- lm(y ~ x, data = dat)
# See the results.
summary(reg)
```

```
##
## Call:
## lm(formula = y ~ x, data = dat)
##
## Residuals:
##       1       2       3       4       5       6
##  0.2381  0.2095  0.1810 -0.8476 -0.8762  1.0952
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.2667     0.7815  -1.621 0.180388
## x             2.0286     0.2007  10.109 0.000539 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8395 on 4 degrees of freedom
## Multiple R-squared:  0.9623, Adjusted R-squared:  0.9529
## F-statistic: 102.2 on 1 and 4 DF,  p-value: 0.000539
```

The function that we minimize in the estimation of the credit risk Merton model is more elaborated but we follow the same principle as above.

2.2 Applied example.

Let's follow the example 24.3 in [Hull, 2015]. Five parameters: $E_0 = 3$, $\sigma_E = 0.8$, $rf = 0.05$, $T = 1$, $D = 10$ lead to an estimate of the value of the assets today V_0 and the volatility of the assets σ_V. With these seven parameters we can estimate the probability of default (pd) among other results.

```
# List of known parameters.
E0 <- 3 # Value of the equity as today, I recommend market capitalization.
se <- 0.8 # Stock returns volatility.
rf <- 0.05 # Risk-free rate.
TT <- 1 # Maturity.
D <- 10 # Value of the debt. The Bloomberg function DDIS is useful.
# We need equations 24.3 and 24.4 to estimate V0 and sv.
eq24.3 <- function(V0, sv) {
  ((V0 * pnorm((log(V0 / D) + (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) -
  D * exp(-rf * TT) * pnorm(((log(V0 / D) +
  (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) - sv * sqrt(TT)) - E0)) }
eq24.4 <- function(V0, sv) {
  ((pnorm((log(V0 / D) + (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) *
      sv * V0 - se * E0)) }
# Footnote 10 indicates that we should minimize F(x,y)^2+G(x,y)^2
min.footnote10 <- function(x)
```

```
  (eq24.3(x[1], x[2]))^2 + (eq24.4(x[1], x[2]))^2
# The minimization leads to the values of V0 and sv.
V0_sv <- optim(c(1, 1), min.footnote10)
# Define the values as parameters.
V0 <- V0_sv$par[1]
sv <- V0_sv$par[2]
# Calculate the probability of default as a function.
PD <- function(V0, sv) {
  pnorm(-(((log(V0 / D) + (rf + sv^2 / 2) * TT) /
             (sv * sqrt(TT))) - sv * sqrt(TT))) }
# Calculate the probability of default given the previous parameters.
pd <- PD(V0, sv)
# Gather the results.
ans <- data.frame(pd, V0, sv)
kable(ans, caption = "Main model results.")
```

Table 2.2.1: Main model results.

pd	V0	sv
0.1269396	12.39578	0.2123041

Let's analyze the role of the values of d_1, d_2, $N(d_1)$ and $N(d_2)$. First we isolate these values.

```
# Inspect the role of d and N(d).
d1 <- (log(V0 / D) + (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))
d2 <- d1 - sv * sqrt(TT)
Nd1 <- pnorm(d1)
Nd2 <- pnorm(d2)
ans <- data.frame(d1, Nd1, d2, Nd2)
kable(ans, caption = "Inspect the role of d and N(d).")
```

Table 2.2.2: Inspect the role of d and N(d).

d1	Nd1	d2	Nd2
1.353282	0.9120172	1.140978	0.8730604

Now let's illustrate how $N(-d_2)$ lead to the probability of default given the properties of a standard normal distribution.

```
xseq <- seq(-4, 4, 0.01)
densities <- dnorm(xseq, 0, 1) # Standard normal distribution.
# Graphical evaluation of N(-d_2).
plot(xseq, densities, xlab = "d values (d_2 is dotted line)",
     ylab = "Density", type = "l", lwd = 2, cex = 2)
abline(h = 0)
polygon(c(xseq[xseq >= d2], d2), c(densities[xseq >= d2], 0), col = "red")
polygon(c(xseq[xseq < d2], d2), c(densities[xseq < d2], 0), col = "blue")
legend("topleft", legend = c("N(d_2)=0.8730604
is represented in blue and
N(-d_2)=0.1269396 in red.

N(d_2)+N(-d_2)=1"),
bg = "white", bty = "n", cex = 0.9)
abline(v = d2, lty = 2, lwd = 2)
```

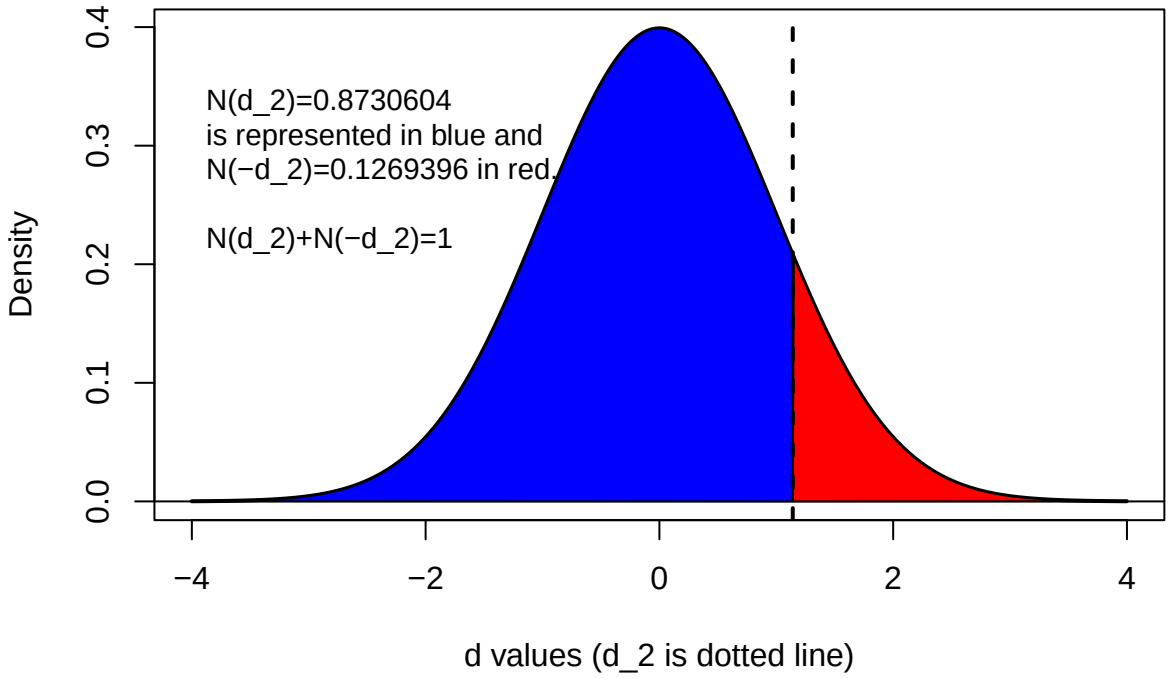


Figure 2.2.1: The red area represents the probability of default.

Now let's analyze the minimization that leads to V_0 and σ_V. We do not have an analytic math expressions (closed form) to calculate V_0 and σ_V, instead we have approximate values or estimates given the minimization of equations 24.3 and 24.4. In order to see how good this approximation is, we can retrieve the value of $[F(x,y)]^2 + [G(x,y)]^2$ evaluated at $x = V_0$ and $y = \sigma_v$. This value is supposed to be positive as both $F(x,y)$ and $G(x,y)$ functions are squared, so we expect a positive number close to zero when evaluated at $x = V_0$ and $y = \sigma_v$.

```
# Evaluate how good is the minimization.
min.footnote10(x = c(V0, sv))
```

```
## [1] 1.425797e-07
```

The minimization conducted in the function `min.footnote10` above worked fairly well.

There is another way to approach how the minimization work. We can propose a graphical analysis. In particular, we can verify whether $V_0 = 12.39578$ and $\sigma_V = 0.2123041$ minimizes

the objective function. To illustrate this in two axis, we need to fix either V_0 or σ_V, and evaluate $[F(x,y)]^2 + [G(x,y)]^2$ with different values of V_0 or σ_V. Let's do that.

```
# Fix sv and evaluate different values of V0.
V0.seq <- seq(from = 4, to = 20, length.out=50)
sv.rep <- rep(sv, 50)
# Function to be evaluated.
FG <- function(V0, sv) { (eq24.3(V0, sv))^2 + (eq24.4(V0, sv))^2 }
# Apply the function with fixed sv and different V0 values.
FG.V0 <- mapply(FG, V0.seq, sv.rep)
```

```
# Plot the results.
plot(V0.seq, FG.V0, type = "l", xlab = expression(paste(V[0])),
     ylab = expression(paste(F(x,y)^2+G(x,y)^2)), cex.lab = 0.8, lwd = 3)
abline(v = V0, lty = 2)
abline(h = FG(V0, sv), lty = 2)
points(V0, FG(V0, sv), pch = 1, cex = 3, col = "red", lwd = 2)
```

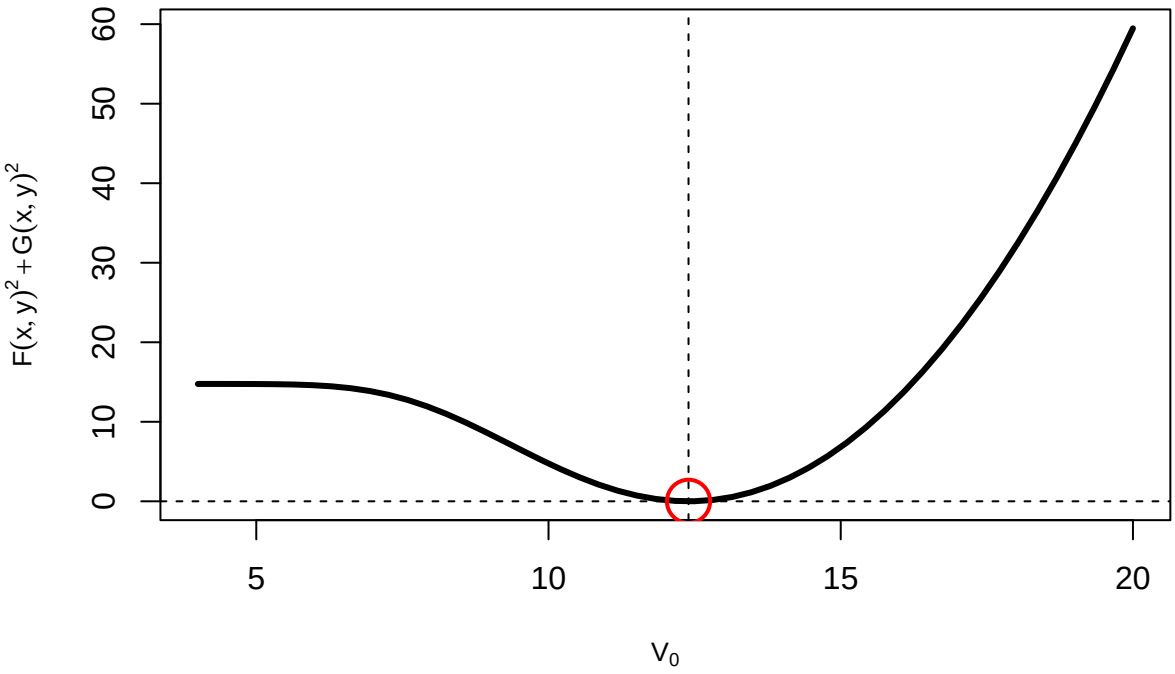


Figure 2.2.2: Here we fix the volatility of the assets and change the value of the assets at time zero. The minimization looks fine.

Clearly the minimization worked well here. For the sake of completeness, now we fix the V_0 value and evaluate the function again.

```
# Now fix V0, and evaluate different sv values.
sv.seq <- seq(0, 0.6, length.out = 50)
V0.rep <- rep(V0, 50)
# Evaluate the function with these parameters.
FG.sv <- mapply(FG, V0.rep, sv.seq)
# Plot the results.
plot(sv.seq, FG.sv, type = "l", xlab = expression(paste(sigma[V])),
     ylab = expression(paste(F(x, y)^2 + G(x, y)^2)),
     cex.lab = 0.8, lwd = 3)
abline(v = sv, lty = 2)
abline(h = FG(V0, sv), lty = 2)
```

```
points(sv, FG(V0, sv), pch = 1, cex = 3, col = "red", lwd = 2)
```

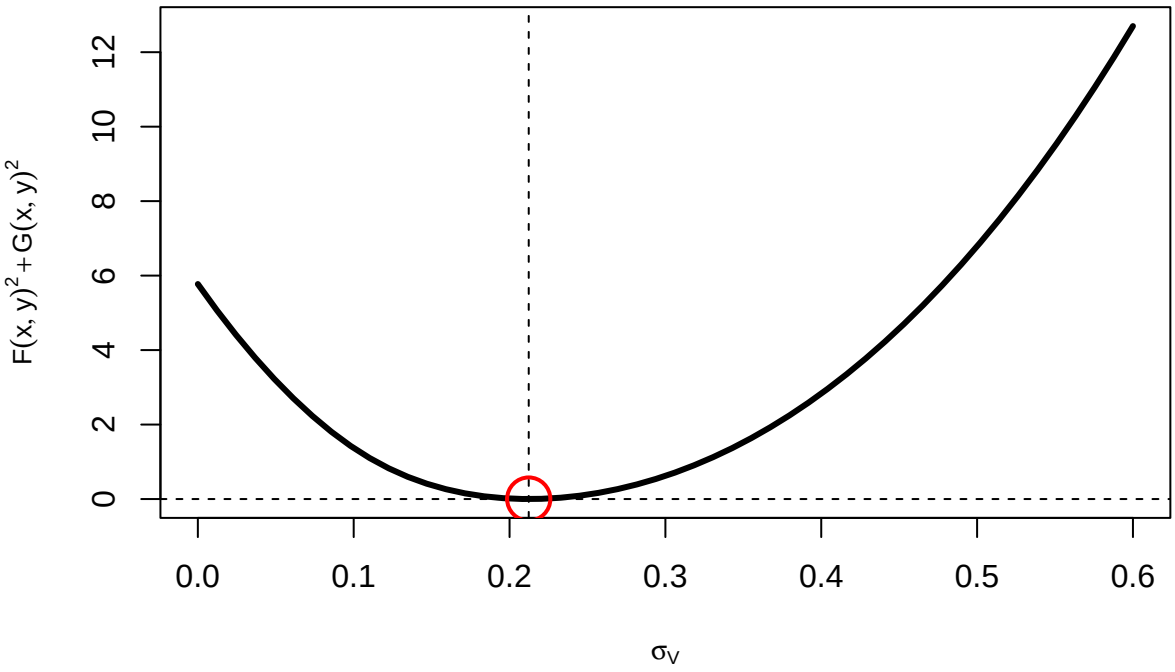


Figure 2.2.3: Here we fix the value of the assets at time zero and change the volatility of the assets. The minimization looks fine.

We can show the same as before but in a 3D plot.

```
# outer evaluates the function for each combination of the vectors.
z <- outer(V0.seq, sv.seq, FG) # z is now 50x50 matrix.
persp(V0.seq, sv.seq, z, col = "springgreen", shade = 0, xlab = "V0",
      ylab = "sv", zlab = "Probability of default")
```

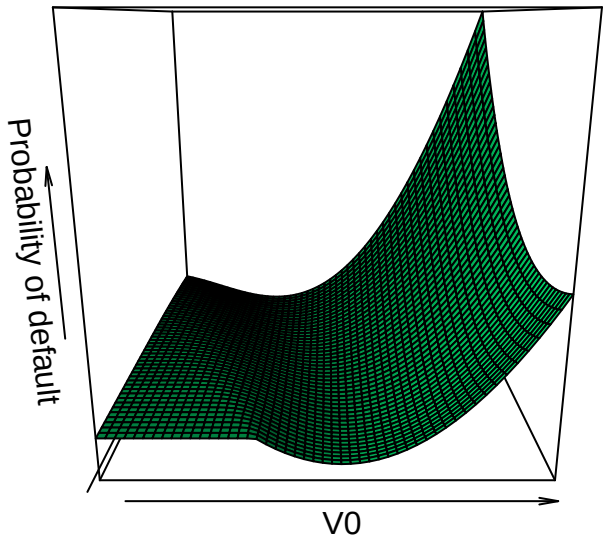


Figure 2.2.4: Minimization of the function.

This green plot is nice but it is not interactive. We could plot interactive three-dimension plots with other R packages like `plotly` and show them in a html context or java environment. Given that this is a PDF document, we cannot show dynamic plots (at least not yet), so we propose to use contour plots. Contour plots or level plots are a way to show a three-dimensional surface on a two-dimensional plane.

```
# The nlevels is high to emphasize the minimum value in blue.
contour(V0.seq, sv.seq, z, xlab = "V0", ylab = "sv", lwd = 2, nlevels = 300)
points(V0, sv, pch = 19, col = "blue", cex = 2)
```

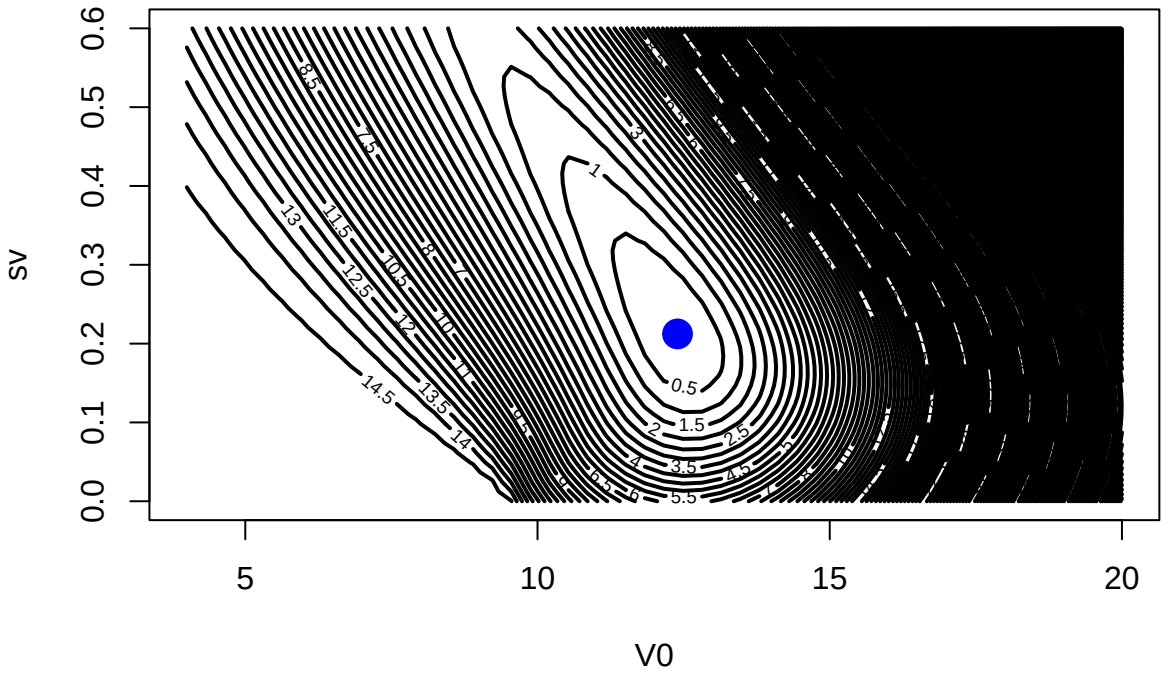


Figure 2.2.5: Minimization of the function: a contour view.

The following plot cannot be seen in a PDF output, but it can be seen in a browser as an interactive plot.

```
library(plotly)
p <- plot_ly(x = V0.seq, y = sv.seq, z = z, type = "surface") %>%
layout(
  title = "Minimization of f at V0=12.395 an sv=0.212",
  scene = list(xaxis = list(title = "V0"), yaxis = list(title = "sv"),
    zaxis = list(title = "f=F^2+G^2")))
add_markers(p, x = 12.3957765, y = 0.2123041, z = 0, inherit = TRUE)
```

The Merton model allows us to calculate more values in this credit risk assessment.

```
# Other results in the Merton's model.
market_value_debt <- V0 - E0
```

```
pv_promised_payment_debt <- D * exp(-rf * TT)
Expected_Loss <- (pv_promised_payment_debt - market_value_debt) /
  pv_promised_payment_debt
recovery_rate <- 1 - (Expected_Loss / pd)
ans <- data.frame(market_value_debt, pv_promised_payment_debt,
                  Expected_Loss, recovery_rate, pd)
kable(t(ans), caption = "Other results in the Merton's model.")
```

Table 2.2.3: Other results in the Merton's model.

market_value_debt	9.3957765
pv_promised_payment_debt	9.5122942
Expected_Loss	0.0122492
recovery_rate	0.9035039
pd	0.1269396

Could we figure out the implied bond yield spread? It would be similar to the fair interest rate that this firm has to pay given its probability of default and recovery rate. Take Hull's equation 24.2 as a reference: $\lambda(T) = S(T)/(1-R)$ and solve for $S(T)$: $S(T) = \lambda(T) \times (1-R)$, so $S(T) = (0.12693963) \times (1 - 0.90350393)$. This is 122.4918 basis points more than the risk-free rate. Then, with five parameters we can calculate (among other things) the firm's probability of default and the correspondent yield spread of this firm's bond. It would be interesting to compare this value with the market yield spread and evaluate whether the market yield is over or underestimated according to our theoretical value.

```
spread <- function(E0, se, rf, TT, D) {
  eq24.3 <- function(V0, sv) {
  ((V0 * pnorm((log(V0 / D) + (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) -
  D * exp(-rf * TT) * pnorm(((log(V0 / D) +
  (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) - sv * sqrt(TT)) - E0)) }
eq24.4 <- function(V0, sv) {
  ((pnorm((log(V0 / D) + (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) *
      sv * V0 - se * E0)) }
  min.footnote10 <- function(x)
  (eq24.3(x[1], x[2]))^2 + (eq24.4(x[1], x[2]))^2
```

```
V0_sv <- optim(c(1, 1), min.footnote10)
V0 <- V0_sv$par[1]
sv <- V0_sv$par[2]
pd <- pnorm(-(((log(V0 / D) + (rf + sv^2 / 2) * TT) /
          (sv * sqrt(TT))) - sv * sqrt(TT)))

market_value_debt <- V0 - E0
pv_promised_payment_debt <- D * exp(-rf * TT)
Expected_Loss <- (pv_promised_payment_debt - market_value_debt) /
  pv_promised_payment_debt
recovery_rate <- 1 - (Expected_Loss / pd)
spread <- pd * (1-recovery_rate)
spread
}
```

```
spread(E0 = 3, se = 0.8, rf = 0.05, TT = 1, D = 10)
```

```
## [1] 0.01224917
```

```
x.T <- seq(0.5, 10, 0.5)
spread.T <- mapply(spread, E0 = 3, se = 0.8, rf = 0.05, x.T, D = 10)
```

```
data.frame(x.T, spread.T)
```

```
##     x.T    spread.T
## 1   0.5 0.001436993
## 2   1.0 0.012249175
## 3   1.5 0.034639827
## 4   2.0 0.067649103
## 5   2.5 0.109582247
## 6   3.0 0.158117325
## 7   3.5 0.210872346
## 8   4.0 0.265378322
## 9   4.5 0.319998224
## 10  5.0 0.373117588
## 11  5.5 0.423629820
## 12  6.0 0.470524979
## 13  6.5 0.514157658
```

```
## 14  7.0 0.553779756
## 15  7.5 0.590028475
## 16  8.0 0.622657593
## 17  8.5 0.652347129
## 18  9.0 0.678931474
## 19  9.5 0.702941509
## 20 10.0 0.724762411
```

2.3 Inside view of the Merton's model.

The model assumes that the assets follow a geometric Brownian stochastic process. Let's assume the daily evolution of the market value of the firm assets over a year. Here are 10 simulated paths of V_t from $t = t, ..., T$. The reference of this section is Chapter 14 and 15 (John C. Hull).

```
# Merton assumes V follows a geometric brownian motion process.
V0.sim <- function(i) {  # the argument i is not used here.
  GBM(N = 365, T = 1, t0 = 0, x0 = V0, theta = 0.05, sigma = sv)
  }
set.seed(3) # Reproducibility
paths <- sapply(1:10, V0.sim) # Create 10 paths for V.
# Plot results.
matplot(paths, type ="l", col = "black", lwd = 1, lty = 1,
        ylab = expression(paste(V[t])), xlab = "Time in days (0 to T)")
abline(h = D, col = "red", lwd = 2)
points(1, V0, pch = 19, cex = 1.5, col = "blue")
```

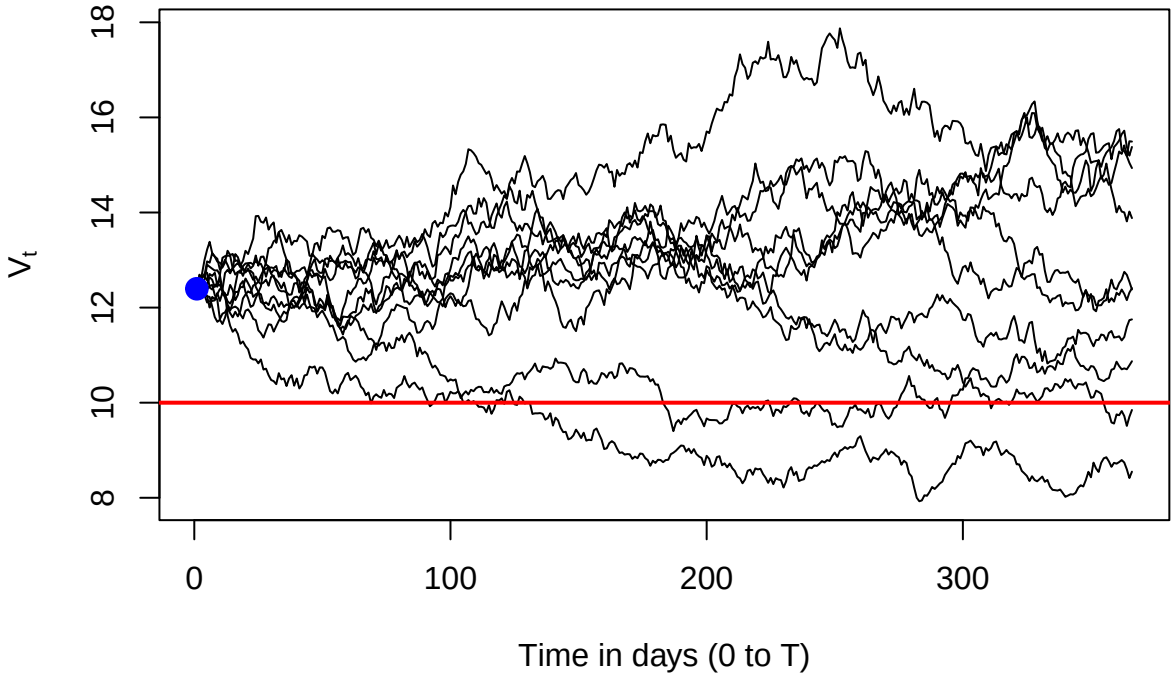


Figure 2.3.1: Simulation of 10 paths of the assets.

These simulations can be interpreted as the evolution of the value of the firm's assets in 10 parallel universes. In the past, some students have called this figure *la grafica de los pelitos*, this is OK as long as you understand the figure implications. The right name is 10 simulated geometric brownian processes. Let's count the cases in which $V_t < D$ note that here I use t and not T, so we are counting the cases in which at least at some time t the value of the assets were lower than 10.

```
# Which path went lower than D?
which(colSums(paths < 10) > 0)
```

```
## [1] 2 4 9
```

The value of the assets was lower than $10 at some time in the second, fourth and ninth alternate universes. Let's plot these cases.

```r
# There might be an easier way to select these cases.
matplot(paths[,which(colSums(paths < 10) > 0)], type = "l",
        col = c("green", "blue", "red"), lwd = 2, lty = 1,
        ylab = expression(paste(V[t])), xlab = "Time in days (0 to T)")
abline(h = D, col = "red", lwd = 2)
points(1, V0, pch = 19, cex = 1.5, col = "blue")
```

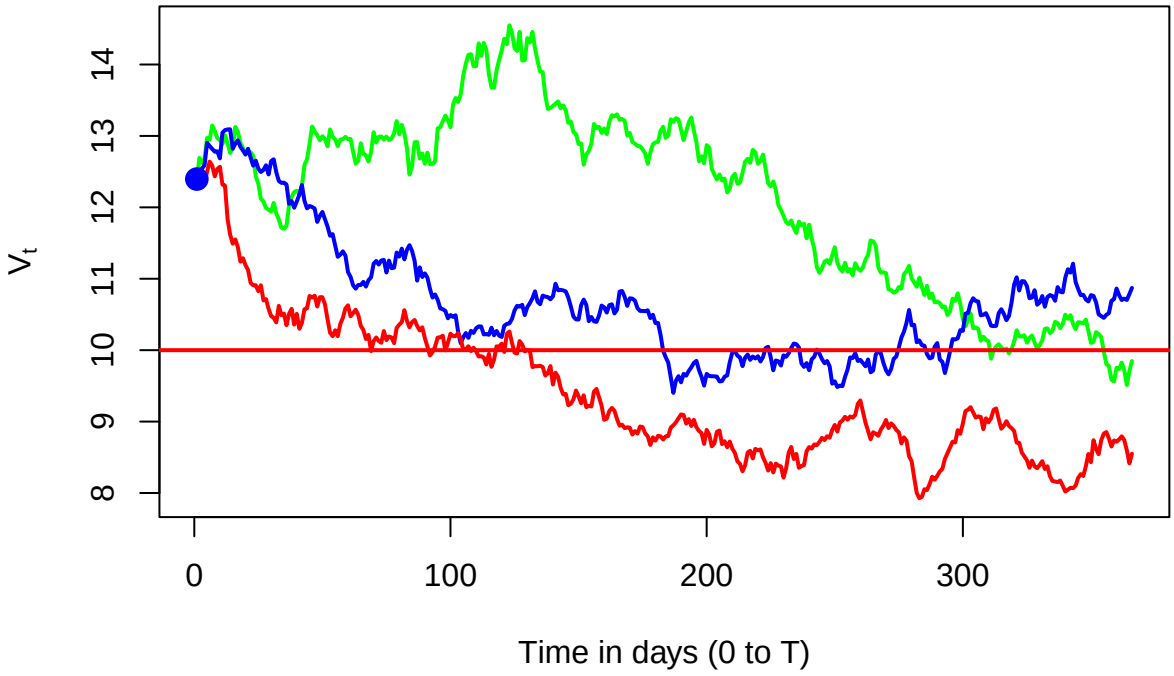


Figure 2.3.2: Note that the blue path finally did not default. Recovery is possible as in the real life.

The blue path was lower than 10 at some time, but at $t = T$ it is clearly higher than 10. For this reason, the blue path does not represent a firm's default. See how the assets are not high enough to pay the debt at maturity only in two cases: green and red. Thus, according to this simulation the probability of default is 20%. Here is how we can calculate the cases in which $V_T < D$.

```r
sum(paths[366,] < 10) / 10
```

```
## [1] 0.2
```

The probability of default of 20% can be disappointing because it is significantly higher than 0.12693963. This is because the number of simulations is small. The following examples show how these values tend to converge as we increase the number of simulations from 100 to 100,000.

Here are 100 simulated paths of V_0 to V_T.

```r
set.seed(3) # Reproducibility
paths <- sapply(1:100, V0.sim) # Create 100 paths.
# Plot results.
matplot(paths, type = "l", col = "black", lwd = 1, lty = 1,
        ylab = expression(paste(V[t])), xlab = "Time in days (0 to T)")
abline(h = D, col = "red", lwd = 2)
points(1, V0, pch = 19, cex = 1.5, col = "blue")
```

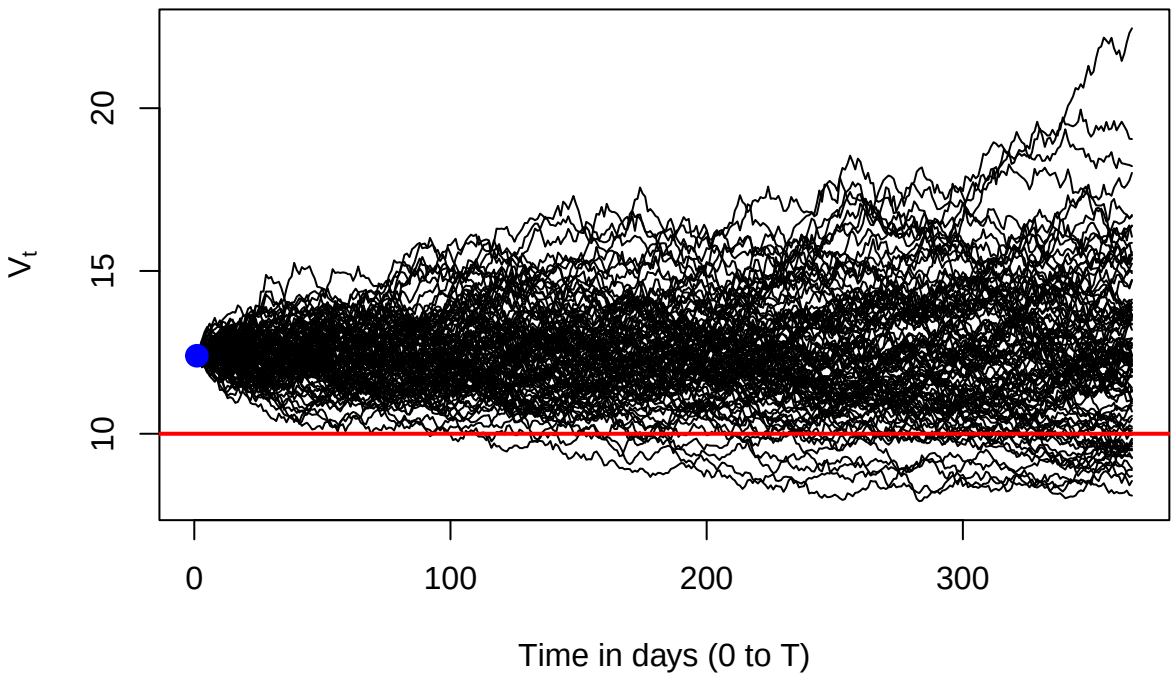


Figure 2.3.3: Simulation of 100 paths of the assets.

Given the simulation above, the probability of default is 17%. This is closer to 12.693963%.

```
sum(paths[366,] < 10) / 100
```

```
## [1] 0.17
```

Let's see all the 17 cases that end in default.

```
# These are the 17 default cases.
matplot(paths[,which(paths[366,] < 10)], type = "l", col = "black",
        lwd = 1, lty = 1, ylab = expression(paste(V[t])),
        xlab = "Time in days (0 to T)")
abline(h = D, col = "red", lwd = 2)
points(1, V0, pch = 19, cex = 1.5, col = "blue")
```

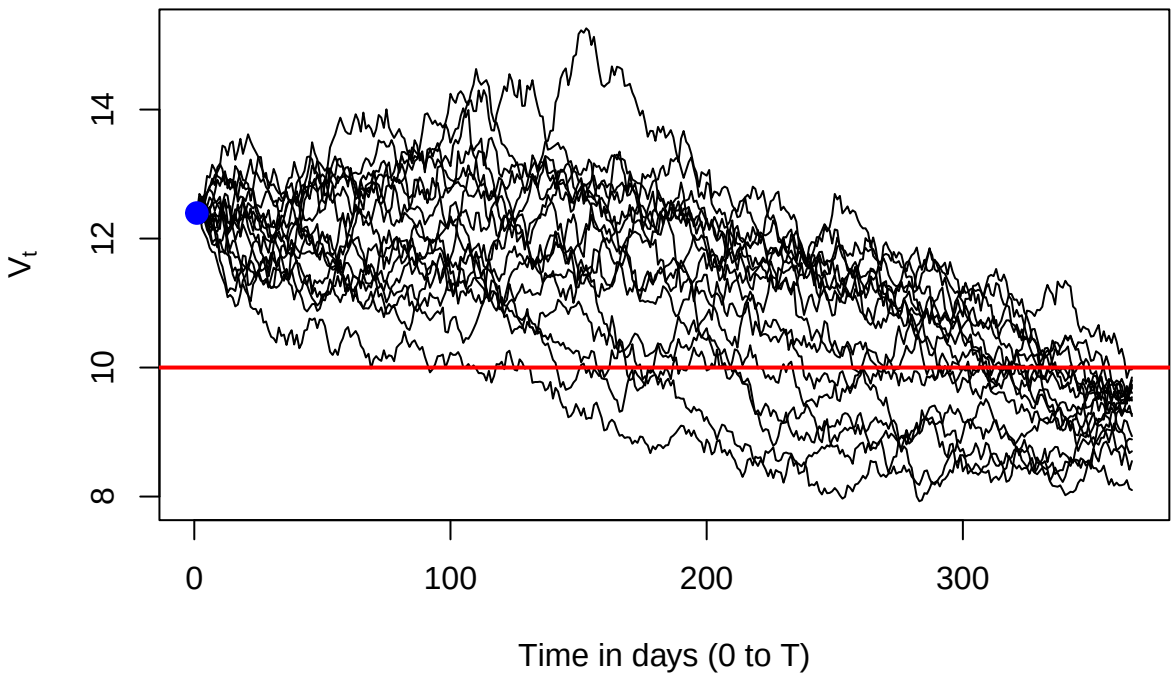


Figure 2.3.4: Default cases.

Now, let's see the 11 cases in which V_t eventually went lower than \$10 but at the end did not default.

```
# V_t < 10
almost_default <- which(colSums(paths < 10) > 0)
# V_T < 10
default <- which((paths[366,] < 10))
# There should be an easier way to do this:
matplot(paths[,almost_default[which(almost_default %in% default == FALSE)]],
        type = "l", col = c(1:11), lwd = 1, lty = 1,
        ylab = expression(paste(V[t])), xlab = "Time in days (0 to T)")
abline(h = D, col = "red", lwd = 2)
points(1, V0, pch = 19, cex = 1.5, col = "blue")
```

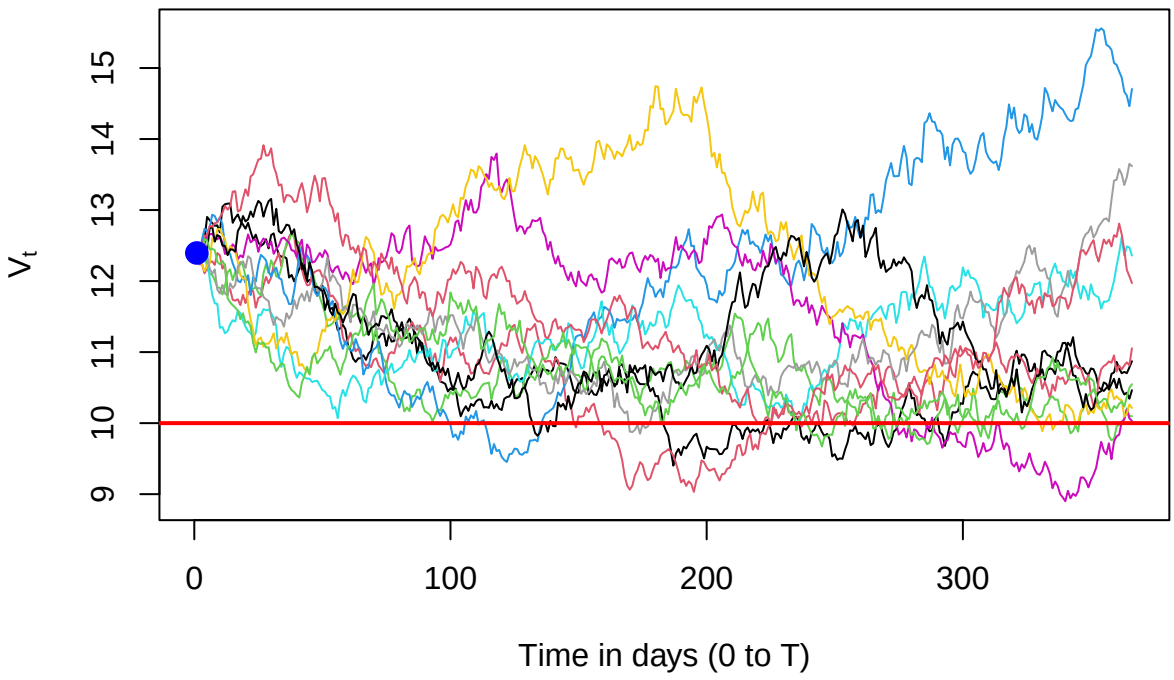
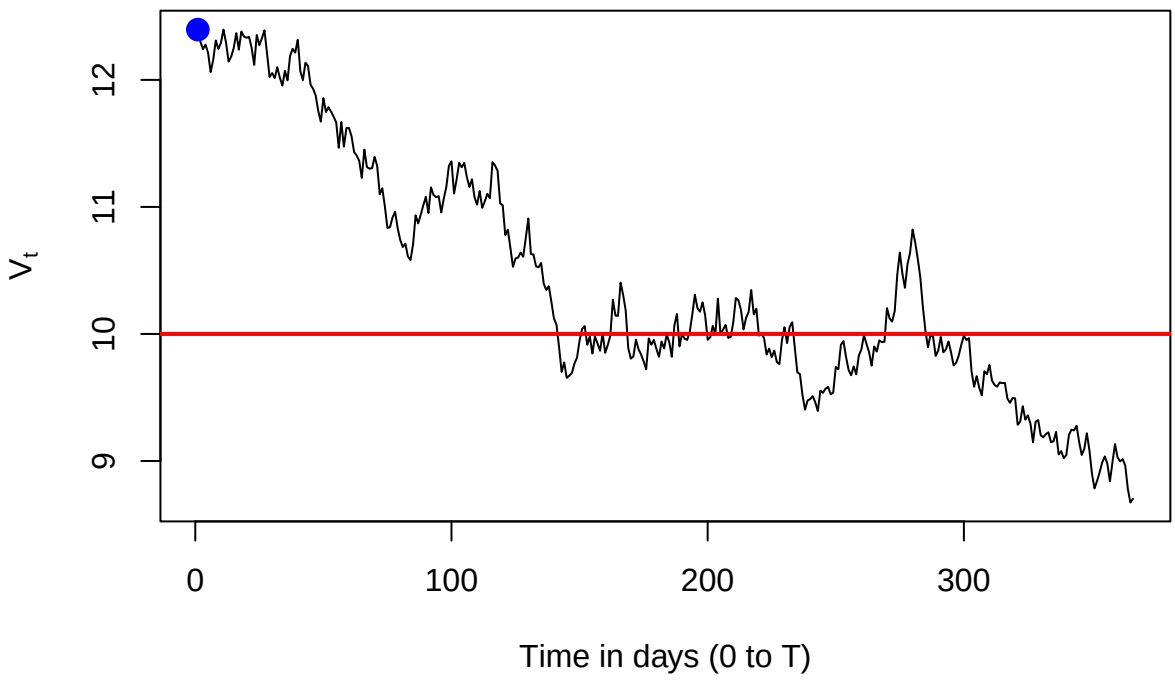


Figure 2.3.5: All these paths went lower than 10 at some point, but at the end the value of the assets are higher than 10. These are paralell universes in which the firm finally survived after some drama.

What if the asset value is decreasing over time $V_t \leq V_0$? Just as in the case of a physical assets that depreciates over time. What if we are interested in evaluating $V_t < D$ at any time, not only at maturity? Ideally, we should consider an alternative to the geometric brownian motion stochastic process in order to conduct our simulation analysis. However, we could consider a shortcut. This is, take the same GBM stochastic process and filter those paths which are $V_t \leq V_0$. There are some minor pitfalls here. For example, we still have some paths which show a decrease and then an increase. Those cases can be interpreted as if the depreciation is lower than an asset value increase. In any case, our approach delivers paths which never lead to an asset value higher than V_0.

```
set.seed(3) # Reproducibility
paths <- sapply(1:100, V0.sim) # Create 100 paths.
# Select those paths with all 366 values lower or equal than V0.
```

```r
select.paths <- which(colSums(paths <= V0) == 366)
new.paths <- paths[,select.paths]
# Plot.
matplot(new.paths,
        type = "l", col = c(1:100), lwd = 1, lty = 1,
        ylab = expression(paste(V[t])),
        xlab = "Time in days (0 to T)")
abline(h = D, col = "red", lwd = 2)
points(1, V0, pch = 19, cex = 1.5, col = "blue")
```



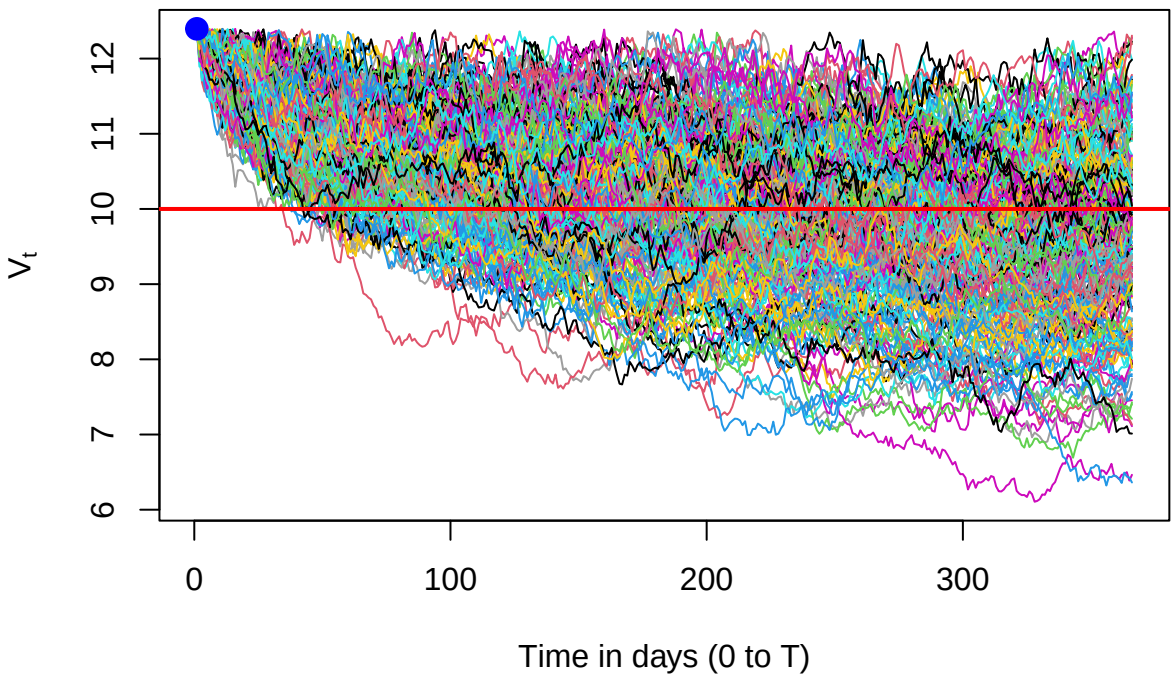

```r
# Probability
sum(new.paths < 10) / length(new.paths)
```

```
## [1] 0.4754098
```

Here, only path number 49 out of 100 meet our condition. This path meets $V_t < D$ in 47.54098% of the cases. Let's increase the number of simulations because 100 was apparently rather small. With 1,000 simulations the probability is 43.23445%. Let's see the case of

10,000 simulations.

```
set.seed(3) # Reproducibility
paths <- sapply(1:10000, V0.sim) # Create 10,000 paths.
# Select those paths with all 366 values lower or equal than V0.
select.paths <- which(colSums(paths <= V0) == 366)
new.paths <- paths[,select.paths]
# Plot.
matplot(new.paths,
        type = "l", col = c(1:10000), lwd = 1, lty = 1,
        ylab = expression(paste(V[t])),
        xlab = "Time in days (0 to T)")
abline(h = D, col = "red", lwd = 2)
points(1, V0, pch = 19, cex = 1.5, col = "blue")
```




```
# Probability
sum(new.paths < 10) / length(new.paths)
```

```
## [1] 0.3596873
```

Now the probability is 35.96873%. We can consider this one as the final result for our purposes.

Here are 100,000 simulated V_T showed in a histogram. The reference for this section is 15.1 (John C. Hull). Note that instead of drawing the complete paths, we can directly get the distribution of V_T. The resulting distribution is log-normal.

```
set.seed(13)
# 100,000 values of VT at once.
# Here, I added a TT value twice. This was not a mistake, because in the tutorial this
VT <- exp(rnorm(100000, log(V0) + (0.05 - (sv^2) / 2)*TT, sv*sqrt(TT)))
# Plot results.
# You may need to change the 100 if colors do not show properly. Sometimes this value
h <- hist(VT, 100, plot = FALSE)
ccat <- cut(h$breaks, c(-Inf, D, Inf))
# You may also need to change the xlim if a big red square appears as changes in matur
plot(h, main = NULL, col = c("red", "blue")[ccat],
     xlab = "Value of the assets at maturity", xlim = c(4, 30))
legend("topright",
legend = c("The red area is 12.657%, and represents
the simulated probability of default.
The probability of default of the model
following the textbook example is
N(-d_2)=12.69396%. As we can see,
both are very close."),
col = c("red", "blue"), pch = c(19), bg = "white", bty = "n", cex = 0.8)
```

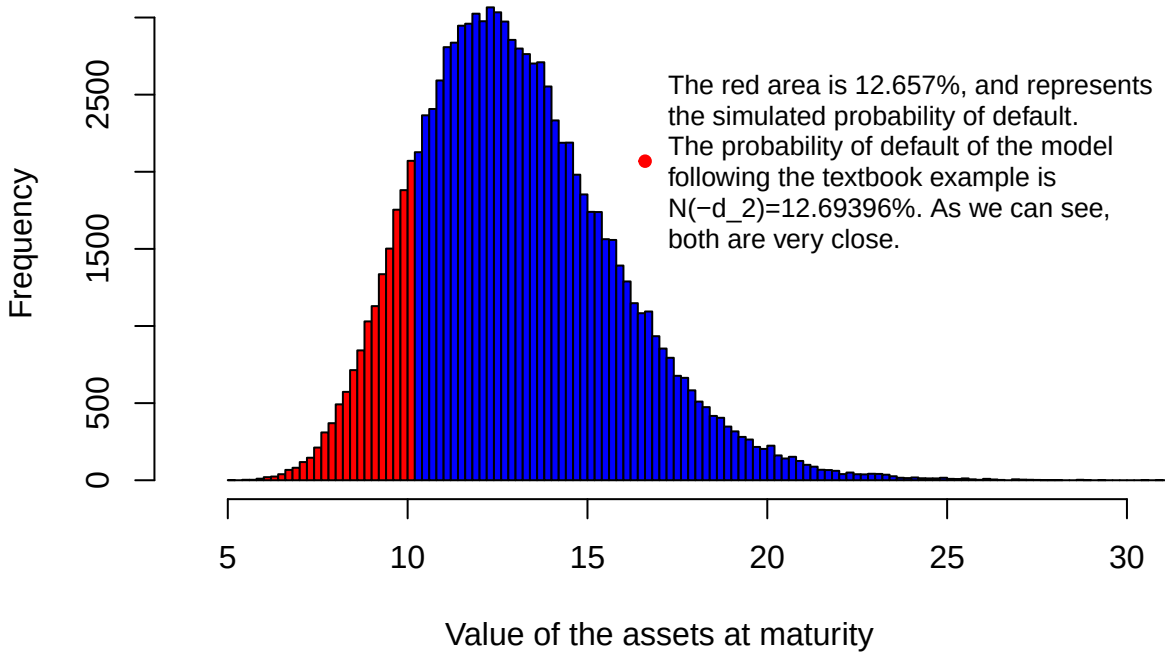


Figure 2.3.6: Histogram of 100,000 values of the assets at maturity.

The probability of default can be calculated as:

```
sum(VT < 10) / 100000
```

```
## [1] 0.12657
```

Note that 0.12657 is now much closer than 0.12693963. Convergence achieved.

This firm can be represented as an European call option payoff. The payoff of a typical stock option is $c = max(S_T - K, 0)$ or equivalently in terms of Merton's model: $E_T = max(V_T - D, 0)$.

```
ET <- pmax(VT - D, 0) # payoff function of a typical call option.
plot(sort(VT), sort(ET), type = "l", lwd = 4, xlim = c(5, 20),
     ylim = c(0, 10), xlab = "Simulated assets at maturity (V_T)",
     ylab ="Simulated equity at maturity (E_T)")
abline(v = 10, lty = 2)
```

```
legend("right", legend = c("E_T=max(V_T-D,0). Given the
simulation, E_T=0 happens in
12.657% of the cases.
According to the model, E_T=0
happens in 12.69396% of the
cases."),
bg = "white", bty = "n", cex = 0.7)
```

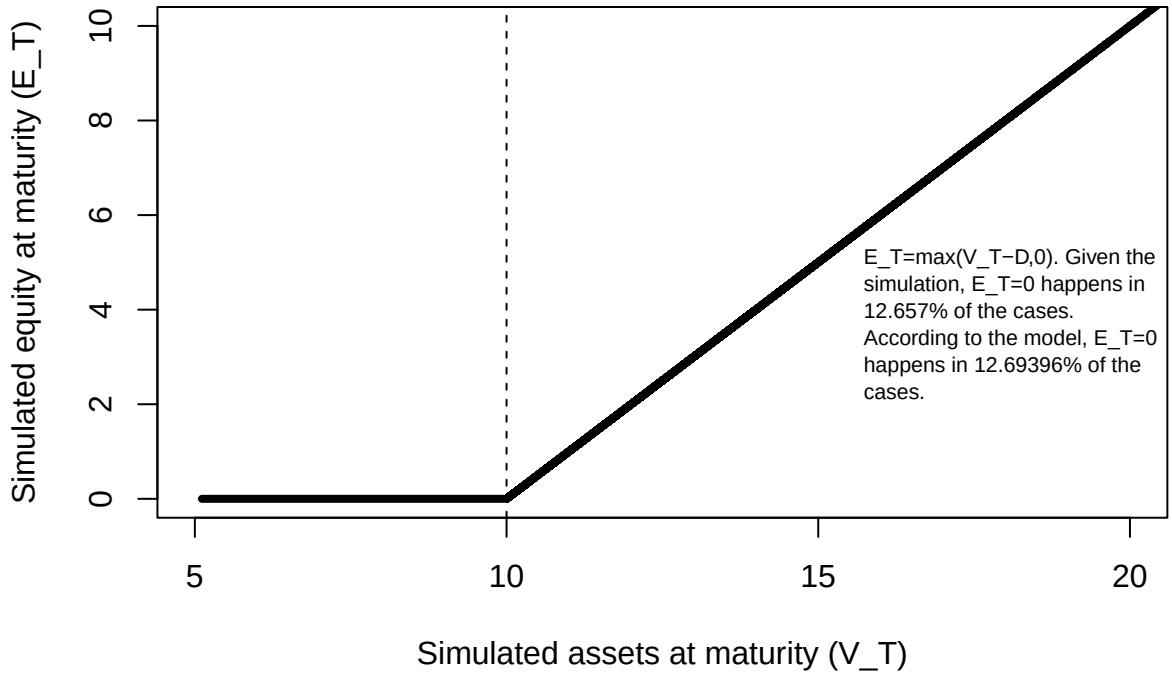


Figure 2.3.7: A view of the firm's balance sheet in the future. Looks like a typical European call option payoff.

This diagram show the relevant balance sheet accounts in the future, at maturity. This is because the E_T is a function of V_T and D. As long as $V_T < D$, then the value of $E_T = 0$. Otherwise, the value of $E_T = V_T - D$, just as the accounting equation suggest but in future terms. Note that the Merton's model say nothing about the estimate value of E_T, instead we have an estimate about how likely is that $V_T < D$, or in other words how likely is that

$E_T < 0$.

Then, we know that $E_T = max(V_T - D, 0)$. The Black???Scholes???Merton formula can also estimate the theoretical value of E_0. This is, if the observable market value given by the market capitalization available in any financial site is $E_0 = 3$, we can retrieve the theoretical value of E_0 so we can compare whether the firm observable market value is over or undervalued. In other words, the Black-Scholes-Merton model can give us an estimate of the firm value. In particular, the model gives the value of the equity today as:

$E_0 = V_0 N(d_1) - De^{-rT} N(d_2)$

This is equation 24.3 in Hull. To estimate E_0, we need V_0 and σ_V and these two values were already estimated before with the implementation of the min.footnote10 function above. It is true that this minimization requires E_0, the point here is to use the observable market capitalization $E_0 = 3$ to find out the unobsevable V_0 and σ_V and then estimate the theoretical value of E_0.

The code is:

```
# Black-Scholes call.
E0.theo <- function(V0, D, rf, TT, sv) {
  d1 <- (log(V0 / D) + (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))
  d2 <- d1 - sv * sqrt(TT)
  c <- V0 * pnorm(d1) - D * exp(-rf * TT) * pnorm(d2)
}
V0 <- 12.39578 # Assets.
D <- 10 # Debt.
rf <- 0.05 # Risk-free rate.
TT <- 1 # Maturity.
sv <- 0.2123041 # Volatility of assets.
(E0.theo(V0, D, rf, TT, sv))
```

```
## [1] 3.000357
```

Then, we can argue that the market value of the firm is slightly undervalued since the theoretical value of the firm is 3.000357 compared with 3. This approach opens the possibility to value firms. It also opens the possibility to compare the book value in the Balance sheet of the total assets versus the estimate V_0. This will allow us to understand the difference between book value and market value of the assets, not only of the equity. We can even implement a similar analysis about the market value of the debt just as we did before.

2.4 The probability of default as a function of some parameters.

We can expand our analysis by evaluating how changes in parameters change the probability of default in the context of the Merton model. This is interesting because we can move from estimating a probability of default to propose changes in the firm's parameters like the capital structure to reduce a firm's probability of default. First, we need the probability of default as a function.

```
pd <- function(E0, se, rf, TT, D) {
  eq24.3 <- function(V0, sv) {
  ((V0 * pnorm((log(V0 / D) + (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) -
  D * exp(-rf * TT) * pnorm(((log(V0 / D) +
  (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) - sv * sqrt(TT)) - E0)) }
eq24.4 <- function(V0, sv) {
  ((pnorm((log(V0 / D) + (rf + sv^2 / 2) * TT) / (sv * sqrt(TT))) *
      sv * V0 - se * E0)) }
  min.footnote10 <- function(x)
  (eq24.3(x[1], x[2]))^2 + (eq24.4(x[1], x[2]))^2
V0_sv <- optim(c(1, 1), min.footnote10)
V0 <- V0_sv$par[1]
sv <- V0_sv$par[2]
pd <- pnorm(-(((log(V0 / D) + (rf + sv^2 / 2) * TT) /
          (sv * sqrt(TT))) - sv * sqrt(TT)))
pd }
```

Now we have a convenient function: $pd = f(E_0, \sigma_E, rf, T, D)$. Remember these five parameters are required to estimate V_0 and σ_V before calculating pd.

```
# Evaluate the case of the textbook example.
pd1 <- pd(3, 0.8, 0.05, 1, 10)
pd1
```

```
## [1] 0.1269396
```

Now, we create vectors of parameters and evaluate the `pd` function.

```
l = 50 # Vectors of length 50.
# Create vectors of the parameters.
x.E <- seq(from = 1, to = 20, length.out = l)
x.rf <- seq(0, 4, length.out = l)
```

```r
x.D <- seq(1, 20, length.out = l)
x.T <- seq(0.5, 20, length.out = l)
x.se <- seq(0, 3, length.out = l)
# Evaluate the pd at different values.
pd.E <- mapply(pd, x.E, se, rf, TT, D)
pd.rf <- mapply(pd, E0, se, x.rf, TT, D)
pd.D <- mapply(pd, E0, se, rf, TT, x.D)
pd.T <- mapply(pd, E0, se, rf, x.T, D)
pd.se <- mapply(pd, E0, x.se, rf, TT, D)
```

```r
pd.E # vemos que hay en pd.E
```

```
##  [1] 0.15540348 0.14901392 0.14320202 0.13768571 0.13261199 0.12764964
##  [7] 0.12308008 0.11884785 0.11459711 0.11076647 0.10703359 0.10358685
## [13] 0.10027904 0.09705949 0.09400846 0.09099382 0.08829345 0.08564796
## [19] 0.08301610 0.08064451 0.07827914 0.07603520 0.07387230 0.07173820
## [25] 0.06986438 0.06783357 0.06602684 0.06416119 0.06256656 0.06089248
## [31] 0.05923871 0.05772111 0.05624785 0.05480055 0.05342181 0.05206815
## [37] 0.05076260 0.04952178 0.04825154 0.04709231 0.04595749 0.04482100
## [43] 0.04375887 0.04272127 0.04173439 0.04072829 0.03984186 0.03887309
## [49] 0.03795462 0.03714601
```

```r
# Busco el valor de pd mas parecido a 0.05
index <- which(almost.equal(pd.E, 0.05, tolerance = 0.01))
# ??Que numero de observacion es? (da 38 con los datos del tutorial)
index
```

```
## [1] 38
```

```r
# ??Que valor de equity me da una pd de 0.05?
x.E[index] # da 15.34694 con los datos del tutorial.
```

```
## [1] 15.34694
```

```r
pd.E[index] # da 0.04952178 con los datos del tutorial.
```

```
## [1] 0.04952178
```

Let's plot the probability of default as a function of equity. What we are doing here is to evaluate the probability of default function 50 times, as the E_0 has 50 values from 1 to 20.

By doing this, we end up with 50 values of the probability of default.

```
plot(x.E, pd.E, type = "l", ylab = "Probability of default",
     xlab = "Equity at time zero", lwd = 3, ylim = c(0, 0.15))
lines(x.E, pd.E)
abline(v = E0, lty = 2)
abline(h = pd1, lty = 2)
points(E0, pd1, pch = 1, cex = 3, col = "red", lwd = 2)
```

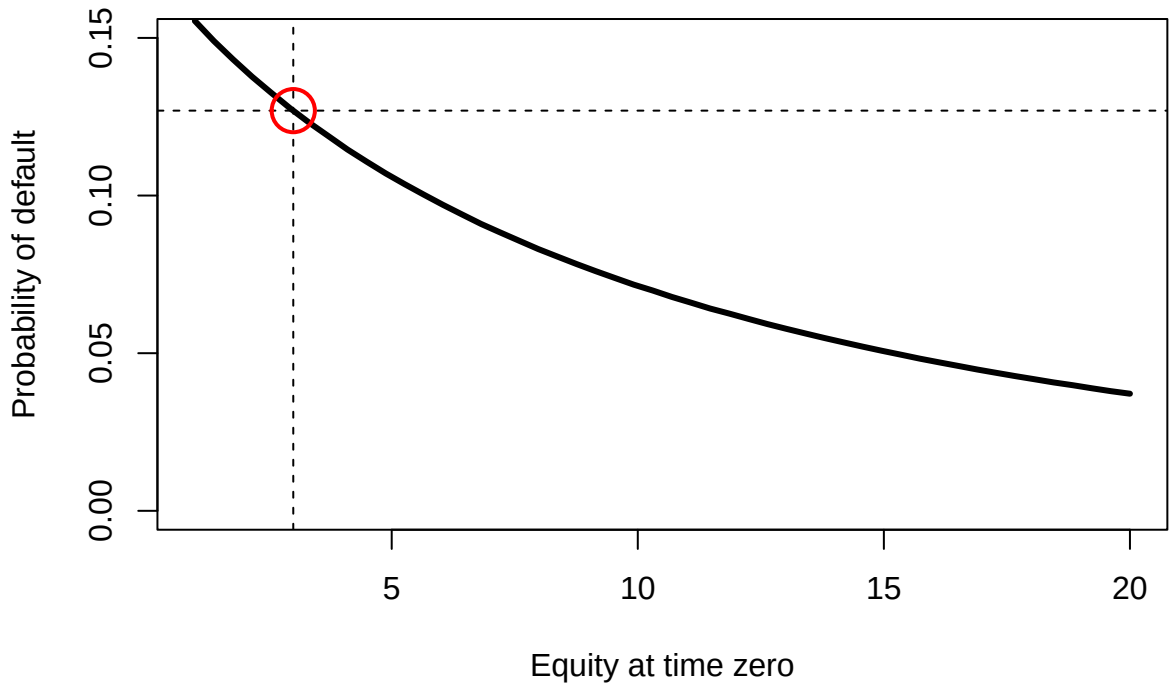


Figure 2.4.1: The probability of default as a function of the equity at time zero.

These relationships between the parameters and the probability of default are all non-linear. This is interesting because according to the model, the current levels of the parameters are important when deciding which parameter might lead to a high or low impact over the probability of default.

```
plot(x.rf, pd.rf, type = "l", ylab = "Probability of default",
     xlab = "Risk-free rate", lwd = 3)
abline(h = 0, lty = 2)
abline(v = rf, lty = 2)
abline(h = pd1, lty = 2)
points(rf, pd1, pch = 1, cex = 3, col = "red", lwd = 2)
```

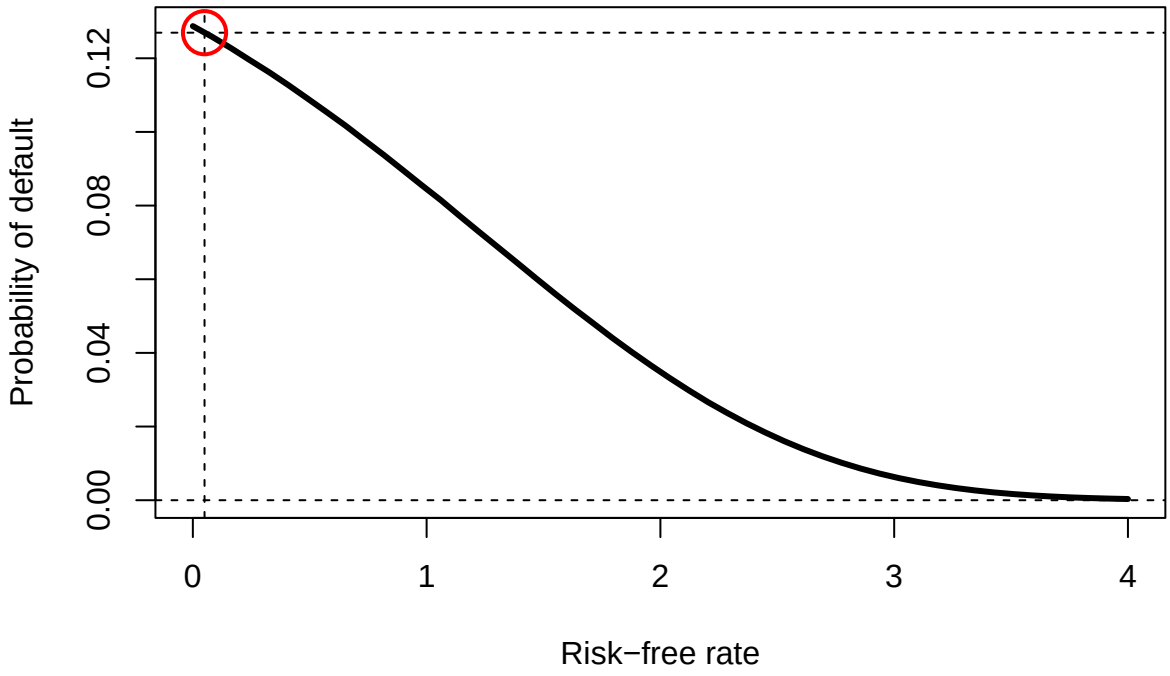


Figure 2.4.2: The probability of default as a function of the risk-free rate.

Probability of default as a function of debt. Note that adding 1 or subtracting 1 unit of debt has different impact over the probability of default.

```
plot(x.D, pd.D, type = "l", ylab = "Probability of default",
     xlab = "Debt", lwd = 3, ylim = c(0, 0.15))
abline(h = 0, lty = 2)
abline(v = D, lty = 2)
```

```
abline(h = pd1, lty = 2)
points(D, pd1, pch = 1, cex = 3, col = "red", lwd = 2)
```

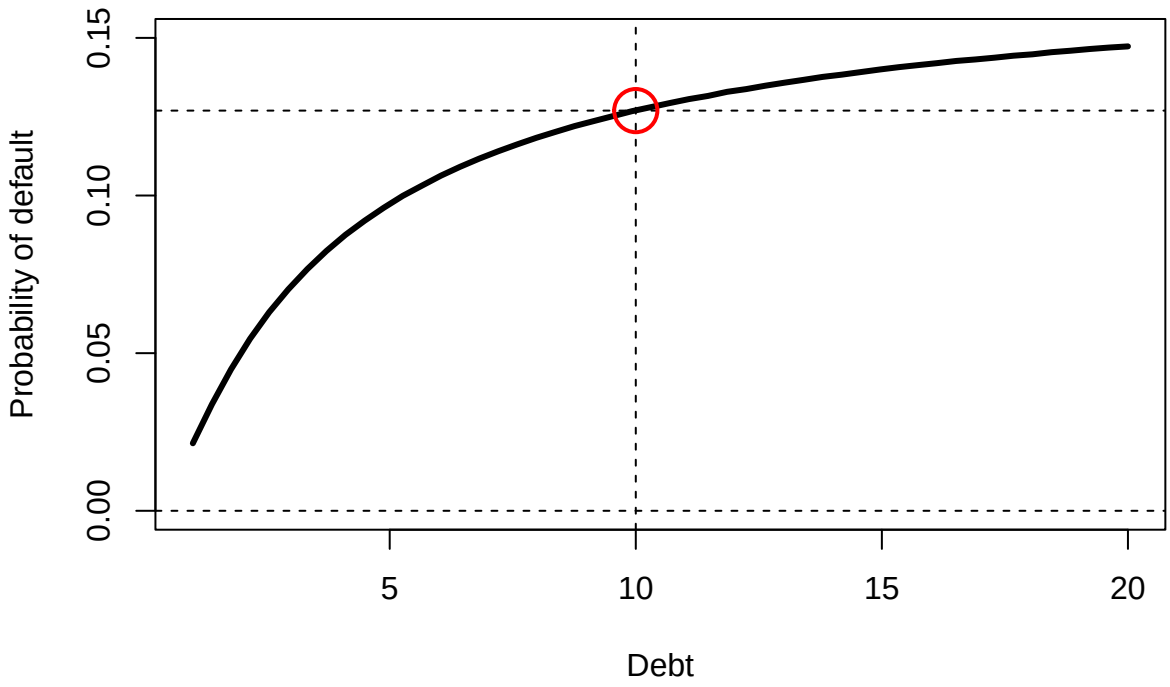


Figure 2.4.3: The probability of default as a function of the debt.

Probability of default as a function of maturity. Note that asking for a higher debt maturity (as in a negotiation), do not lead to a lower probability of default if we keep the rest of the parameters unchanged in our firm. Imagine we arrange a meeting with the bank manager and we ask for more time to pay our debt. The bank manager should not (in principle) extend the debt maturity as long as we commit to do some changes in the firm. For example, we can negotiate a longer maturity and promise to increase the firm's equity.

```
plot(x.T, pd.T, type = "l", ylab = "Probability of default",
     xlab = "Maturity", lwd = 3)
abline(h = 0, lty = 2)
abline(v = TT, lty = 2)
```

```
abline(h = pd1, lty = 2)
points(TT, pd1, pch = 1, cex = 3, col = "red", lwd = 2)
```

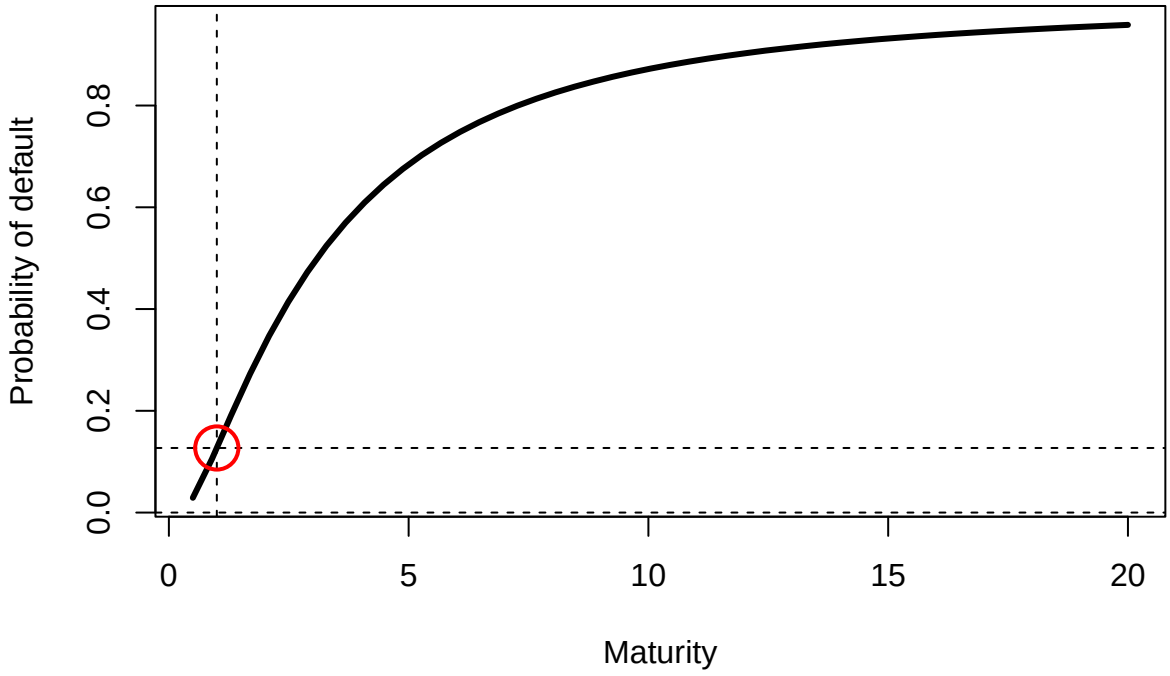


Figure 2.4.4: The probability of default as a function of the debt's maturity.

Probability of default as a function of the volatility of the equity σ_E.

```
plot(x.se, pd.se, type = "l", ylab = "Probability of default",
     xlab = "Standard deviation of equity", lwd = 3)
abline(h = 0, lty = 2)
abline(v = se, lty = 2)
abline(h = pd1, lty = 2)
points(se, pd1, pch = 1, cex = 3, col = "red", lwd = 2)
```

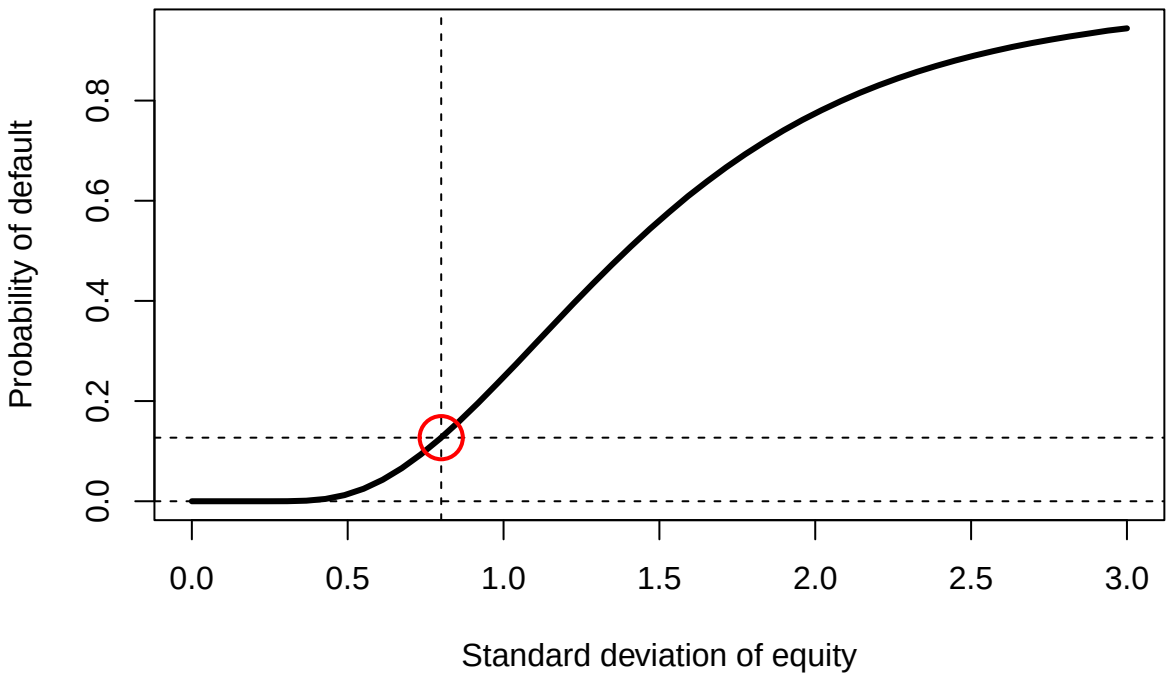


Figure 2.4.5: The probability of default as a function of the volatility of the equity.

In the previous plots we evaluate the probability of default function by changing one of the following parameters: E_0, σ_E, rf, T or D. We were able to do that because we constructed a vector of 50 different values for these five parameters. Note that V_0 and σ_V do not remain fixed because they are at the same time a function of E_0, σ_E, rf, T and D. Now, let's start by plotting the probability of default as a function of E_0 and σ_E.

One alternative is to evaluate the probability of default function 50 times by taking the 50 pairs of values of E_0 and σ_E. By doing that, we will end with three vectors of size 50 that we can plot as a three dimension scatter plot.

```
ED <- mapply(pd, x.E, x.se, rf, TT, D)
p.ED <- scatterplot3d(x.E, x.se, ED, pch = 16, type = "h", color = 1:50,
                      angle = 100, xlab = "Equity at time zero",
                      ylab = "Volatility of equity",
                      zlab = "Probability of default")
```

```
p.ED$points3d(E0, se, pd1, type = "h", col = "red", pch = 20, cex = 3)
```

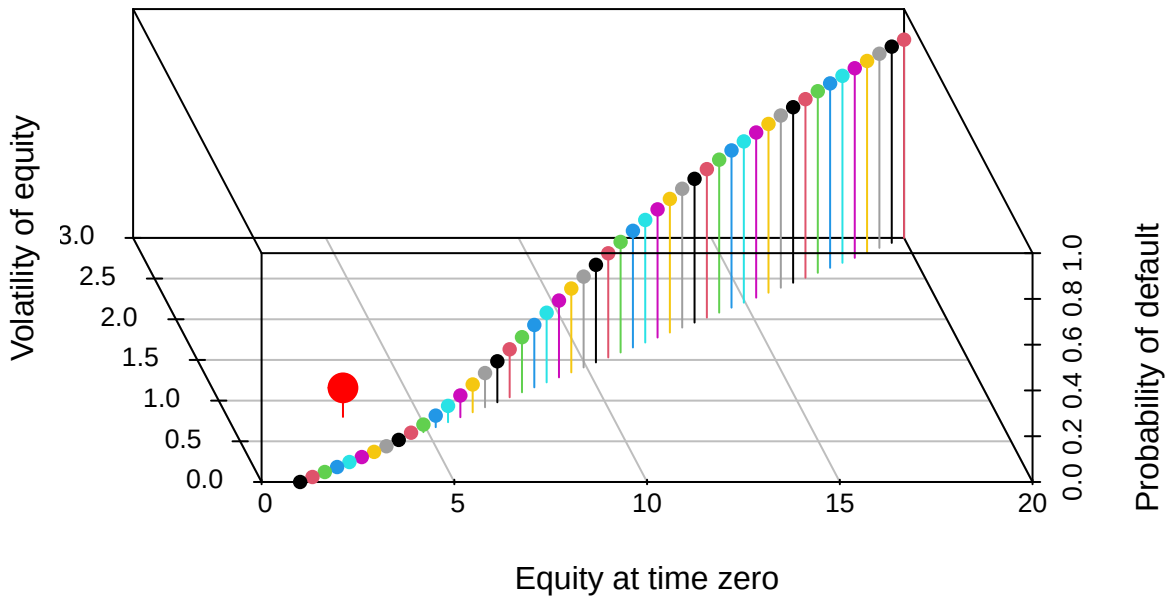


Figure 2.4.6: The probability of default as a function of the equity at time zero and the volatility of equity.

The red point represent the case of the original textbook example in which $pd = f(E_0 = 3, \sigma_E = 0.8, rf = 0.05, T = 1, D = 10) = 0.1269396$. Note that the red point is not part of the 50 plotted observations simply because there is no case in which the probability of default function is evaluated in $E_0 = 3, \sigma_E = 0.8$. Although the plot above is not incorrect, it might be incomplete as we are not showing all possible values of the probability of default. One alternative to show a more complete plot is to evaluate all possible combinations of E_0 and σ_E to evaluate the probability of default function. If we do that, we will have 50 values for E_0 and σ_E that represent the x and y axis, and a 50×50 matrix containing the probability of default. This allows us to plot a surface plot and a contour plot.

```
# Create the empty matrix.
p_E_se <- matrix(0, nrow = l, ncol = l)
# Fill the empty matrix with probability of default values.
for(i in 1 : l){ # Is there an easier way to do this?
  for(j in 1 : l){
    p_E_se[i,j] <- mapply(pd, x.E[i], x.se[j], rf, TT, D) } }
```

Let's plot the results.

```
# Plot results.
p.ED1 <- persp(x.E, x.se, p_E_se, zlab = "pd", xlab = "Equity at time zero",
               ylab = "Volatility of equity", theta = 60, phi = 10,
               expand =  0.5, col = "orange", shade = 0.2,
               ticktype = "detailed")
# Add the original pd value as in the textbook example.
points(trans3d(E0, se, pd1, p.ED1), cex = 2, pch = 19, col = "blue")
```

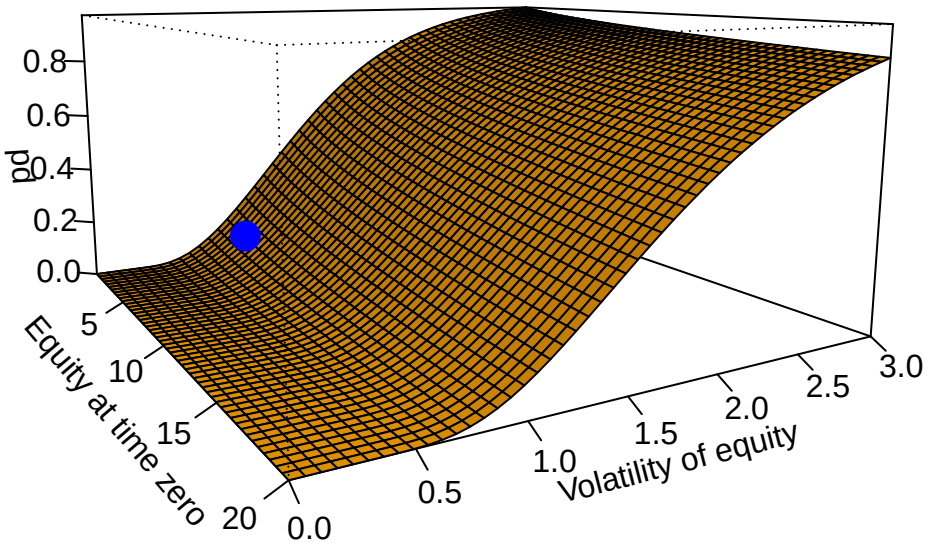


Figure 2.4.7: The probability of default as a function of the equity at time zero and the volatility of equity: A plane or surface view.

Now the plot becomes a plane. A contour plot simplify the reading of a three dimension plot by reducing it into two dimensions: x and y coordinates. The value of the probability of default is represented by the contour lines.

```
contour(x.E, x.se, p_E_se, xlab = "Equity at time zero",
        ylab = "Volatility of equity", lwd = 2)
points(E0, se, pch = 19, col = "blue", cex = 2)
```

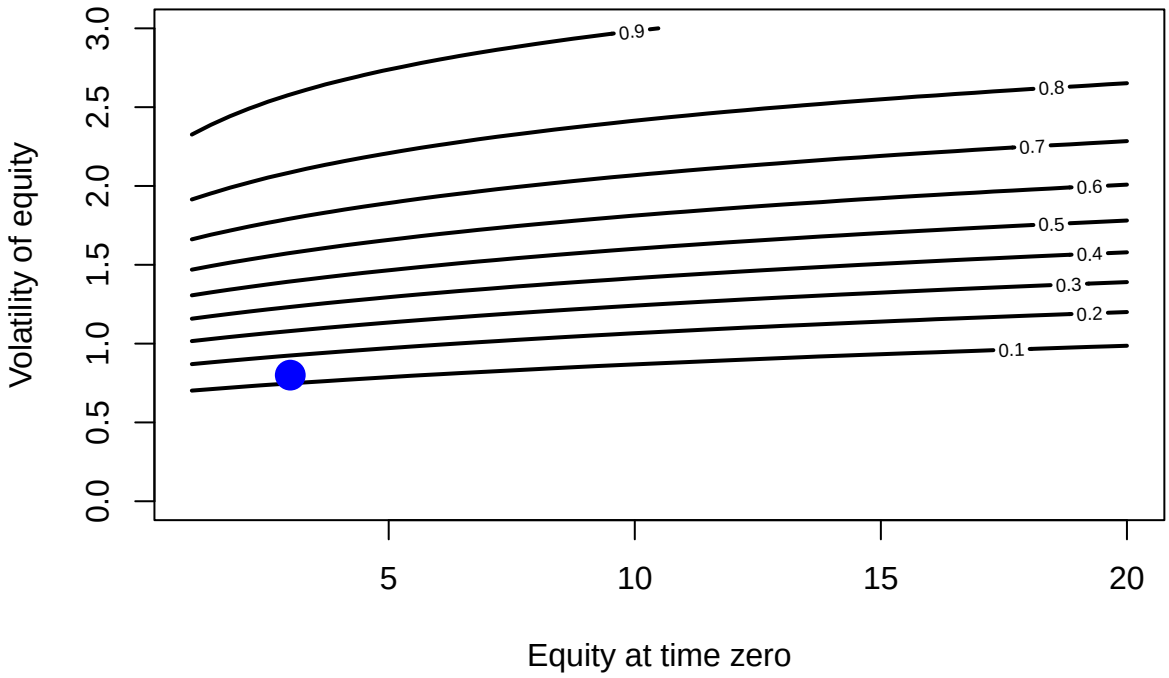


Figure 2.4.8: The probability of default as a function of the equity at time zero and the volatility of equity: A contour view.

The blue point is the original case. Note that the value is slightly higher than 0.1, which is consistent with the $pd = 0.1269396$.

2.5 GoT: capital structure.

The problem. Daenerys Targaryen owns a big manufacturing firm that produces fire extinguishers. Surprisingly, the firm data corresponds exactly as the data in the example 24.3. She is planning to ask the Iron Bank of Braavos for a loan using the peaceful civilized way (without dragons). However, she would like to reduce the probability of default of the firm first, in that way she might negotiate better credit conditions. Daenerys has some understanding about very basic finance because she knows that she could either reduce the debt, or increase the capital in order to reduce the probability of default. Doing both alternatives at the same time is clearly more expensive so she is not very keen about it.

What is the best strategy to reduce the probability of default? Reduce the debt by 2 or increase the capital by 2?

```
D.seq <- seq(from = 0, to = 13, by = 0.1)
# Evaluate pd function: E0 changes from 1 to 5; debt goes from 0 to 13.
E1 <- mapply(pd, 1, se, rf, TT, D.seq)
E2 <- mapply(pd, 2, se, rf, TT, D.seq)
E3 <- mapply(pd, 3, se, rf, TT, D.seq)
E4 <- mapply(pd, 4, se, rf, TT, D.seq)
E5 <- mapply(pd, 5, se, rf, TT, D.seq)
```

Plot the results.

```
# We use these vectors for colors and legends in the following plots.
colors <- c("green", "purple", "black", "blue", "red")
leg <- c("E0=1", "E0=2", "E0=3 (initial value)", "E0=4", "E0=5")
# Plot.
plot(D.seq, E1, type = "l", col = "green", lwd = 3,
     xlab = "Debt value. Initially, D=10",
     ylab = "Probability of default. Initially PD=12.69%")
lines(D.seq, E2, col = "purple", lwd = 3)
lines(D.seq, E3, col = "black", lwd = 3)
lines(D.seq, E4, col = "blue", lwd = 3)
lines(D.seq, E5, col = "red", lwd = 3)
abline(v = D, lty = 2)
abline(h = pd1, lty = 2)
legend("bottomright", legend = leg, lwd = rep(3, 5), col = colors,
       bg = "white")
points(D, pd1, pch = 19, cex = 2)
```

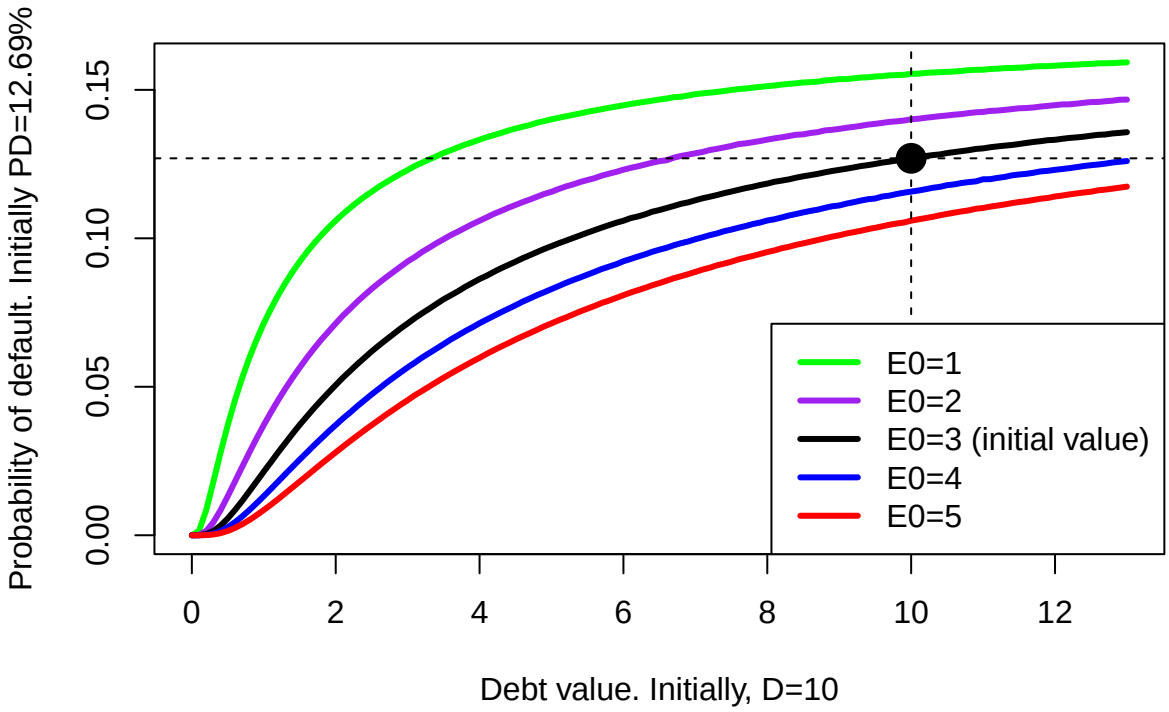


Figure 2.5.1: GoT problem. Probability of default and capital structure.

Let's illustrate the alternatives as clear as possible.

```
plot(D.seq, E1, type = "l", col = "green", lwd = 3,
     xlab = "Debt value. Initially, D=10",
     ylab = "Probability of default. Initially PD=12.69%")
lines(D.seq, E2, col = "purple", lwd = 3)
lines(D.seq, E3, col = "black", lwd = 3)
lines(D.seq, E4, col = "blue", lwd = 3)
lines(D.seq, E5, col = "red", lwd = 3)
abline(v = D, lty = 2)
abline(v = 8, lty = 2)
abline(h = pd1, lty = 2)
abline(h = pd(E0, se, rf, TT, 8), lty = 2)
abline(h = pd(5, se, rf, TT, D), lty = 2)
```

```
legend("bottomright", legend = leg, lwd = rep(3, 5), col = colors,
       bg = "white")
points(D, pd1, pch = 19, cex = 2)
points(D, pd(5, se, rf, TT, D), pch = 19, cex = 2, col = "red")
points(8, pd(3, se, rf, TT, 8), pch = 19, cex = 2, col = "yellow")
```

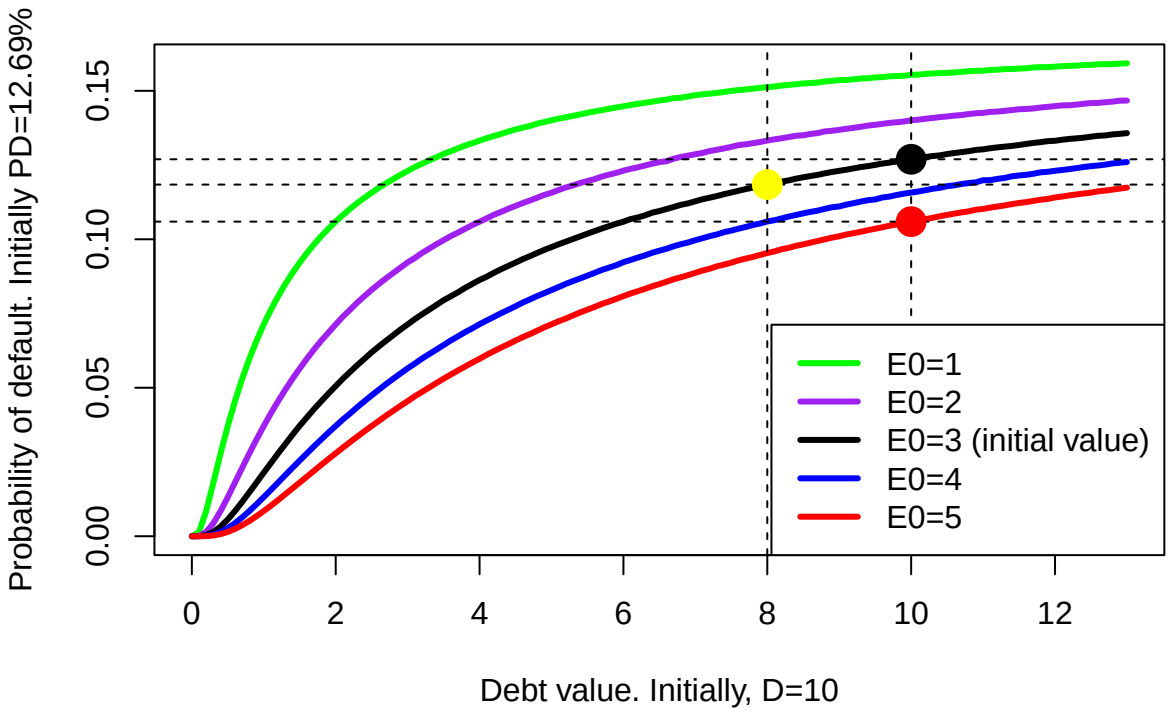


Figure 2.5.2: GoT solution. Probability of default and capital structure.

Consider a different initial situation. Now, $E_0 = 2$ and $D = 2$ and the rest remains the same. In this case, the probability of default is now 7.13%. What is the best strategy to reduce the probability of default? Reduce the debt by 1 or increase the capital by 1?

```
leg2 <- c("E0=1", "E0=2 (initial value)", "E0=3", "E0=4", "E0=5")

plot(D.seq, E1, type = "l", col = "green", lwd = 3,
     xlab = "Debt value. Initially, D=2",
```

```
     ylab = "Probability of default. Initially PD=7.13%",
     xlim = c(0, 2.5), ylim = c(0, 0.08))
lines(D.seq, E2, col = "purple", lwd = 3)
lines(D.seq, E3, col = "black", lwd = 3)
lines(D.seq, E4, col = "blue", lwd = 3)
lines(D.seq, E5, col = "red", lwd = 3)
abline(v = 2, lty = 2)
abline(h = pd(2, se, rf, TT, 2), lty = 2)
legend("bottomright", legend = leg2, lwd = rep(3, 5), col = colors,
       bg = "white", cex = 0.6)
points(2, pd(2, se, rf, TT, 2), pch = 19, cex = 2, col = "purple")
```

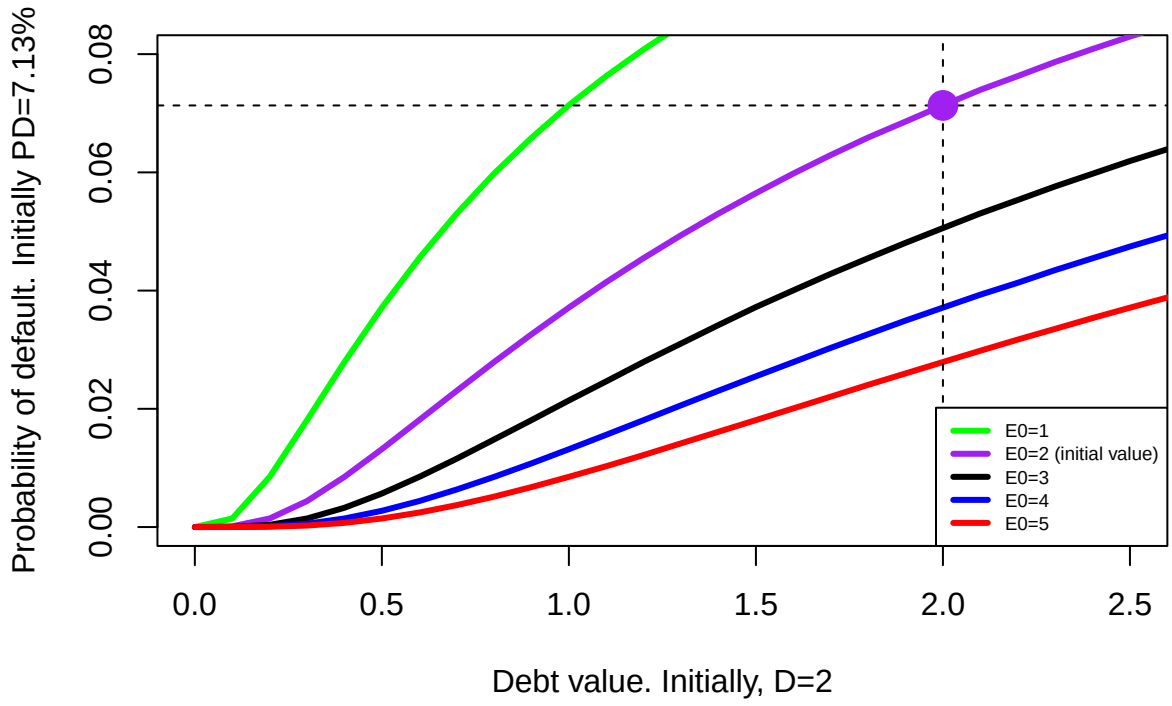


Figure 2.5.3: GoT problem 2. Probability of default and capital structure.

The solution can be illustrated as follows:

```
plot(D.seq, E1, type = "l", col = "green", lwd = 3,
     xlab = "Debt value. Initially, D=2",
     ylab = "Probability of default. Initially PD=7.13%",
     xlim = c(0, 2.5), ylim = c(0, 0.08))
lines(D.seq, E2, col = "purple", lwd = 3)
lines(D.seq, E3, col = "black", lwd = 3)
lines(D.seq, E4, col = "blue", lwd = 3)
lines(D.seq, E5, col = "red", lwd = 3)
abline(v = 2, lty = 2)
abline(v = 1, lty = 2)
abline(h = pd(2, se, rf, TT, 2), lty = 2)
abline(h = pd(2, se, rf, TT, 1), lty = 2)
abline(h = pd(E0, se, rf, TT, 2), lty = 2)
legend("bottomright", legend = leg2, lwd = rep(3, 5), col = colors,
       bg = "white", cex = 0.6)
points(2, pd(2, se, rf, TT, 2), pch = 19, cex = 2, col = "purple")
points(1, pd(2, se, rf, TT, 1), pch = 19, cex = 2, col = "yellow")
points(2, pd(E0, se, rf, TT, 2), pch = 19, cex = 2)
```

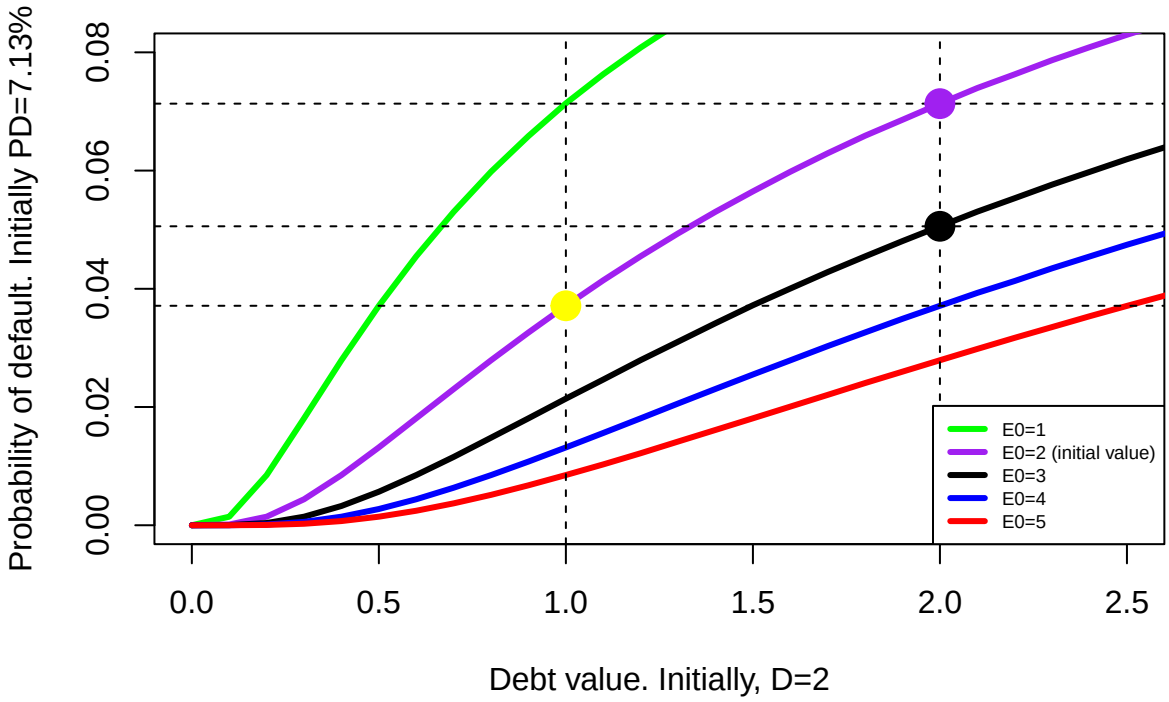


Figure 2.5.4: GoT solution 2. Probability of default and capital structure.

Interesting.

Consider a third problem. In a parallel universe, Daenerys Targaryen's firm has some liquidity short-term troubles. She needs either more cash now or some more time to pay her current debt. She would like to know which alternative leads to the lowest increase of the probability of default. Add \$2 more to her current debt, or ask for a half year more time to pay her current debt?

```
T1 <- mapply(pd, E0, se, rf, 0.5, D.seq)
T2 <- mapply(pd, E0, se, rf, 1, D.seq)
T3 <- mapply(pd, E0, se, rf, 1.5, D.seq)
T4 <- mapply(pd, E0, se, rf, 2, D.seq)
T5 <- mapply(pd, E0, se, rf, 2.5, D.seq)
```

```
leg3 <- c("T=2.5", "T=2", "T=1.5", "T=1", "T=0.5")

plot(D.seq, T5, type = "l", col = "green", lwd = 3,
     xlab = "Debt value. Initially, D=6",
     ylab = "Probability of default. Initially PD=20.33%")
lines(D.seq, T4, col = "purple", lwd = 3)
lines(D.seq, T3, col = "black", lwd = 3)
lines(D.seq, T2, col = "blue", lwd = 3)
lines(D.seq, T1, col = "red", lwd = 3)
abline(v = 6, lty = 2)
abline(h = pd(E0, se, rf, 1.5, 6), lty = 2)
legend("bottomright", legend = leg3, lwd = rep(3, 5), col = colors,
       bg = "white")
points(6, pd(E0, se, rf, 1.5, 6), pch = 19, cex = 2)
```

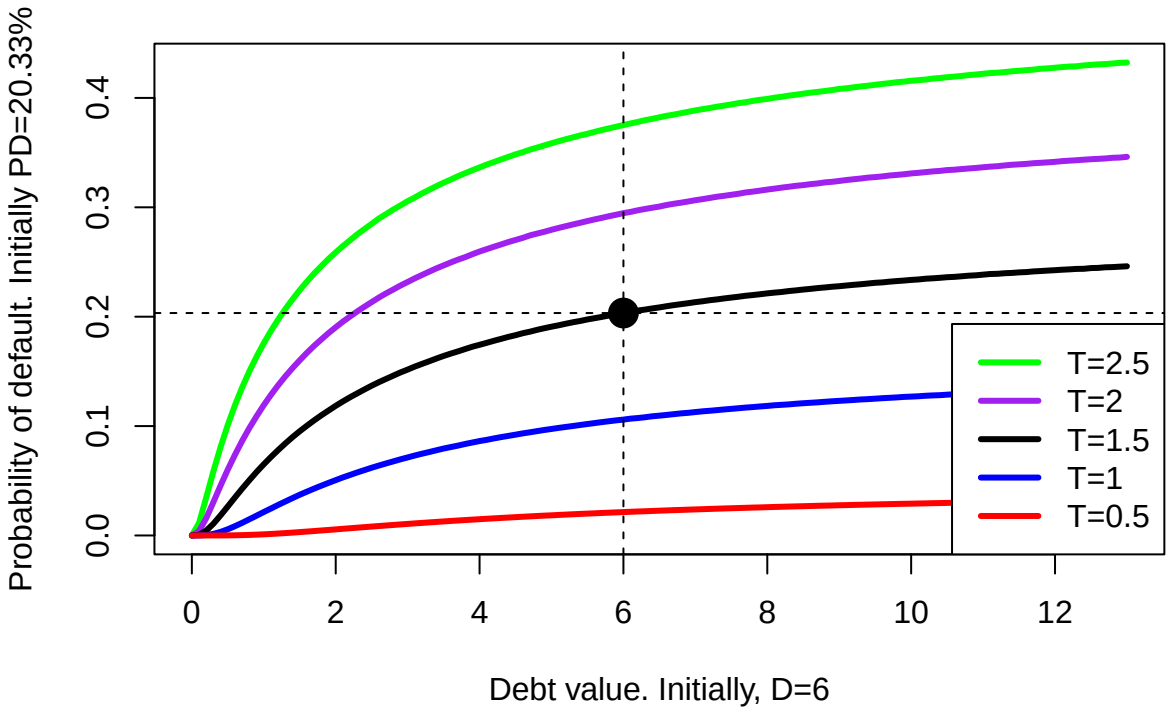


Figure 2.5.5: GoT problem 3. More debt or longer maturity?

Let's visualize the result.

```
plot(D.seq, T5, type = "l", col = "green", lwd = 3,
     xlab = "Debt value. Initially, D=6",
     ylab = "Probability of default. Initially PD=20.33%")
lines(D.seq, T4, col = "purple", lwd = 3)
lines(D.seq, T3, col = "black", lwd = 3)
lines(D.seq, T2, col = "blue", lwd = 3)
lines(D.seq, T1, col = "red", lwd = 3)
abline(v = 6, lty = 2)
abline(v = 8, lty = 2)
abline(h = pd(E0, se, rf, 1.5, 6), lty = 2)
abline(h = pd(E0, se, rf, 2, 6), lty = 2)
abline(h = pd(E0, se, rf, 1.5, 6), lty = 2)
abline(h = pd(E0, se, rf, 1.5, 8), lty = 2)
legend("bottomright", legend = leg3, lwd = rep(3, 5), col = colors,
       bg = "white")
points(6, pd(E0, se, rf, 1.5, 6), pch = 19, cex = 2)
points(8, pd(E0, se, rf, 1.5, 8), pch = 19, cex = 2, col = "yellow")
points(6, pd(E0, se, rf, 2, 6), pch = 19, cex = 2, col = "purple")
```

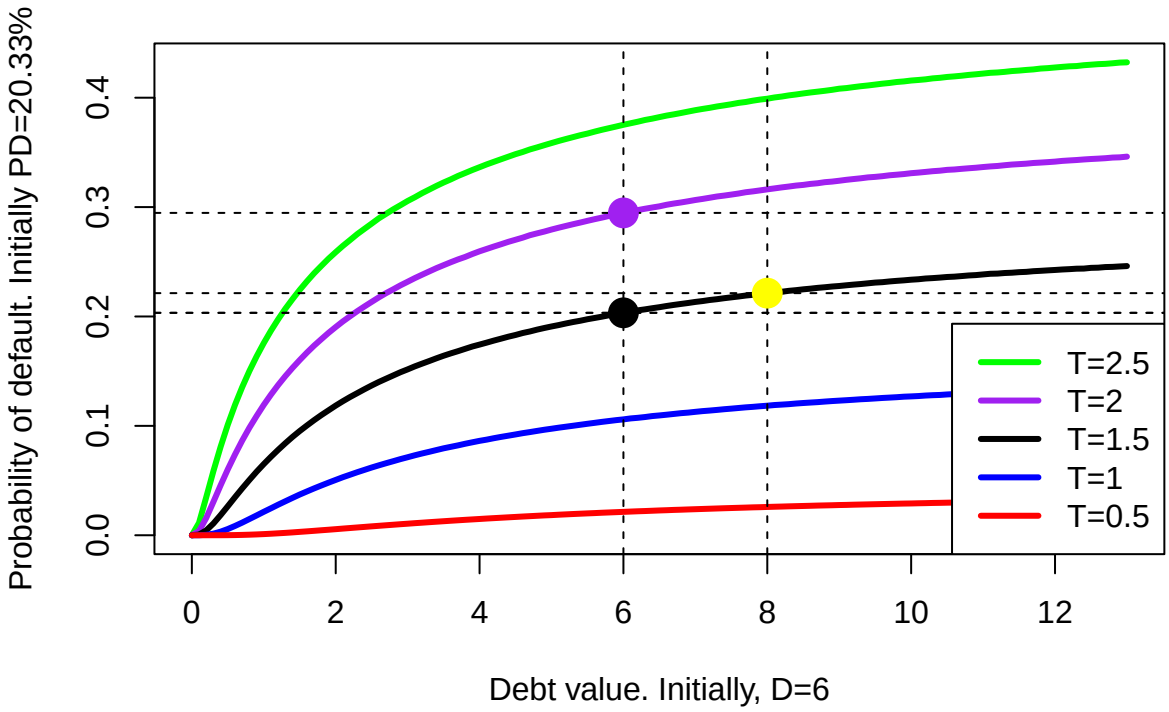


Figure 2.5.6: GoT answer 3. More debt or longer maturity?

You are expected to discuss the results and implications.

3 The Gaussian copula model.

Copulas allow us to decompose a joint probability distribution into their marginals (which by definition have no correlation) and a function which couples (hence the name) them together and thus allows us to specify the correlation separately. The copula is that coupling function. Here, we introduce the simplest type of copulas into a very common problem in credit risk which is the time to default.

3.1 The basics.

In a finance-context, the variable x represents a firm's performance measure that goes from -4 to 4 in the horizontal axis. Strictly speaking, more extreme values of x like $-\infty$ and $+\infty$ are theoretically possible but are very rare and happen quite infrequently at least in real-life situations. At the moment, we do not care too much about what kind of performance measure this is, it could be liquidity for example, solvency, or any other that it is normalized to have values between -4 and 4. In a statistics-context, we can think that x is the so-called z-score in the context of the standardized normal distribution function.

```
# The standard normal distribution function is: y = f(x).
x <- seq(-4, 4, length = 500) # First define x.
y <- 1 / sqrt(2 * pi) * exp(-x ^ 2 / 2) # Define y as a function of x.
# Now plot.
plot(x, y, type = "l", lwd = 2, col = "red" , ylab = "dnorm(x)")
```

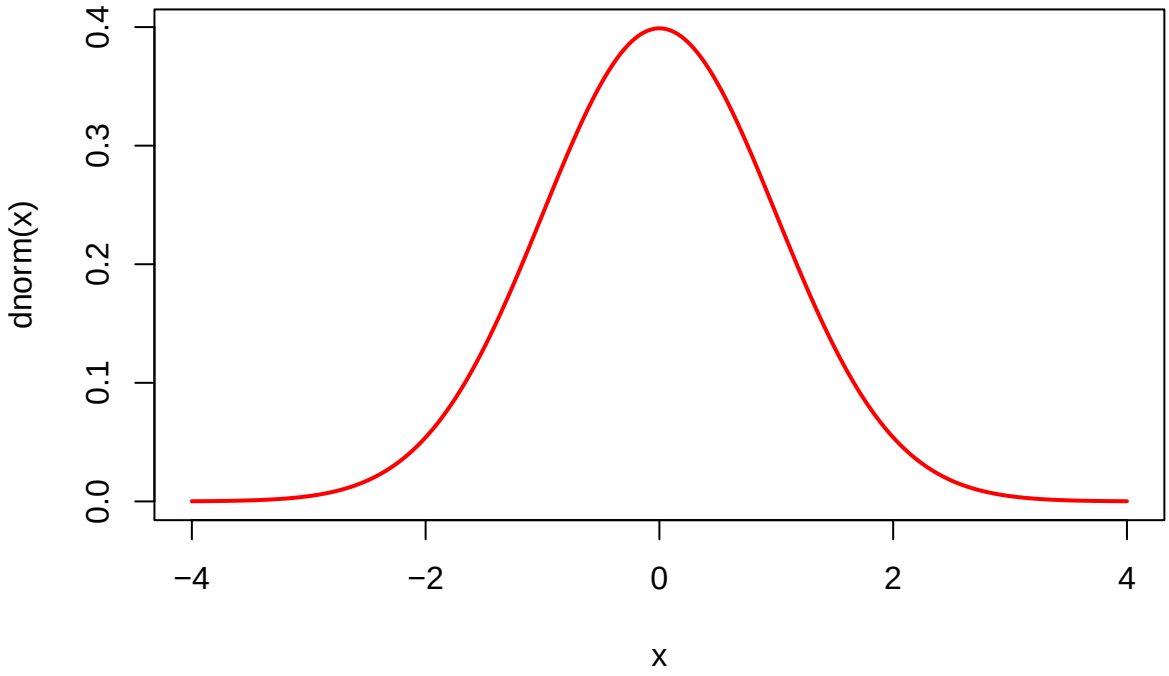


Figure 3.1.1: Standard normal distribution function.

The copula model, including the Gaussian, considers that this performance measure x in the horizontal axis is related with the firm's probability of default (from 0 to 1, or 0% to 100%). Graphically, the probability of default is represented by the area under the curve at the left of x. Low values of x accumulate small probabilities whereas high values of x accumulate high probabilities.

See the following example. If $x = 3$ (bad performance of a firm), then the firm's probability of default in math notation is $N(x)$, or $N(3)$, or `pnorm`$(3) = 0.998$ in R code, which is very high as it is close to 1. On the other hand, if $x = -3$ (good performance), then the probability of default $N(-3)$ or `pnorm`$(-3) = 0.001349898$ is very low as it is close to 0. Here, good performance is associated with negative values of x as the accumulated probability (default probability) is low. In the same way, bad performance is related with positive x values.

Then the `pnorm` R function allows us to transform x into a probability of default. Transform probabilities into x is also possible as the function `qnorm` is the inverse of `pnorm`. We can

easily demonstrate this by retrieving the x value given the probability. See for example: $N^{-1}(0.001349898) = -3$, or in R code: `qnorm`$(0.001349898) = -3$.

The function `dnorm` is relevant when we implement a graphical approach because it represents how frequent (or how likely) these values are given the standard normal distribution, so in both extreme values of x the value of `dnorm` is low. Here, extreme values of x can be represented by -4 and 4. This function delivers the height of the standard normal distribution, `dnorm`$(4) = 0.0001338302$, and `dnorm`$(-4) = 0.0001338302$. Given that the standard normal function is symmetrical, we have that in general: `dnorm`$(-x) =$ `dnorm`(x). The letter d in `dnorm` stands for density and it is maximum at $x = 0$. When plotting the density values we get the bell-shaped normal curve.

See how these `pnorm`, `qnorm`, `dnorm` R functions work and relate:

```
# Here, x has 11 values only.
x <- c(-Inf, -4:4, Inf) # vector of x values to evaluate functions.
ans <- data.frame(x, dnorm(x), pnorm(x), qnorm(pnorm(x)))
colnames(ans) <- c("x", "dnorm(x)=height", "pnorm(x)=pd", "qnorm(pd)=x")
kable(ans, caption = "Review of normal distribution functions.", digits = 5)
```

Table 3.1.1: Review of normal distribution functions.

x	dnorm(x)=height	pnorm(x)=pd	qnorm(pd)=x
-Inf	0.00000	0.00000	-Inf
-4	0.00013	0.00003	-4
-3	0.00443	0.00135	-3
-2	0.05399	0.02275	-2
-1	0.24197	0.15866	-1
0	0.39894	0.50000	0
1	0.24197	0.84134	1
2	0.05399	0.97725	2
3	0.00443	0.99865	3
4	0.00013	0.99997	4
Inf	0.00000	1.00000	Inf

Let's demonstrate that `dnorm` is maximum at $x = 0$.

```
x[which.max(ans$`dnorm(x)=height`)]
```

```
## [1] 0
```

You can type for example `?dnorm` in the RStudio console to see more details about this (and other) functions.

Note that the only distribution function that we are currently analyzing is the standard normal (Gaussian) distribution function. A standard normal distribution function is the one that has mean 0 and variance 1. There are other copula models that assume other kinds of distributions. These other kinds of copulas are useful when we are interested in modeling cases in which extreme values are more likely to happen compared with the standard normal distribution. Understanding Gaussian copulas allows you to understand other more elaborated copulas.

The variable D below represents the density function for the normal distribution. As we said before, the density is basically how frequent is a given value of x in a normal distribution, so it helps to draw the typical bell-shape of the normal distribution function. Finally, P is the cumulative probability function of the normal distribution, it is equivalent to the function $N(\cdot)$ in Merton's model.

Now, let's apply the `dnorm` and `pnorm` functions for all possible values of x, not only in a few (11) as we did before. This will allow us to characterize a normal distribution function graphically for all possible x values. To do this, we simply create a new x that has 8,001 values, that would be enough.

```
# Now, x has 8001 values, this is ok to do some continuous plots.
x.theo <- seq(-4, 4, 0.001)
D <- dnorm(x.theo)
P <- pnorm(x.theo)
```

The red area represents the probability of default. This representation requires to calculate this area, and this can be easily done by the $N(\cdot)$ function. Graphically:

```
plot(x.theo, D, type = "l", lwd = 2,
     ylab = "Density function: dnorm(x)", xlab = "x")
polygon(c(x.theo[x.theo < 0], 0), c(D[x.theo < 0], 0), col = "red")
legend("topleft", legend = c("Here, the red area is the
middle of the bell-shaped curve.
Smaller area means lower
```

```
probability of default.

The probability of default
at x=0 is N(0)=50%

pnorm(0)=0.5
qnorm(0.5)=0"),
bg = "white", bty = "n", cex = 0.7)
abline(v = 0, col = "black")
abline(h = dnorm(0), lty = 2)
```

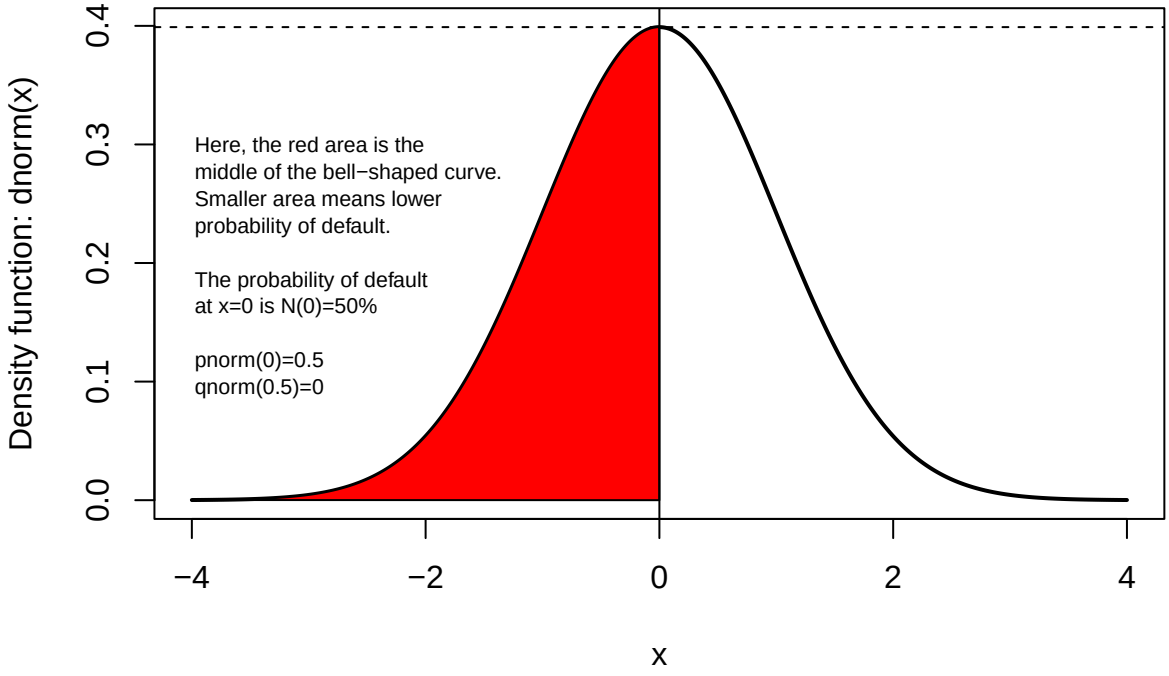


Figure 3.1.2: The red area represents the probability of default.

A good performing firm, with a $x = -2$, imply a low probability of default.

```
plot(x.theo, D, type = "l", lwd = 2, ylab = "Density function: dnorm(x)",
     xlab = "x")
polygon(c(x.theo[x.theo < -2], -2), c(D[x.theo < -2], 0), col = "red")

legend("topright", legend = c("Smaller area means
lower probability of default.

The probability of default
at x=-2 is N(-2)=2.275%

pnorm(-2)=0.02275013
qnorm(0.02275013)=-2"),
bg = "white", bty = "n", cex = 0.7)
abline(v = -2, col = "black")
```

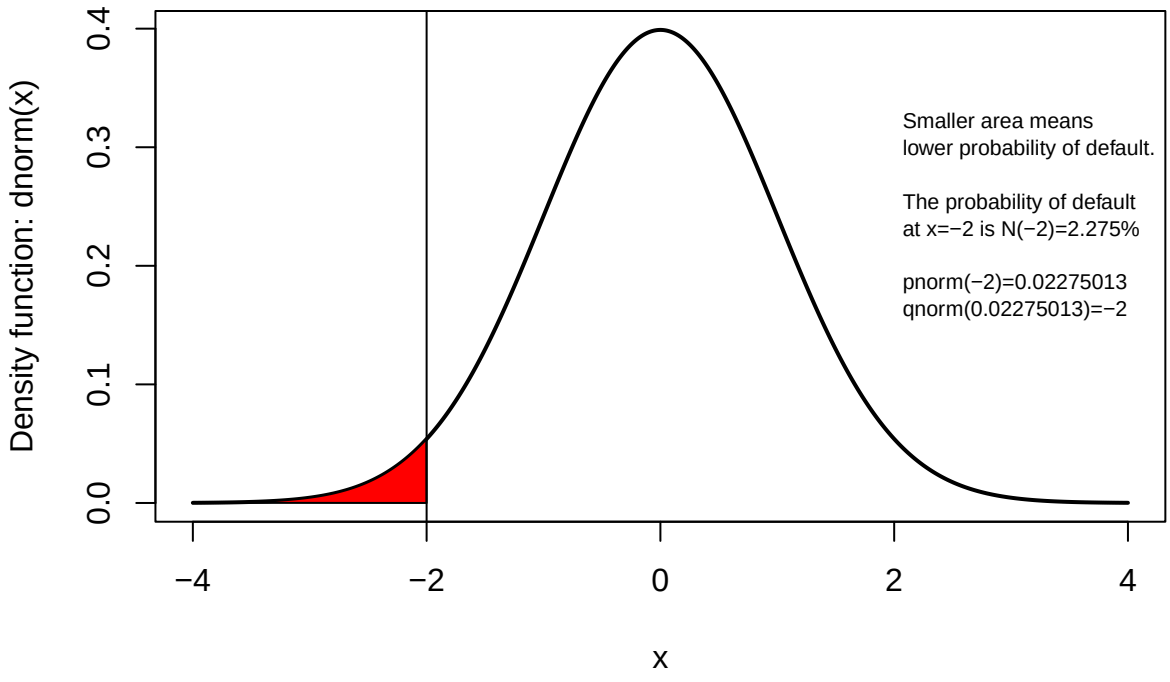


Figure 3.1.3: Low probability of default.

And this is how a bankrupt firm looks like:

```
plot(x.theo, D, type = "l", lwd = 2, ylab = "Density function: dnorm(x)",
     xlab = "x")
polygon(c(x.theo[x.theo < 4], 4), c(D[x.theo < 4], 0), col = "red")

legend("topleft", legend = c("The probability of default
at x=4 is N(4)=99.996%

pnorm(4)=0.9999683
qnorm(0.9999683)=4"),
bg = "white", bty = "n", cex = 0.8)
abline(v = 4, col = "black")
```

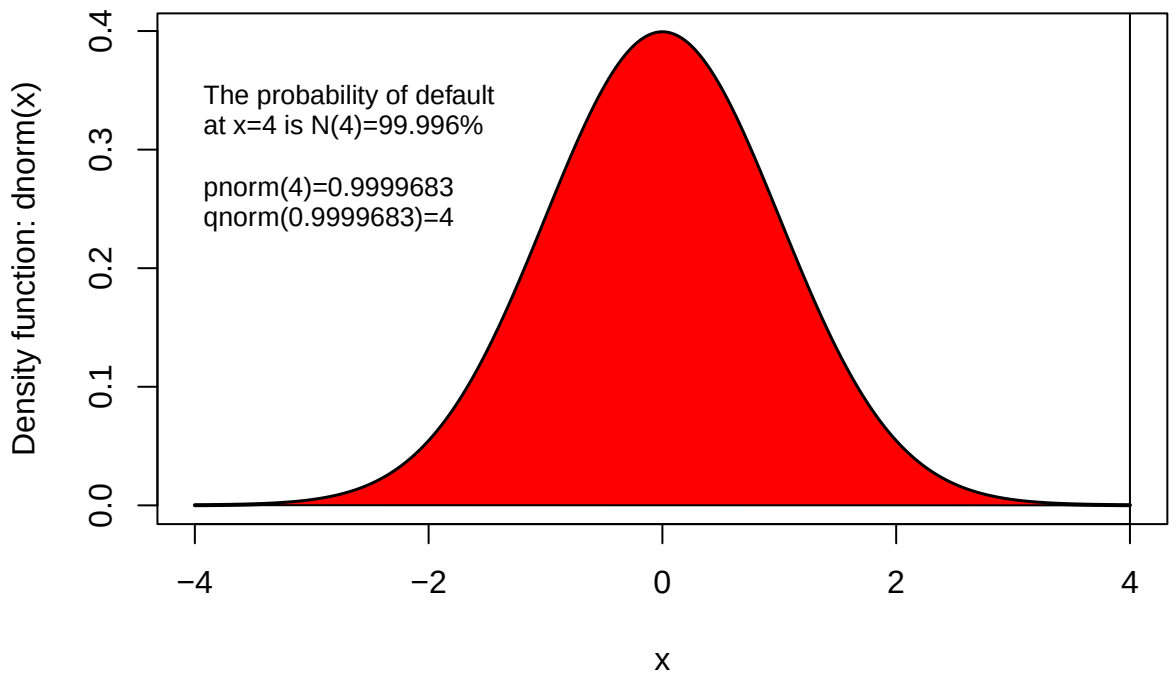


Figure 3.1.4: Imminent default.

Instead of a density function D, we can plot the cumulative probability distribution P. Now,

we do not need the $N(\cdot)$ function as the vertical axis represents the probability of default.

```
plot(x.theo, P, type = "l", lwd = 2,
     ylab = "Cumulative probability function: pnorm(x)", xlab = "x")
abline(h = 0.5, lty = 2)
abline(v = 0, lty = 2)
```

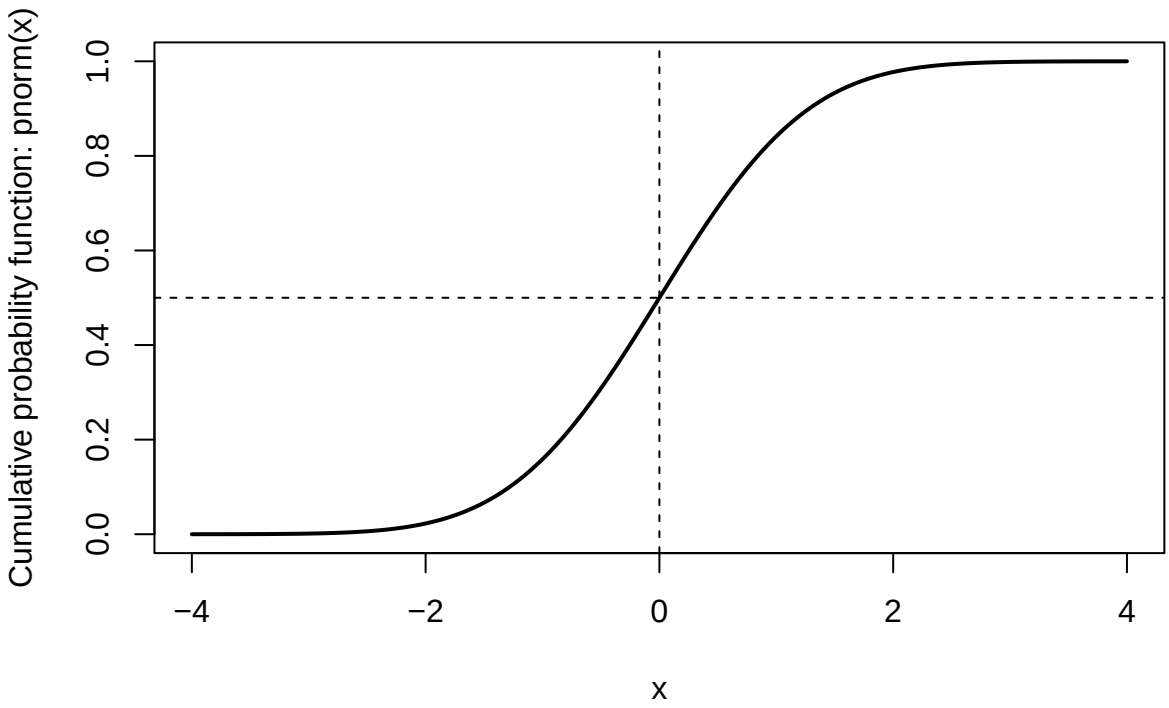


Figure 3.1.5: The higher the x, the higher the probability of default.

Next section requires a good understanding of the example 24.6 in Hull's textbook.

3.2 Introduction to example 24.6.

Here, we start analyzing Hull's example 24.6. We do not analyze the 10 firms yet as in the original example. We first make some sense about the data, the relevant analysis, the logic, and the model basics before dealing with the full features in example 24.6.

Recall that, `pnorm` leads to a probability, whereas `qnorm` leads to a value of x. Here are

some examples taken directly from the textbook example, where the cumulative probabilities of default of 1%, 3%, 6%, 10% and 15% are taken as given for the maturities of 1, 2, 3, 4 and 5 years respectively. To implement the model, we take this information to derive the corresponding x values:

```
pd <- c(0.01, 0.03, 0.06, 0.1, 0.15) # Probabilities of default per year.
x.y <- qnorm(pd) # Transform probabilities of default into x values.
ans <- data.frame(1:5, pd, x.y) # Gather the results.
colnames(ans) <- c("year", "pd", "x (values given in Hull)")
kable(ans, caption = "Main parameters.")
```

Table 3.2.1: Main parameters.

year	pd	x (values given in Hull)
1	0.01	-2.326348
2	0.03	-1.880794
3	0.06	-1.554774
4	0.10	-1.281552
5	0.15	-1.036433

Note that the example assumes that the probability of default increases as we consider a longer maturity (from 1 year to 5 years). Probabilities above are cumulative, so they go from year 0 to year 1, from year 0 to year 2 and so on. To calculate the probability of default during a specific year we need to calculate the differences. In particular:

```
ans <- data.frame(diff(pd))
colnames(ans) <- c("PD")
rownames(ans) <- c("x.y1 to x.y2", "x.y2 to x.y3",
                   "x.y3 to x.y4", "x.y4 to x.y5")
kable(ans, caption = "PD at specific years.")
```

Table 3.2.2: PD at specific years.

	PD
x.y1 to x.y2	0.02
x.y2 to x.y3	0.03
x.y3 to x.y4	0.04

	PD
x.y4 to x.y5	0.05

This is how we can illustrate the case of a probability of default of 15% in 5 years. The green area represents the 15% of the whole area below the bell-shaped curve.

```
plot(x.theo, D, type = "l", col = 'black', lwd = 3, ylim = c(0, 0.4),
     xlab = "This could represent a firm's performance measure.
The higher, the worst performance as it accumulates more prob. of default.",
     ylab = "Density: dnorm(x)")
abline(h = 0, lty = 2)
polygon(c(x.theo[x.theo < x.y[5]], x.y[5]),
        c(D[x.theo < x.y[5]], 0), col = "green")

legend("topright", legend=c(
"pd(5years)=15%

The left green area
represents 15% of the
whole bell-shape.

N^-1(0.15)=-1.036433
N(-1.036433)=0.15"),
bg = "white", pch = 19, cex = 0.8, bty = "n", col = "green")
```

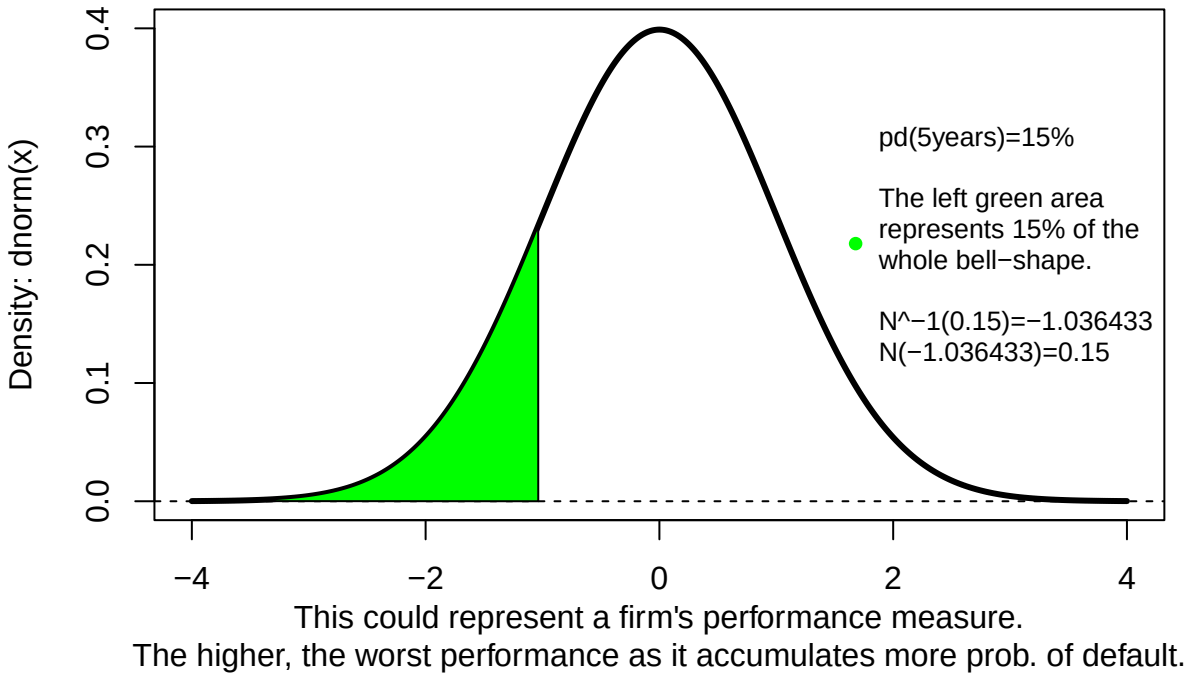


Figure 3.2.1: Gaussian density distribution function.

A complementary view is the cumulative probability function. Let's illustrate the same case: a probability of default of 15% in 5 years. In this case we do not need to calculate the area since the y-axis already represents the probability.

```
plot(x.theo, P, type = "l", col = 'black', lwd = 3, ylim = c(0, 1),
     xlab = "This could represent a firm's performance measure.
The higher, the worst performance as it accumulates more prob. of default.",
     ylab = "N(x) is a cumulative probability")
abline(h = 0, lty = 2)
abline(h = 1, lty = 2)
lines(seq(-5, x.y[5], length.out = 2), rep(pnorm(x.y[5]), 2),
      col = "green", lwd = 3)
lines(rep(x.y[5], 2), seq(0, pnorm(x.y[5]), length.out = 2),
      col = "green", lwd = 3)
```

```
points(x.y[5], 0.15, pch = 19, col = "green", cex = 2)
legend("right", legend=c(
"pd(5years)=15%

N^-1(0.15)=-1.036433
N(-1.036433)=0.15"),
bg = "white", pch = 19, cex = 1, bty = "n", col = "green")
```

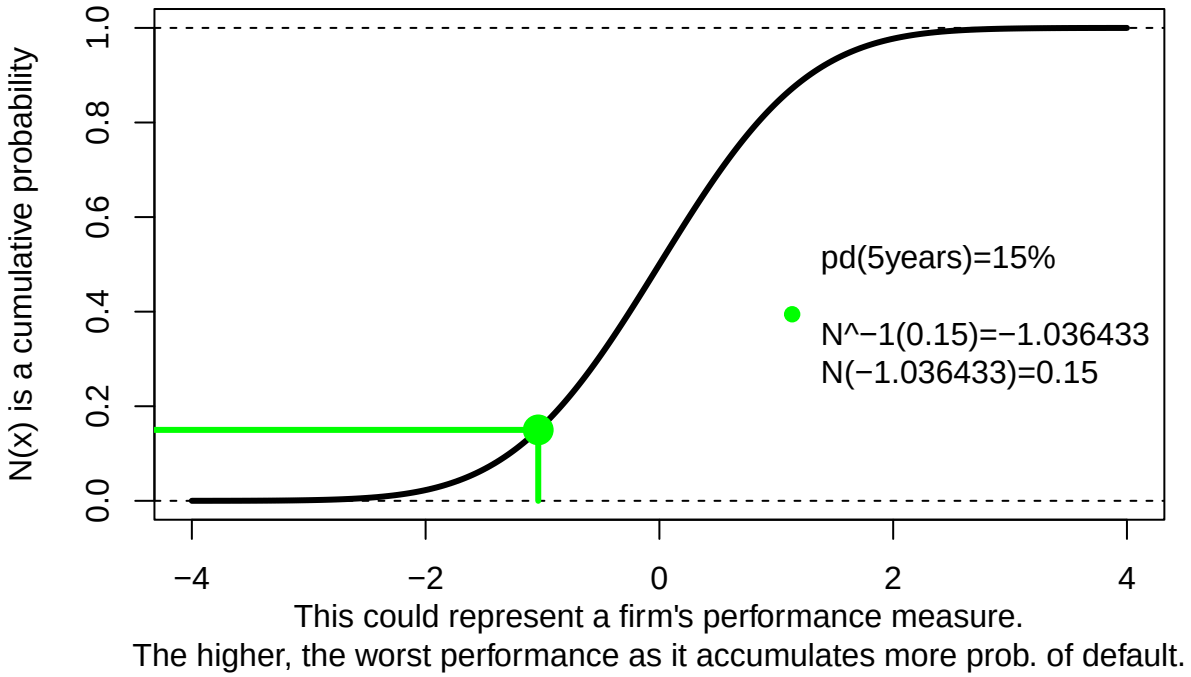


Figure 3.2.2: Gaussian probability distribution function.

A closer view to the figure above to see the 3-year and 5-year cases:

```
plot(x.theo, P, type = "l", col = 'black', lwd = 5, ylim = c(0, 0.22),
     xlim = c(-4, 0), ylab = "N(x) is a cumulative probability",
     xlab = "This could represent a firm's performance measure.
The higher, the worst performance as it accumulates more prob. of default.")
```

```
abline(h = 0, lty = 2)
abline(h = 1, lty = 2)
lines(seq(-5, x.y[3], length.out= 2), rep(pnorm(x.y[3]), 2),
      col = "purple", lwd = 3, lty = 2)
lines(rep(x.y[3], 2), seq(0, pnorm(x.y[3]), length.out = 2),
      col = "purple", lwd = 3, lty = 2)
lines(seq(-5, x.y[5], length.out = 2), rep(pnorm(x.y[5]), 2),
      col = "green", lwd = 3, lty = 2)
lines(rep(x.y[5], 2), seq(0, pnorm(x.y[5]), length.out = 2),
      col = "green", lwd = 3, lty = 2)
points(x.y[3], 0.06, pch = 19, col = "purple", cex = 2)
points(x.y[5], 0.15, pch = 19, col = "green", cex = 2)
legend("topleft", legend=c("pd(5years)=15%: N(-1.036433)=0.15",
                           "pd(3years)=6%: N(-1.554774)=0.06"),
pch = 19, col = c("green", "purple"), bg = "white", cex = 1, bty = "n")
```

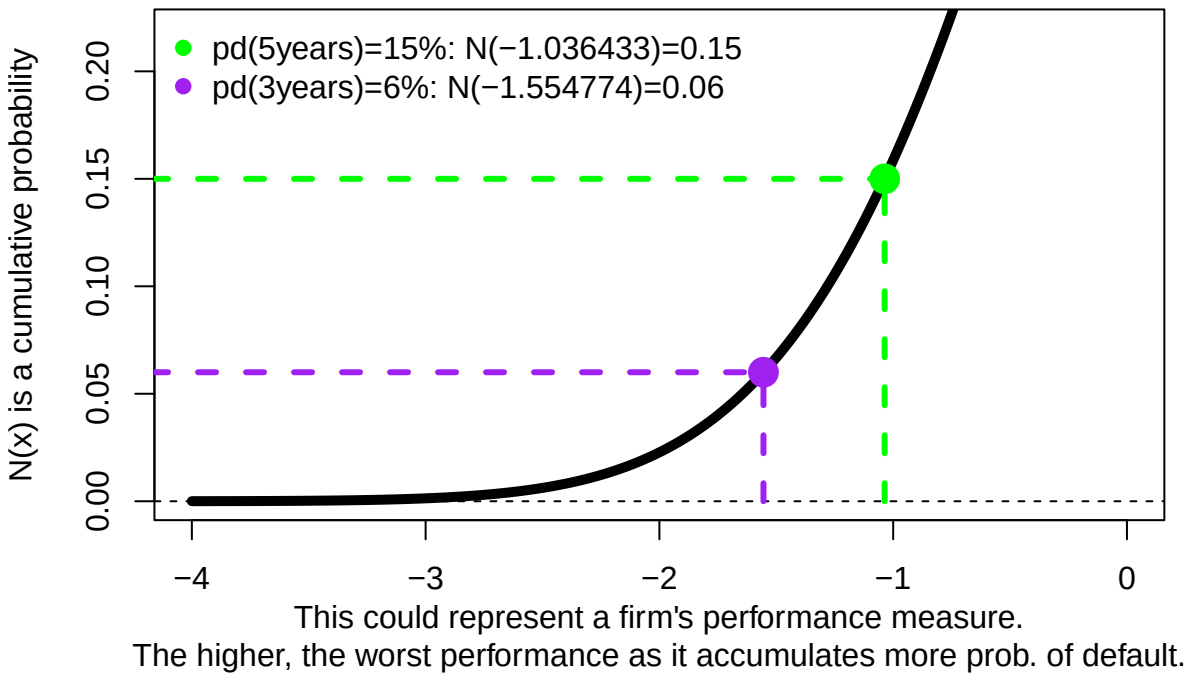


Figure 3.2.3: Gaussian probability distribution function: Zoom version.

The Gaussian copula approach is a method that takes the Gaussian distribution function to match the credit risk profile of firms.

3.3 One firm.

Now let's use a simulation approach instead of a theoretical approach. This is, instead of generating continuous values of x from -4 to 4, we simulate many x values (10,000 in this case) that follow a standard normal distribution function using the `rnorm` function. The simulation approach is useful especially when we are interested in replicating what happens in real-life situations because we can replicate the observed distribution many times and this facilitates the analysis. In other words, `x.theo` was used before to characterize a *perfect* normal distribution. Now, we incorporate `x.sim` that behaves as a normal distribution. These are now simulated values that follow a normal distribution, this means that we allow for some error or deviation with respect to the *perfect* normal distribution analyzed before.

Note that the probabilities of default per maturity in the simulated approach are close to the values of the previous section. In particular, 0.01 is equivalent to 0.0102, and 0.03 is equivalent to 0.0307. They do not match exactly simply because we are comparing theoretical versus simulated probabilities.

```
N <- 10000 # Number of simulated values.
set.seed(130575) # Reproducibility.
x.sim <- rnorm(N, 0, 1) # Simulation.
```

```
# Function to calculate proportions that we understand as probabilities.
prop <- function(x) {
  ans <- length(x.sim[x.sim <= x]) / N
  }
pd.sim <- mapply(prop, x.y) # Apply the function.
ans <- data.frame(x.y, pd, pd.sim)
colnames(ans) <- c("x", "pd.theo", "pd.sim")
rownames(ans) <- c("y1", "y2", "y3", "y4", "y5")
kable(ans, caption = "Theoretic versus simulated probabilities of default.")
```

Table 3.3.1: Theoretic versus simulated probabilities of default.

	x	pd.theo	pd.sim
y1	-2.326348	0.01	0.0102
y2	-1.880794	0.03	0.0307
y3	-1.554774	0.06	0.0629
y4	-1.281552	0.10	0.1008
y5	-1.036433	0.15	0.1480

Let's view the results of the simulated approach. First in a histogram.

```
# Some parameters we need to plot.
L <- c(-4, 4) # axis limits.
colors2 <- c("blue", "red", "purple", "pink", "green")
legend2 = c("pd(1year)=1.02%: x<=-2.326348",
"pd(2years)=3.07%: x<=-1.880794", "pd(3years)=6.29%: x<=-1.554774",
"pd(4years)=10.08%: x<=-1.281552", "pd(5years)=14.8%: x<=-1.036433")
# The histogram.
```

```
hist(x.sim, 500, xlim = L, ylim = c(0, 100), main = NULL, xlab = "x
This could represent a firm's simulated performance measure")
abline(h = 0, lty = 2)
abline(v = x.y[1], lwd = 3, col = "blue")
abline(v = x.y[2], lwd = 3, col = "red")
abline(v = x.y[3], lwd = 3, col = "purple")
abline(v = x.y[4], lwd = 3, col = "pink")
abline(v = x.y[5], lwd = 3, col = "green")
legend("topright", legend = legend2, bg = "white",
       text.col = colors2, cex = 0.7)
```

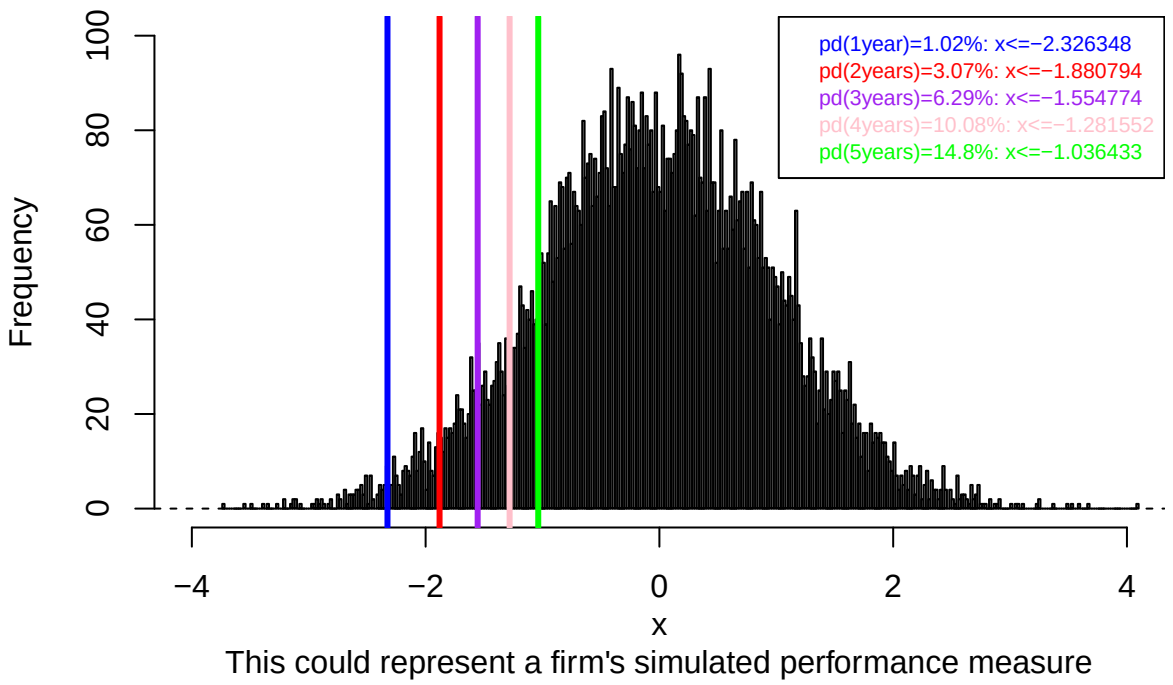


Figure 3.3.1: Simulated Gaussian probability distribution function. Somewhat different with respect to the theoretical.

The area at the left hand side of each colored line represents the cumulative probability of default just as we explained before. In the same way, the area between two colored lines

represents the probability of default in a specific period of time.

Now let's see all the simulated data at once.

```
plot(x.sim, ylab = "One firm performance", pch = ".",
     ylim = c(-4, 7),
     xlab = "10,000 simulated performance data")
abline(h = x.y[1], lwd = 2, col = "blue")
abline(h = x.y[2], lwd = 2, col = "red")
abline(h = x.y[3], lwd = 2, col = "purple")
abline(h = x.y[4], lwd = 2, col = "pink")
abline(h = x.y[5], lwd = 2, col = "green")
abline(v = 0, lty = 2)
abline(v = 10000, lty = 2)
legend("topright", legend = legend2, bg = "white",
       text.col = colors2, cex = 0.8)
```

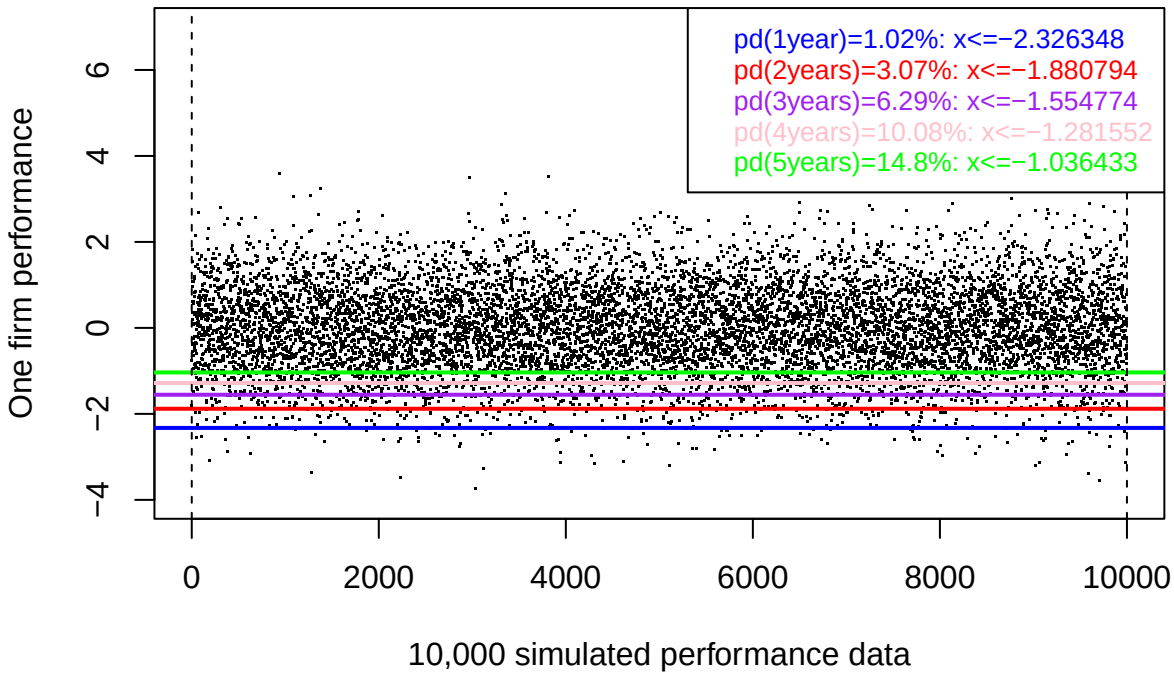


Figure 3.3.2: An alternative view.

We normally conduct a simulation approach because we might adapt the distribution function parameters to match what we see in the real life situations. The simulation approach allows us to have such flexibility.

3.4 Two firms.

Now consider the case in which we have two firms instead of one. The main difference now is that instead of simulating one firm we need two. Moreover, each firm follows a standard normal distribution function and both of them are correlated by a given correlation value so the firms are not independent. If they are not independent, then what happens to one firm has some impact on what happens to the other. In this case, we assume 0.2 as a correlation value. The case of two firms is not the one presented in Hull's example but it can help us to visualize how the Gaussian copula approach works in a two-dimension plot.

The simulation of both firms' performance measures is `x2`.

```
m <- 2 # number of firms
n <- 10000 # number of simulations
rho_pos02 <- 0.2 # correlation
corr_pos02 <- matrix(rep(rho_pos02, m * m), m, m) # correlation matrix
diag(corr_pos02) <- 1
set.seed(130575)
x2 <- mvrnorm(n, mu = rep(0, m), Sigma = corr_pos02)
x2 <- data.frame(x2)
colnames(x2) <- c("Firm1", "Firm2")
```

The matrix `x2` length is 10,000 for each firm. In other words, we have 10,000 observations of the performance measure or z-score for two firms that are related. This matrix is big, but we can visualize the header (the first six observations).

```
kable(head(x2), caption = "Firm's performance.", row.names = TRUE)
```

Table 3.4.1: Firm's performance.

	Firm1	Firm2
1	0.0880039	-2.8671108
2	0.0092849	-1.1735732
3	0.0323809	-0.5854275
4	1.5547886	-0.0630241

	Firm1	Firm2
5	0.9679767	-0.3977597
6	0.7751329	0.8847116

Remember the cumulative probabilities of default are 1%, 3%, 6%, 10% and 15% for the maturities of 1, 2, 3, 4 and 5 years respectively. How do we extract those cases in which both firms will default in 5 years? In rows 15, 53, 61 and so on both firms default at the same time. Note that in all cases the x values are indeed below -1.036433.

```
# These names are going to be useful later.
n.year <- c("year 1", "year 2", "year 3", "year 4", "year 5")
n.pd <- c("pd.y1", "pd.y2", "pd.y3", "pd.y4", "pd.y5")
n.f <- c("Firm1", "Firm2", "Firm1 default?", "Firm2 default?")

# Function to calculate cases in which firms default and probabilities.
fun.X <- function(x) {
  both <- x2[x2$Firm2 < x & x2$Firm1 < x, ] # both default.
  atleast1 <- x2[x2$Firm2 < x | x2$Firm1 < x, ] # at least one firm default.
  onlyfirm1 <- x2[x2$Firm2 > x & x2$Firm1 < x, ] # only firm 1 default.
  onlyfirm2 <- x2[x2$Firm1 > x & x2$Firm2 < x, ] # only firm 2 default.
  onlyone <- x2[(x2$Firm2 < x & x2$Firm1 > x | # only one firm default.
                   x2$Firm2 > x & x2$Firm1 < x),]
  none <- x2[x2$Firm2 > x & x2$Firm1 > x, ] # no firm default.
# Gather results and probabilities in a list.
ans <- list(both = both, atleast1 = atleast1, onlyfirm1 = onlyfirm1,
  onlyfirm2 = onlyfirm2, onlyone = onlyone, none = none,
  both.pd = (nrow(both) / n), atleast1.pd = (nrow(atleast1) / n),
  onlyfirm1.pd = (nrow(onlyfirm1) / n), onlyfirm2.pd = (nrow(onlyfirm2)/ n),
  onlyone.pd = (nrow(onlyone) / n), none.pd = (nrow(none) / n))
}
# X has all the relevant results for x2.
X <- mapply(fun.X, x.y)
```

See the cases in which both firms default in 5 years.

```
# Extract "both" cases, year 5.
both <- data.frame(X[["both", 5]], X[["both", 5]] < x.y[5])
colnames(both) <- n.f
kable(head(both), caption = "Cases in which both firms default in 5 years.",
      row.names = TRUE)
```

Table 3.4.2: Cases in which both firms default in 5 years.

	Firm1	Firm2	Firm1 default?	Firm2 default?
15	-1.123330	-1.783569	TRUE	TRUE
53	-2.724912	-1.235614	TRUE	TRUE
61	-1.968789	-1.516563	TRUE	TRUE
108	-2.121617	-1.807062	TRUE	TRUE
156	-1.559930	-1.132879	TRUE	TRUE
157	-1.194204	-2.120824	TRUE	TRUE

In total, we have 343 cases in which both firms default at the same time. The first case is number 15, the second 53, the third 61 and so on. It is easy to know the total cases if we count the number of rows.

How do we extract those cases in which at least one firm will default in 5 years? This is, only firm 1, only firm 2 and even both at the same time. This is a less strict condition so we would expect to have more cases to match this new criteria compared with `both`. Note that in all cases at least one one firm is indeed below -1.036433. In row 1, 2 and 10 firm 2 defaults. In row 15 both firms default. In row 18 and 20 firm 1 and firm 2 default respectively.

```
atleast1 <- data.frame(X[["atleast1", 5]], X[["atleast1", 5]] < x.y[5])
colnames(atleast1) <- n.f
kable(head(atleast1), caption = "Cases in which at least one firm default.",
      row.names = TRUE)
```

Table 3.4.3: Cases in which at least one firm default.

	Firm1	Firm2	Firm1 default?	Firm2 default?
1	0.0880039	-2.8671108	FALSE	TRUE
2	0.0092849	-1.1735732	FALSE	TRUE
10	1.4475216	-1.1512274	FALSE	TRUE

	Firm1	Firm2	Firm1 default?	Firm2 default?
15	-1.1233296	-1.7835692	TRUE	TRUE
18	-1.5829965	-0.7174298	TRUE	FALSE
20	-0.9221666	-1.9982869	FALSE	TRUE

Now we have 2,660 cases. Considerably more as the | restriction is less strict than the &. Let's see the cases in which only firm 1 defaults.

```
onlyfirm1 <- data.frame(X[["onlyfirm1", 5]], X[["onlyfirm1", 5]] < x.y[5])
colnames(onlyfirm1) <- n.f
kable(head(onlyfirm1), caption = "Cases in which only firm 1 default.")
```

Table 3.4.4: Cases in which only firm 1 default.

	Firm1	Firm2	Firm1 default?	Firm2 default?
18	-1.582996	-0.7174298	TRUE	FALSE
34	-2.842355	1.5400020	TRUE	FALSE
35	-2.090320	0.7431275	TRUE	FALSE
36	-1.600142	0.3082653	TRUE	FALSE
41	-2.193496	-0.8465008	TRUE	FALSE
49	-1.764711	-0.3914202	TRUE	FALSE

Only firm 2 defaults.

```
onlyfirm2 <- data.frame(X[["onlyfirm2", 5]], X[["onlyfirm2", 5]] < x.y[5])
colnames(onlyfirm2) <- n.f
kable(head(onlyfirm2), caption = "Cases in which only firm 2 default.")
```

Table 3.4.5: Cases in which only firm 2 default.

	Firm1	Firm2	Firm1 default?	Firm2 default?
1	0.0880039	-2.867111	FALSE	TRUE
2	0.0092849	-1.173573	FALSE	TRUE
10	1.4475216	-1.151227	FALSE	TRUE
20	-0.9221666	-1.998287	FALSE	TRUE
22	-0.1110555	-1.348605	FALSE	TRUE

	Firm1	Firm2	Firm1 default?	Firm2 default?
40	0.1058949	-1.752620	FALSE	TRUE

Only one firm default at year 5.

```
onlyone <- data.frame(X[["onlyone", 5]], X[["onlyone", 5]] < x.y[5])
colnames(onlyone) <- n.f
kable(head(onlyone), caption = "Cases in which only one firm default.")
```

Table 3.4.6: Cases in which only one firm default.

	Firm1	Firm2	Firm1 default?	Firm2 default?
1	0.0880039	-2.8671108	FALSE	TRUE
2	0.0092849	-1.1735732	FALSE	TRUE
10	1.4475216	-1.1512274	FALSE	TRUE
18	-1.5829965	-0.7174298	TRUE	FALSE
20	-0.9221666	-1.9982869	FALSE	TRUE
22	-0.1110555	-1.3486051	FALSE	TRUE

Lastly, when no firm defaults.

```
none <- data.frame(X[["none", 5]], X[["none", 5]] < x.y[5])
colnames(none) <- n.f
kable(head(none), caption = "Cases in which no firm default.")
```

Table 3.4.7: Cases in which no firm default.

	Firm1	Firm2	Firm1 default?	Firm2 default?
3	0.0323809	-0.5854275	FALSE	FALSE
4	1.5547886	-0.0630241	FALSE	FALSE
5	0.9679767	-0.3977597	FALSE	FALSE
6	0.7751329	0.8847116	FALSE	FALSE
7	1.3361498	1.0392145	FALSE	FALSE
8	1.3039572	1.5191303	FALSE	FALSE

Finally, probabilities.

```
both.pd <- t(data.frame(X["both.pd",]))
atleast1.pd <- t(data.frame(X["atleast1.pd",]))
onlyfirm1.pd <- t(data.frame(X["onlyfirm1.pd",]))
onlyfirm2.pd <- t(data.frame(X["onlyfirm2.pd",]))
onlyone.pd <- t(data.frame(X["onlyone.pd",]))
none.pd <- t(data.frame(X["none.pd",]))
ans <- data.frame(both.pd, atleast1.pd, onlyfirm1.pd, onlyfirm2.pd,
                  onlyone.pd, none.pd)
rownames(ans) <- n.year
kable(ans, caption = "Probabilities of default.")
```

Table 3.4.8: Probabilities of default.

	both.pd	atleast1.pd	onlyfirm1.pd	onlyfirm2.pd	onlyone.pd	none.pd
year 1	0.0003	0.0192	0.0103	0.0086	0.0189	0.9808
year 2	0.0019	0.0560	0.0282	0.0259	0.0541	0.9440
year 3	0.0066	0.1141	0.0561	0.0514	0.1075	0.8859
year 4	0.0176	0.1854	0.0864	0.0814	0.1678	0.8146
year 5	0.0343	0.2660	0.1210	0.1107	0.2317	0.7340

So interesting.

Note that 15% is the theoretical probability that one firm will default in 5 years. Here, this 15% is 12.1% for firm 1 and 11.07% for firm 2 when the data is simulated.

We can even perform a nice test to see that everything is alright. For example, this equation must hold: `onlyFirm1+onlyFirm2=onlyonefirm`. Substituting for the year 5: $0.121 + 0.1107 = 0.2317$. As you can see, everything is alright. This equation must hold as well: `atleast1-both=onlyonefirm`. Substituting for the year 5: $0.266 - 0.0343 = 0.2317$. As you can see, everything is alright.

Let's visualize all 10,000 cases. Each dot represents a couple of Firm1 and Firm2 x values and the dotted lines the threshold that represents the probability of default in 5 years.

```
par(pty = "s") # Figures are shown in a perfect square (not a rectangle).
plot(x2, pch = ".", cex = 0.8)
points(mean(x2[,1]), mean(x2[,2]), col = "red", pch = 19, cex = 1)
```

```
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("bottomright", legend = c(paste(nrow(x2))), bty = "n")
```

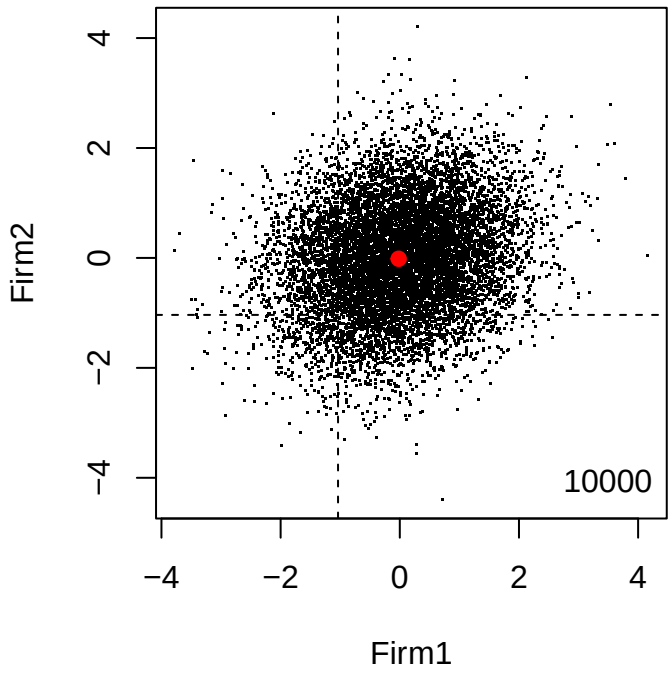


Figure 3.4.1: All 10,000 simulated cases.

These 10,000 observations are highly concentrated around the mean which is very close to zero $(-0.01883215, -0.0147721)$, note the red point. This can be also easily seen in the following density plot.

```
df <- tibble(x2)
par(pty = "s")
ggplot(df, aes(x = x2$Firm1, y = x2$Firm2)) +
  stat_density2d(aes(fill = ..density..), contour = F,
                 geom = 'tile') +
  scale_fill_viridis()+
```

```
  coord_fixed()
```

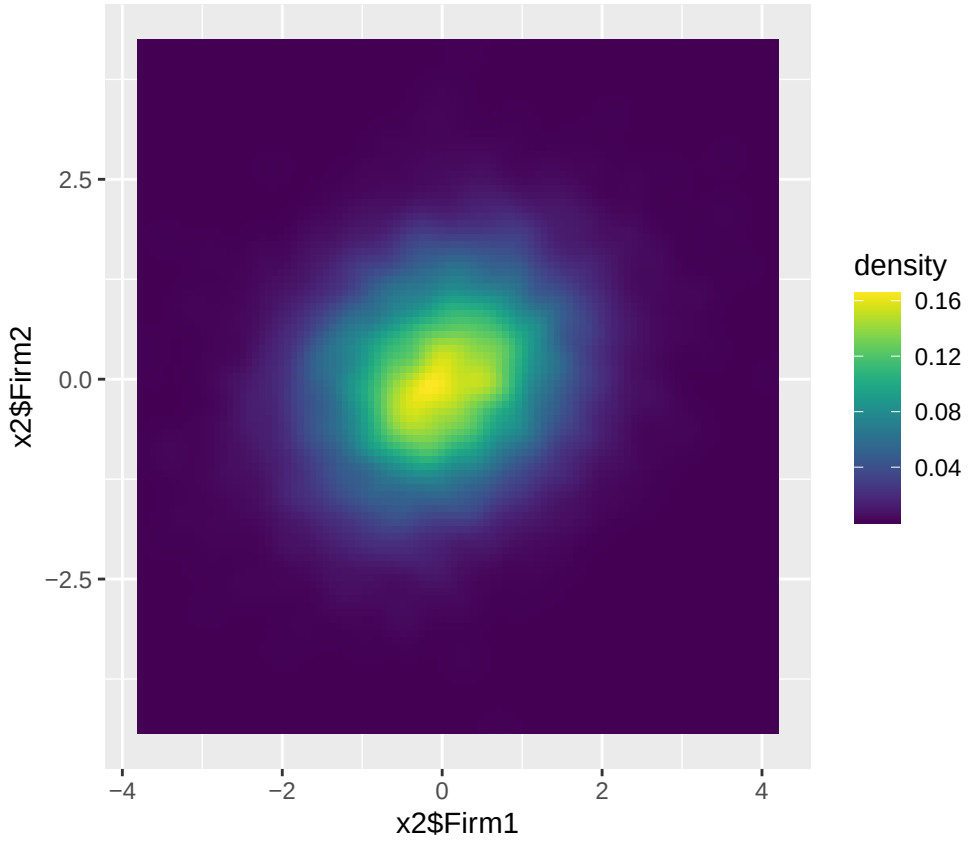


Figure 3.4.2: All 10,000 simulated cases: A density view.

We can visualize the default cases. First, the case when both firms default at year 5.

```
par(pty = "s")
plot(X[["both", 5]], xlim  = L, ylim = L, pch = ".", cex = 0.8)
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("topright", legend = c(paste(both.pd[5]*n)), bty = "n")
```

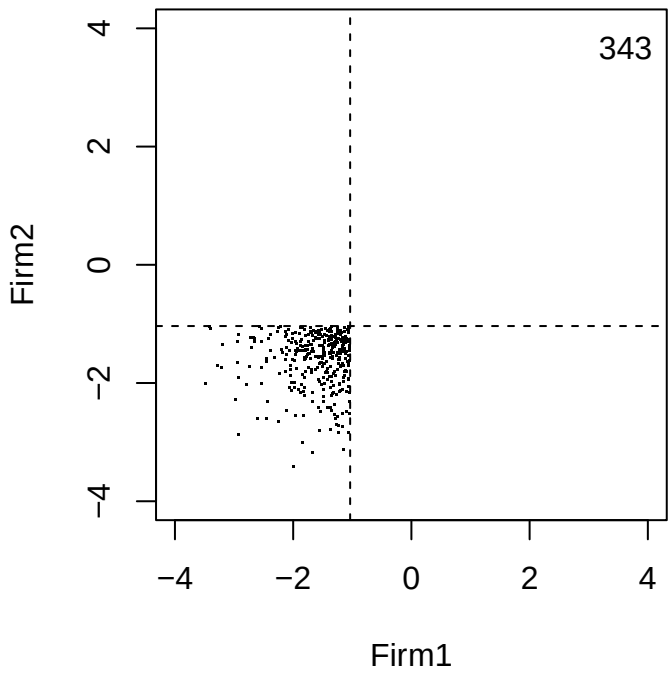


Figure 3.4.3: Both firms default at year 5.

The values within the plot represent the number of cases. Here, we have 343 times (out of 10,000) in which both firms default at the same time in 5 years. Note that this is a cumulative probability of default.

Now, the case in which at least one firm defaults in 5 years.

```
par(pty = "s")
plot(X[["atleast1", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8)
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("topright", legend = c(paste(atleast1.pd[5]*n)), bty = "n")
```

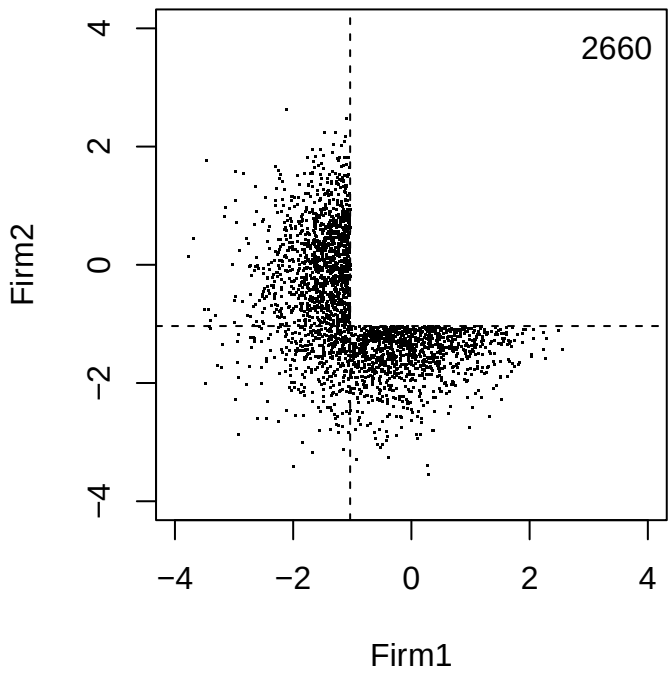


Figure 3.4.4: At least one firm defaults in 5 years.

Only firm 1 defaults in 5 years.

```
par(pty = "s")
plot(X[["onlyfirm1", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8)
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("topright", legend = c(paste(onlyfirm1.pd[5]*n)), bty = "n")
```

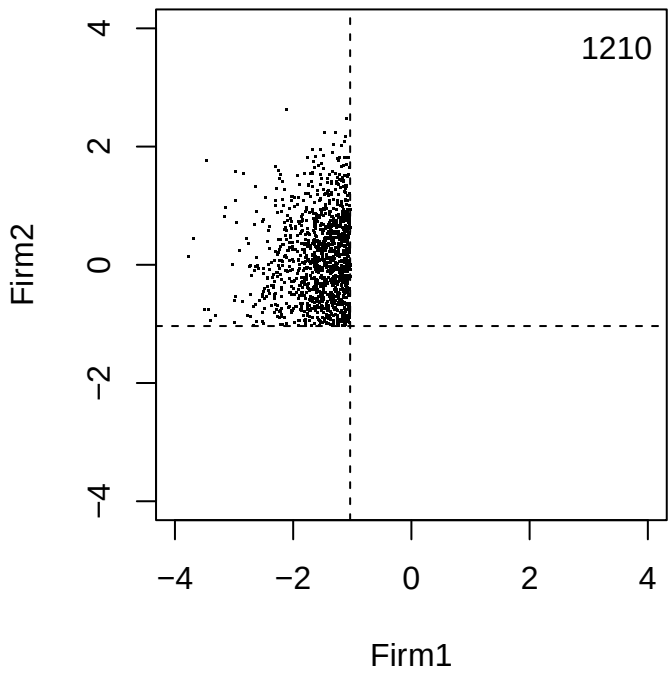


Figure 3.4.5: Only firm 1 defaults in 5 years.

Only firm 2 defaults in 5 years.

```
par(pty = "s")
plot(X[["onlyfirm2", 5]], xlim = L, ylim=L, pch = ".", cex = 0.8)
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("topright", legend = c(paste(onlyfirm2.pd[5]*n)), bty = "n")
```

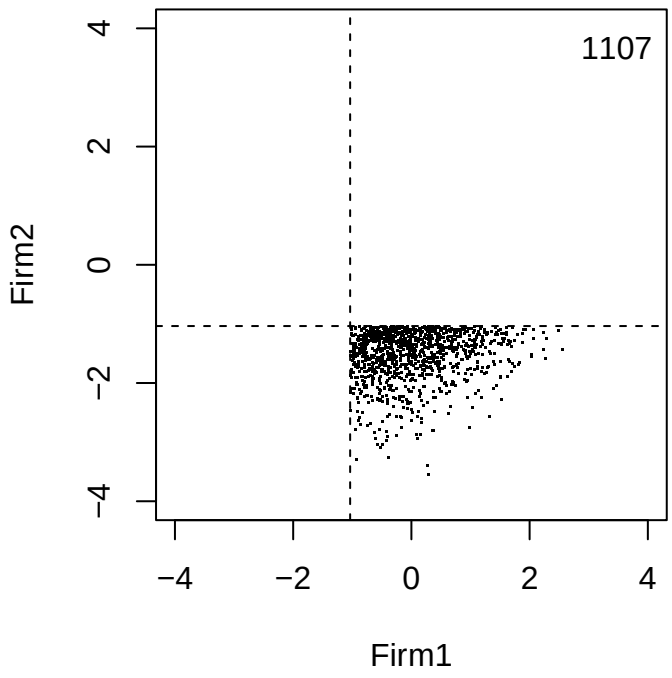


Figure 3.4.6: Only firm 2 defaults in 5 years.

This is the case in which only one firm defaults.

```
par(pty = "s")
plot(X[["onlyone", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8)
abline(h = x.y[5], lty =2)
abline(v = x.y[5], lty =2)
legend("topright", legend = c(paste(onlyone.pd[5]*n)), bty = "n")
```

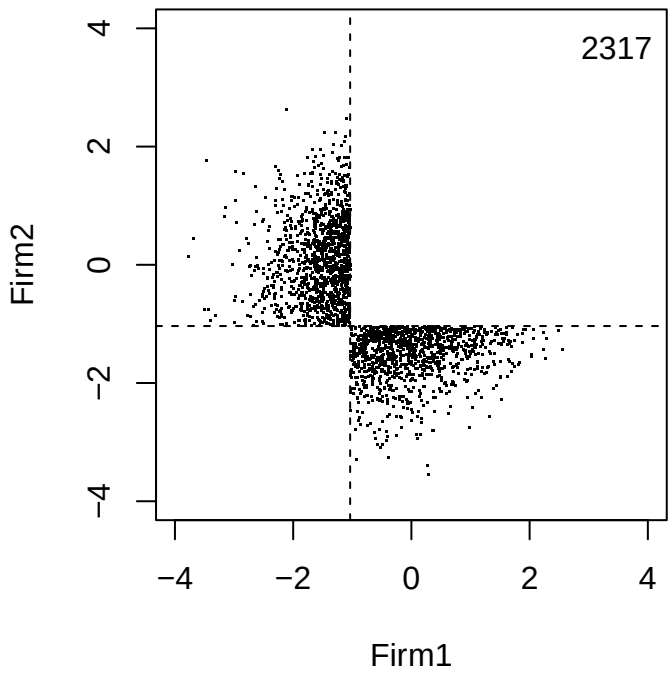


Figure 3.4.7: Only one firm defaults in 5 years.

This is the case in which no one firm defaults.

```
par(pty = "s")
plot(X[["none", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8)
abline(h = x.y[5], lty =2)
abline(v = x.y[5], lty =2)
legend("topright", legend = c(paste(none.pd[5]*n)), bty = "n")
```

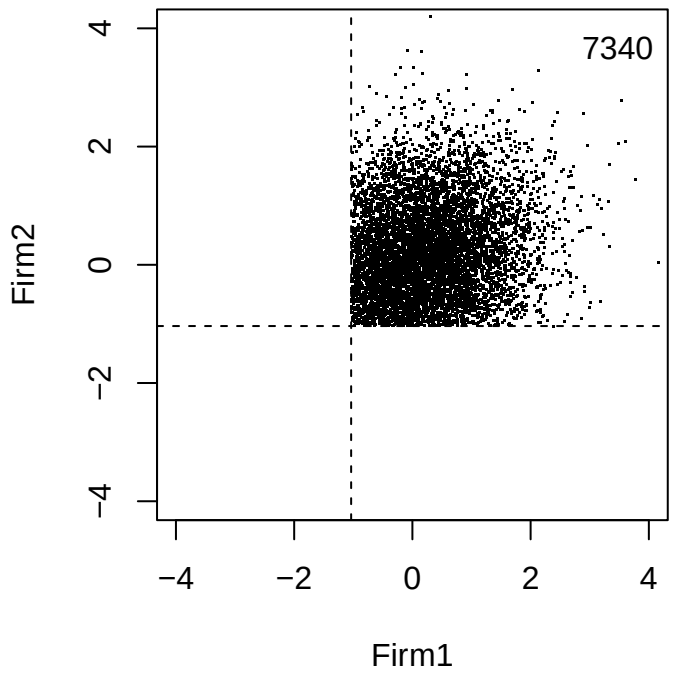


Figure 3.4.8: No firm defaults in 5 years.

It is a good idea to summarize all previous plots in one.

```
# none
par(mfrow = c(2, 3), oma = c(0, 0, 2, 0))
par(pty = "s")
plot(X[["none", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8,
     main = "None.")
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("bottomright", legend = c(paste(none.pd[5]*n)), bty = "n")
# both
par(pty = "s")
plot(X[["both", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8,
     main = "Both.")
```

```
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("topright", legend = c(paste(both.pd[5]*n)), bty = "n")
# atleast1
par(pty = "s")
plot(X[["atleast1", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8,
     main = "At least one.")
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("topright", legend = c(paste(atleast1.pd[5]*n)), bty = "n")
# onlyfirm1
par(pty = "s")
plot(X[["onlyfirm1", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8,
     main = "Only Firm1.")
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("topright", legend = c(paste(onlyfirm1.pd[5]*n)), bty = "n")
# onlyfirm2
par(pty = "s")
plot(X[["onlyfirm2", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8,
     main = "Only Firm2.")
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("topright", legend = c(paste(onlyfirm2.pd[5]*n)), bty = "n")
# onlyone
par(pty = "s")
plot(X[["onlyone", 5]], xlim = L, ylim = L, pch = ".", cex = 0.8,
     main = "Only one.")
abline(h = x.y[5], lty = 2)
abline(v = x.y[5], lty = 2)
legend("topright", legend = c(paste(onlyone.pd[5]*n)), bty = "n")
```

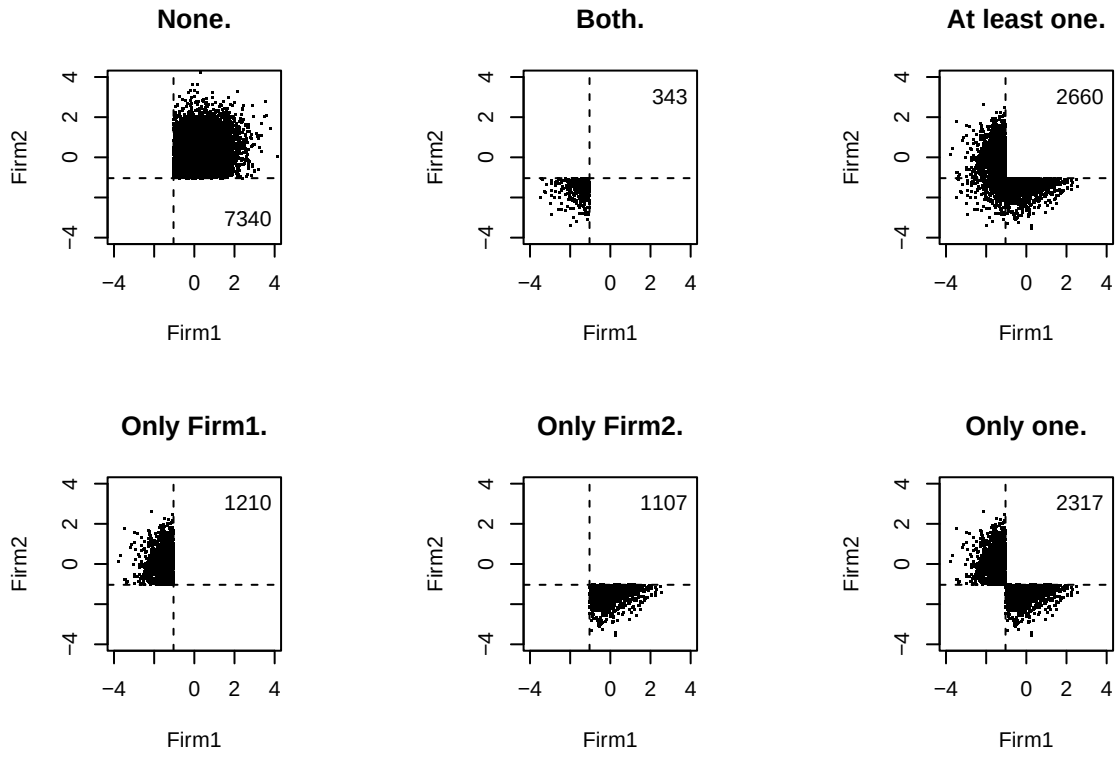


Figure 3.4.9: Which firm defaults at year 5?

It is interesting to compare two different correlation values. Here, we compare 0 versus 0.2.

```
m <- 2 # number of firms
n <- 10000 # number of simulations
rho_pos02 <- 0 # correlation
corr_pos02 <- matrix(rep(rho_pos02, m * m), m, m) # correlation matrix
diag(corr_pos02) <- 1
set.seed(130575)
X0 <- mvrnorm(n, mu = rep(0, m), Sigma = corr_pos02)
X0 <- data.frame(X0)
colnames(X0) <- c("Firm1", "Firm2")
```

Just as before, we create a function, evaluate it, and store results in `X0`.

```
fun.X0 <- function(x) {
  both <- X0[X0$Firm2 < x & X0$Firm1 < x, ]
  atleast1 <- X0[X0$Firm2 < x | X0$Firm1 < x, ]
  onlyfirm1 <- X0[X0$Firm2 > x & X0$Firm1 < x, ]
  onlyfirm2 <- X0[X0$Firm1 > x & X0$Firm2 < x, ]
  onlyone <- X0[(X0$Firm2 < x & X0$Firm1 > x |
                   X0$Firm2 > x & X0$Firm1 < x),]
  none <- X0[X0$Firm2 > x & X0$Firm1 > x, ]

ans <- list(both = both, atleast1 = atleast1, onlyfirm1 = onlyfirm1,
  onlyfirm2 = onlyfirm2, onlyone = onlyone, none = none,
  both.pd = (nrow(both) / n), atleast1.pd = (nrow(atleast1) / n),
  onlyfirm1.pd = (nrow(onlyfirm1) / n), onlyfirm2.pd = (nrow(onlyfirm2) /n),
  onlyone.pd = (nrow(onlyone) / n), none.pd = (nrow(none) / n)) }

X0 <- mapply(fun.X0, x.y)

none.pd0 <- data.frame(X0["none.pd",]) * n
both.pd0 <- data.frame(X0["both.pd",]) * n

onlyfirm1.pd0 <- data.frame(X0["onlyfirm1.pd",]) * n
onlyfirm2.pd0 <- data.frame(X0["onlyfirm2.pd",]) * n
```

A graphical analysis shows that in the case of 0.2 it is more likely that both firms default at the same time, and it is less likely that any firm default at the same time.

```
par(mfrow=c(1, 2), oma = c(0, 0, 2, 0))
par(pty = "s")
plot(X[["none", 5]], pch = ".", xlim = L, ylim = L,
     cex = 0.8, main = "Correlation=0.2")
points(X[["both", 5]], pch = ".", col = "red")
points(X[["onlyfirm1", 5]], pch = ".", col = "purple")
points(X[["onlyfirm2", 5]], pch = ".", col = "blue")
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("bottomleft", legend = c(paste(both.pd[5]*n)), bty = "n")
legend("topright", legend = c(paste(none.pd[5]*n)), bty = "n")
```

```
legend("topleft", legend = c(paste(onlyfirm1.pd[5]*n)), bty = "n")
legend("bottomright", legend = c(paste(onlyfirm2.pd[5]*n)), bty = "n")
par(pty = "s")
plot(X0[["none", 5]], pch = ".", xlim = L, ylim = L,
     cex = 0.8, main = "Correlation=0.")
points(X0[["both", 5]], pch = ".", col = "red")
points(X0[["onlyfirm1", 5]], pch = ".", col = "purple")
points(X0[["onlyfirm2", 5]], pch = ".", col = "blue")
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("bottomleft", legend = c(paste(both.pd0[5])), bty = "n")
legend("topright", legend = c(paste(none.pd0[5])), bty = "n")
legend("topleft", legend = c(paste(onlyfirm1.pd0[5])), bty = "n")
legend("bottomright", legend = c(paste(onlyfirm2.pd0[5])), bty = "n")
```

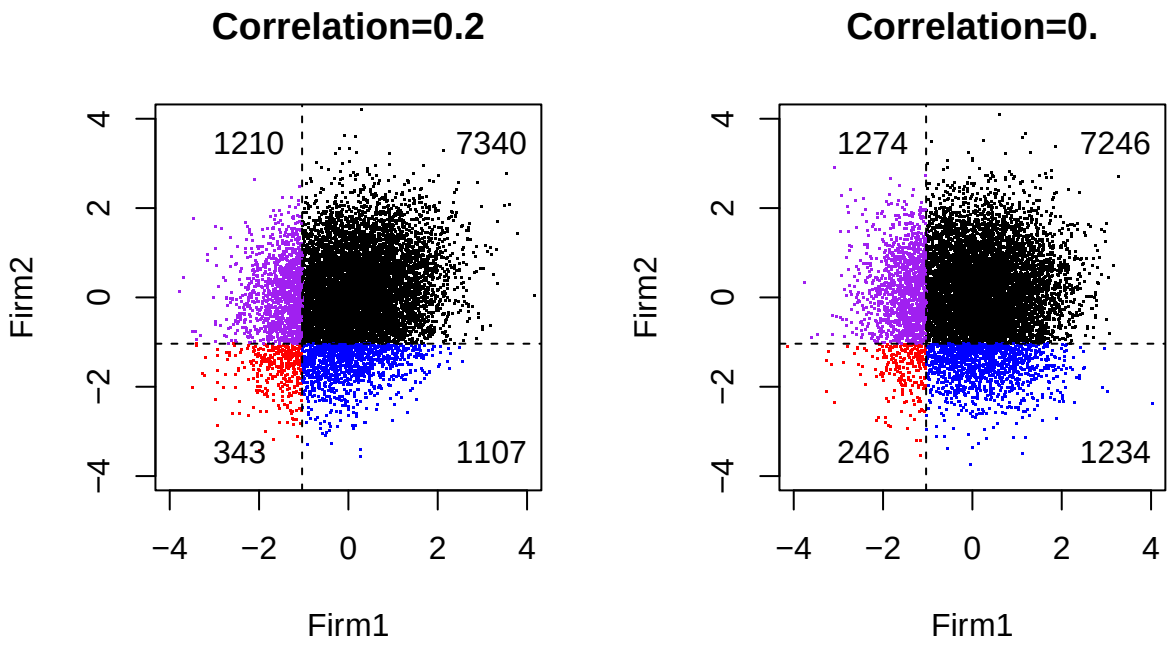


Figure 3.4.10: Cases per quadrant. Dotted lines corresponds to year 5.

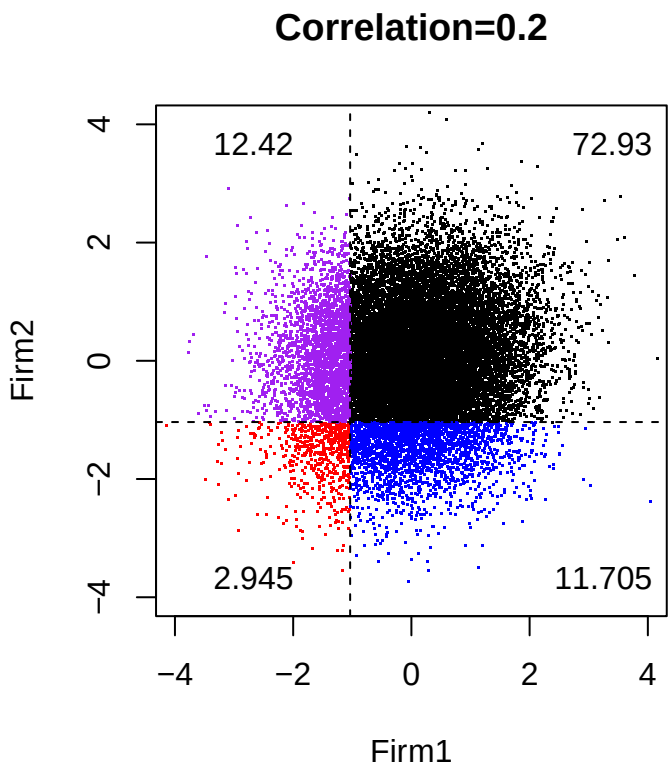
```
par(pty = "s")
```

Very interesting indeed.

```
m <- 2 # number of firms
n <- 10000 # number of simulations
rho_pos00 <- 0 # correlation
rho_pos02 <- 0.2
corr_pos00 <- matrix(rep(rho_pos00, m * m), m, m) # correlation matrix
corr_pos02 <- matrix(rep(rho_pos02, m * m), m, m) # correlation matrix
diag(corr_pos00) <- 1
diag(corr_pos02) <- 1
set.seed(130575)
X00 <- mvrnorm(n, mu = rep(0, m), Sigma = corr_pos00)
set.seed(130575)
X02 <- mvrnorm(n, mu = rep(0, m), Sigma = corr_pos02)
X00.02 <- data.frame(rbind(X00, X02))
colnames(X00.02) <- c("Firm1", "Firm2")

n=20000
fun.X0 <- function(x) {
  both <- X00.02[X00.02$Firm2 < x & X00.02$Firm1 < x, ]
  atleast1 <- X00.02[X00.02$Firm2 < x | X00.02$Firm1 < x, ]
  onlyfirm1 <- X00.02[X00.02$Firm2 > x & X00.02$Firm1 < x, ]
  onlyfirm2 <- X00.02[X00.02$Firm1 > x & X00.02$Firm2 < x, ]
  onlyone <- X00.02[(X00.02$Firm2 < x & X00.02$Firm1 > x |
                     X00.02$Firm2 > x & X00.02$Firm1 < x),]
  none <- X00.02[X00.02$Firm2 > x & X00.02$Firm1 > x, ]

ans <- list(both = both, atleast1 = atleast1, onlyfirm1 = onlyfirm1,
  onlyfirm2 = onlyfirm2, onlyone = onlyone, none = none,
  both.pd = (nrow(both) / n), atleast1.pd = (nrow(atleast1) / n),
  onlyfirm1.pd = (nrow(onlyfirm1) / n), onlyfirm2.pd = (nrow(onlyfirm2) /n),
  onlyone.pd = (nrow(onlyone) / n), none.pd = (nrow(none) / n)) }

X00.02 <- mapply(fun.X0, x.y)
```

```
none.pd0002 <- data.frame(X00.02["none.pd",]) * n
both.pd0002 <- data.frame(X00.02["both.pd",]) * n

onlyfirm1.pd0002 <- data.frame(X00.02["onlyfirm1.pd",]) * n
onlyfirm2.pd0002 <- data.frame(X00.02["onlyfirm2.pd",]) * n
```

```
par(pty = "s")
plot(X00.02[["none", 5]], pch = ".", xlim = L, ylim = L,
     cex = 0.8, main = "Correlation=0.2")
points(X00.02[["both", 5]], pch = ".", col = "red")
points(X00.02[["onlyfirm1", 5]], pch = ".", col = "purple")
points(X00.02[["onlyfirm2", 5]], pch = ".", col = "blue")
abline(v = x.y[5], lty = 2)
abline(h = x.y[5], lty = 2)
legend("bottomleft", legend = c(paste(both.pd0002[5]/n*100)), bty = "n")
legend("topright", legend = c(paste(none.pd0002[5]/n*100)), bty = "n")
legend("topleft", legend = c(paste(onlyfirm1.pd0002[5]/n*100)), bty = "n")
legend("bottomright", legend = c(paste(onlyfirm2.pd0002[5]/n*100)), bty = "n")
```



3.5 Ten firms.

The original Hull's example proposes a 10 firm case and here we implement this example following a simulation approach. First, we need a 10×10 correlation matrix to produce the new x values using a random multi-variate distribution algorithm. According to Hull's example the copula default correlations between each pair of companies is 0.2. The code below has the option to vary the default correlation given a uniform random distribution.

The new 10×10 correlation matrix is then:

```
# Create the correlation matrix.
m <- 10 # number of firms.
n <- 1000000 # number of simulations per firm.
x <- matrix(rep(0.2, m * m), m, m)
ind <- lower.tri(x)
x[ind] <- t(x)[ind]
diag(x) = 1
kable(x, caption = "Correlation matrix 0.2.")
```

Table 3.5.1: Correlation matrix 0.2.

1.0	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
0.2	1.0	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
0.2	0.2	1.0	0.2	0.2	0.2	0.2	0.2	0.2	0.2
0.2	0.2	0.2	1.0	0.2	0.2	0.2	0.2	0.2	0.2
0.2	0.2	0.2	0.2	1.0	0.2	0.2	0.2	0.2	0.2
0.2	0.2	0.2	0.2	0.2	1.0	0.2	0.2	0.2	0.2
0.2	0.2	0.2	0.2	0.2	0.2	1.0	0.2	0.2	0.2
0.2	0.2	0.2	0.2	0.2	0.2	0.2	1.0	0.2	0.2
0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	1.0	0.2
0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	1.0

Now, we can simulate the multivariate normal distribution. The variable `X10` length is 1,000,000 for each firm. This is big, but we can visualize the header of this variable. We choose a higher number of simulations because now we need 10 firms to meet a single constraint.

```
# Create the simulated cases.
set.seed(130575) # Reproducibility.
X10 <- mvrnorm(n, mu = rep(0, m), Sigma = x)
X10 <- data.frame(X10) # 10,000,000 observations.
kable(head(X10), caption = "10 firms' performance, 1,000,000 simulations.",
      digits = 3)
```

Table 3.5.2: 10 firms' performance, 1,000,000 simulations.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
1.541	0.747	2.270	0.546	1.500	0.106	1.587	1.269	0.453	-0.527
1.867	-0.840	0.272	-0.935	-0.323	0.688	-0.411	-0.642	2.869	1.432
-1.924	-0.058	1.678	1.840	-0.964	0.211	-0.277	0.704	0.197	0.482
-0.451	-0.997	-1.471	-0.565	0.227	0.429	-1.809	-0.783	-0.659	0.983
-1.105	0.219	0.191	1.552	-1.037	0.480	1.277	-1.878	-1.581	-0.066
-0.968	-0.284	0.309	-1.529	0.012	-1.413	-0.188	-0.939	-0.496	-0.174

How do we extract those cases in which all 10 firms will default in 5 years (at the same time)? Here are the first 6 of those cases. Note that in all cases the `X10` values are lower than -1.036433.

```
# Given that we have 10 firms, it is easier to use filter_all function.
# Although probably this could be simplified even further.
y5.all.2 <- filter_all(X10, all_vars(. < x.y[5]))
y4.all.2 <- filter_all(X10, all_vars(. < x.y[4]))
y3.all.2 <- filter_all(X10, all_vars(. < x.y[3]))
y2.all.2 <- filter_all(X10, all_vars(. < x.y[2]))
y1.all.2 <- filter_all(X10, all_vars(. < x.y[1]))
```

Let's analyze the case of 5 years.

```
kable(head(y5.all.2),
      caption = "Cases in which all 10 firms default in five years.",
      digits = 3)
```

Table 3.5.3: Cases in which all 10 firms default in five years.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
-1.793	-3.143	-1.945	-1.054	-3.511	-1.746	-1.789	-2.098	-1.998	-2.809
-1.423	-1.779	-1.209	-1.486	-1.304	-1.630	-1.455	-2.358	-1.238	-1.070
-2.272	-1.640	-2.152	-2.374	-1.943	-3.206	-1.372	-1.582	-2.682	-3.230
-3.229	-1.487	-1.443	-2.503	-1.582	-1.050	-1.202	-1.525	-1.608	-3.308
-1.172	-1.079	-1.508	-1.605	-2.093	-1.407	-1.972	-3.347	-1.878	-1.624
-2.925	-2.470	-1.483	-1.612	-1.452	-1.404	-2.710	-2.011	-1.296	-2.440

```
kable((head(y5.all.2) < x.y[5]), caption = "Check if all of them default.")
```

Table 3.5.4: Check if all of them default.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

How many cases are there?

```
nrow(y5.all.2)
```

```
## [1] 72
```

Which are those 72 cases?

```
# Here, I compare only firm 1 as if firm 1 defaults, then the rest default.
kable(matrix(which(X10[,1] %in% y5.all.2[,1]), 6, 9),
      caption = "Which of the 1,000,000 cases represent a default of all
      firms in five years?")
```

Table 3.5.5: Which of the 1,000,000 cases represent a default of all firms in five years?

16257	99432	171012	319557	413425	511731	643935	715625	796118
32298	101750	266761	353059	423852	527750	644895	727562	797265
57098	104810	297762	391778	437629	574643	647511	736041	808143
62188	118416	300925	402689	444338	597147	690749	745317	808624
74173	125947	311908	405804	492912	618391	700476	747221	813924
81982	132862	317946	406553	509996	622673	705705	749223	823963

In total, we only have 72 cases. This is, the 10 firms will default at the same time in 5 years in 72 out of 1,000,000 total cases. For the rest of the years, the cases are less frequent, in fact we have zero cases for year 1, 2 and 3.

How do we extract those cases in which at least one of the 10 firms will default?

```
y5.any.2 <- filter_all(X10, any_vars(. < x.y[5]))
y4.any.2 <- filter_all(X10, any_vars(. < x.y[4]))
y3.any.2 <- filter_all(X10, any_vars(. < x.y[3]))
y2.any.2 <- filter_all(X10, any_vars(. < x.y[2]))
```

```
y1.any.2 <- filter_all(X10, any_vars(. < x.y[1]))
```

Here are the first 6 cases for the 5-year default.

```
kable(head(y5.any.2), caption = "At least one firm default in five years.",
      digits = 3)
```

Table 3.5.6: At least one firm default in five years.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
-1.924	-0.058	1.678	1.840	-0.964	0.211	-0.277	0.704	0.197	0.482
-0.451	-0.997	-1.471	-0.565	0.227	0.429	-1.809	-0.783	-0.659	0.983
-1.105	0.219	0.191	1.552	-1.037	0.480	1.277	-1.878	-1.581	-0.066
-0.968	-0.284	0.309	-1.529	0.012	-1.413	-0.188	-0.939	-0.496	-0.174
-0.836	-0.828	0.982	0.375	-0.341	-1.382	-2.533	-1.268	-1.214	-1.069
-0.495	-1.382	-0.932	-0.625	1.327	-2.546	0.071	-2.605	-1.116	-1.340

```
kable((head(y5.any.2) < x.y[5]), caption = "Check which one(s) default.")
```

Table 3.5.7: Check which one(s) default.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	TRUE

How many cases are these?

```
nrow(y5.any.2)
```

```
## [1] 682148
```

In total, we have 682,148 cases. This is, at least one of 10 firms will default in 5 years in 682,148 of 1,000,000 cases.

And how to convert them into probabilities?

```
atleastone02 <- t(data.frame(nrow(y1.any.2), nrow(y2.any.2),
                 nrow(y3.any.2), nrow(y4.any.2), nrow(y5.any.2)))
all02 <- t(data.frame(nrow(y1.all.2), nrow(y2.all.2),
                 nrow(y3.all.2), nrow(y4.all.2), nrow(y5.all.2)))
res02 <- data.frame(all02 / n, atleastone02 / n)
rownames(res02) <- n.year
colnames(res02) <- c("All firms", "At least one")
kable(res02, caption = "Probabilities of default (10 firms, corr=0.2).")
```

Table 3.5.8: Probabilities of default (10 firms, corr=0.2).

	All firms	At least one
year 1	0.0e+00	0.087276
year 2	0.0e+00	0.226406
year 3	0.0e+00	0.386348
year 4	1.2e-05	0.543162
year 5	7.2e-05	0.682148

Let's see the difference when we assume a different correlation matrix. This case, the correlation vary randomly between 0.45 and 0.65.

```
m <- 10 # number of firms
n <- 1000000 # number of simulations
set.seed(130575)
x <- matrix(runif(m * m, 0.45, 0.65), m, m)
ind <- lower.tri(x)
x[ind] <- t(x)[ind]
diag(x) = 1
kable(x, caption = "Correlation between 0.45 and 0.65.", digits = 3)
```

Table 3.5.9: Correlation between 0.45 and 0.65.

1.000	0.622	0.597	0.622	0.530	0.517	0.518	0.490	0.455	0.578
0.622	1.000	0.523	0.594	0.465	0.501	0.558	0.560	0.460	0.513

0.597	0.523	1.000	0.556	0.485	0.603	0.534	0.524	0.611	0.547
0.622	0.594	0.556	1.000	0.564	0.555	0.496	0.500	0.502	0.513
0.530	0.465	0.485	0.564	1.000	0.608	0.513	0.559	0.647	0.505
0.517	0.501	0.603	0.555	0.608	1.000	0.452	0.486	0.644	0.518
0.518	0.558	0.534	0.496	0.513	0.452	1.000	0.577	0.635	0.466
0.490	0.560	0.524	0.500	0.559	0.486	0.577	1.000	0.585	0.524
0.455	0.460	0.611	0.502	0.647	0.644	0.635	0.585	1.000	0.474
0.578	0.513	0.547	0.513	0.505	0.518	0.466	0.524	0.474	1.000

The resulting values of `Xr` are:

```
set.seed(130575)
Xr <- mvrnorm(n, mu = rep(0, m), Sigma = x)
Xr <- data.frame(Xr)
kable(head(Xr), caption = "Firms' performance.", digits = 3)
```

Table 3.5.10: Firms' performance.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
-2.107	-1.206	-2.033	-1.730	-0.760	-2.207	-0.933	-1.141	-0.296	-1.308
-1.346	-2.023	-1.661	0.372	-0.326	0.874	-0.328	-0.315	-0.712	-0.266
0.178	-1.366	1.120	-1.732	-0.582	-0.820	-0.058	0.379	-0.151	0.294
0.641	1.168	1.146	1.028	-0.116	1.488	1.292	0.430	-0.186	0.500
1.263	-0.132	1.482	-0.572	0.485	0.649	-0.415	-1.470	0.669	0.797
0.538	0.810	0.435	0.215	0.797	1.257	1.007	1.166	1.767	0.181

Extract the cases in which all ten firms default at the same time in 5 years and the cases in which either firm default at the same time in 5 years.

```
Xr <- as_tibble(Xr)
# All firms.
Xr.y1.all <- filter_all(Xr, all_vars(. < x.y[1]))
Xr.y2.all <- filter_all(Xr, all_vars(. < x.y[2]))
Xr.y3.all <- filter_all(Xr, all_vars(. < x.y[3]))
Xr.y4.all <- filter_all(Xr, all_vars(. < x.y[4]))
```

```
Xr.y5.all <- filter_all(Xr, all_vars(. < x.y[5]))
# At least one firm.
Xr.y1.any <- filter_all(Xr, any_vars(. < x.y[1]))
Xr.y2.any <- filter_all(Xr, any_vars(. < x.y[2]))
Xr.y3.any <- filter_all(Xr, any_vars(. < x.y[3]))
Xr.y4.any <- filter_all(Xr, any_vars(. < x.y[4]))
Xr.y5.any <- filter_all(Xr, any_vars(. < x.y[5]))
```

All firms default in 5,935 cases out of 1,000,000.

```
nrow(Xr.y5.all)
```

```
## [1] 5935
```

Let's see the first 6 cases:

```
kable(head(Xr.y5.all),
      caption = "Cases in which all 10 firms default in five years.",
      digits = 3)
```

Table 3.5.11: Cases in which all 10 firms default in five years.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
-1.332	-2.060	-1.775	-2.084	-2.537	-1.948	-2.566	-2.234	-1.185	-1.897
-1.935	-1.393	-2.217	-1.527	-1.609	-1.794	-1.680	-1.235	-2.578	-1.188
-1.575	-1.215	-1.974	-1.116	-1.557	-2.516	-1.933	-2.999	-2.253	-2.294
-2.197	-2.460	-2.398	-2.829	-2.294	-1.923	-2.744	-2.217	-2.970	-1.460
-2.201	-1.222	-1.372	-2.246	-1.607	-1.540	-3.018	-1.733	-1.201	-1.597
-2.106	-2.355	-2.526	-2.044	-2.002	-2.416	-1.833	-2.936	-2.041	-2.080

Verify that those cases default.

```
kable((head(Xr.y5.all) < x.y[5]), caption = "Check if all of them default.")
```

Table 3.5.12: Check if all of them default.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE

At least one firm default in 497,987 out of 1,000,000.

```
nrow(Xr.y5.any)
```

```
## [1] 497987
```

```
kable(head(Xr.y5.any), caption = "At least one firm default in five years.",
      digits = 3)
```

Table 3.5.13: At least one firm default in five years.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
-2.107	-1.206	-2.033	-1.730	-0.760	-2.207	-0.933	-1.141	-0.296	-1.308
-1.346	-2.023	-1.661	0.372	-0.326	0.874	-0.328	-0.315	-0.712	-0.266
0.178	-1.366	1.120	-1.732	-0.582	-0.820	-0.058	0.379	-0.151	0.294
1.263	-0.132	1.482	-0.572	0.485	0.649	-0.415	-1.470	0.669	0.797
-0.001	-0.744	-1.625	0.673	-0.665	-0.442	-1.272	0.853	-0.647	-0.234
-0.180	-1.380	-0.383	-0.292	1.604	1.021	1.007	0.079	0.390	-0.426

```
kable((head(Xr.y5.any) < x.y[5]), caption = "Check which one(s) default.")
```

Table 3.5.14: Check which one(s) default.

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE
TRUE	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

And the corresponding probabilities:

```
allR <- t(data.frame(nrow(Xr.y1.all), nrow(Xr.y2.all),
        nrow(Xr.y3.all), nrow(Xr.y4.all), nrow(Xr.y5.all)))
atleastoneR <- t(data.frame(nrow(Xr.y1.any), nrow(Xr.y2.any),
        nrow(Xr.y3.any), nrow(Xr.y4.any), nrow(Xr.y5.any)))
resR <- data.frame(allR / n, atleastoneR / n)
ans <- data.frame(res02, resR)
rownames(ans) <- n.year
colnames(ans) <- c("All (corr=0.2)", "At least one (corr=0.2)",
                   "All (corr=rand)", "At least one (corr=rand)")
kable(ans, caption = "Probability of default.",
      row.names = TRUE)
```

Table 3.5.15: Probability of default.

	All (corr=0.2)	At least one (corr=0.2)	All (corr=rand)	At least one (corr=rand)
year 1	0.0e+00	0.087276	0.000018	0.062795
year 2	0.0e+00	0.226406	0.000163	0.156515
year 3	0.0e+00	0.386348	0.000825	0.267014
year 4	1.2e-05	0.543162	0.002421	0.382983
year 5	7.2e-05	0.682148	0.005935	0.497987

4 Credit VaR example 24.7.

This replicates Hull [2015] example 24.7.

This is the Vasicek model (equation 24.10) in a function form.

```
CVaR <- function(exp, pd, r, c, l) {
v <- pnorm((qnorm(pd) + (c^0.5) * qnorm(l)) / (1 - c)^0.5)
VaR <- exp * v * (1 - r)
}
```

See if it works. Let's evaluate equation 24.10 at 99% and 99.9%. The 1-year 99% and 99.9% credit VaR is:

```
CVaR.999 <- CVaR(100, 0.02, 0.6, 0.1, 0.999)
CVaR.99 <- CVaR(100, 0.02, 0.6, 0.1, 0.99)
CVaR.999
```

```
## [1] 5.129484
```

```
CVaR.99
```

```
## [1] 3.294271
```

Let's evaluate the model at all confidence levels (from 0 to 1) simulating 10,000 values.

```
set.seed(13)
l <- runif(10000, 0, 1)
Loss <- CVaR(100, 0.02, 0.6, 0.1, l)
sim.var999 <- sort(Loss)[10000 * 0.999]
sim.var99 <- sort(Loss)[10000 * 0.99]
```

Now, visually:

```
hist(Loss, 100, xlim = c(0, 7), xlab = "Losses in millions", main = NULL)
abline(v = CVaR.999, lty = 2, col = "red", lwd = 2)
abline(v = CVaR.99, lty = 2, col = "blue", lwd = 2)
legend("topright", legend = c("1-year 99% credit VaR is 3.294271",
         "1-year 99.9% credit VaR is 5.129484"),
col = c("blue", "red"), lwd = 2, lty = 2, bg = "white", cex = 0.8)
```

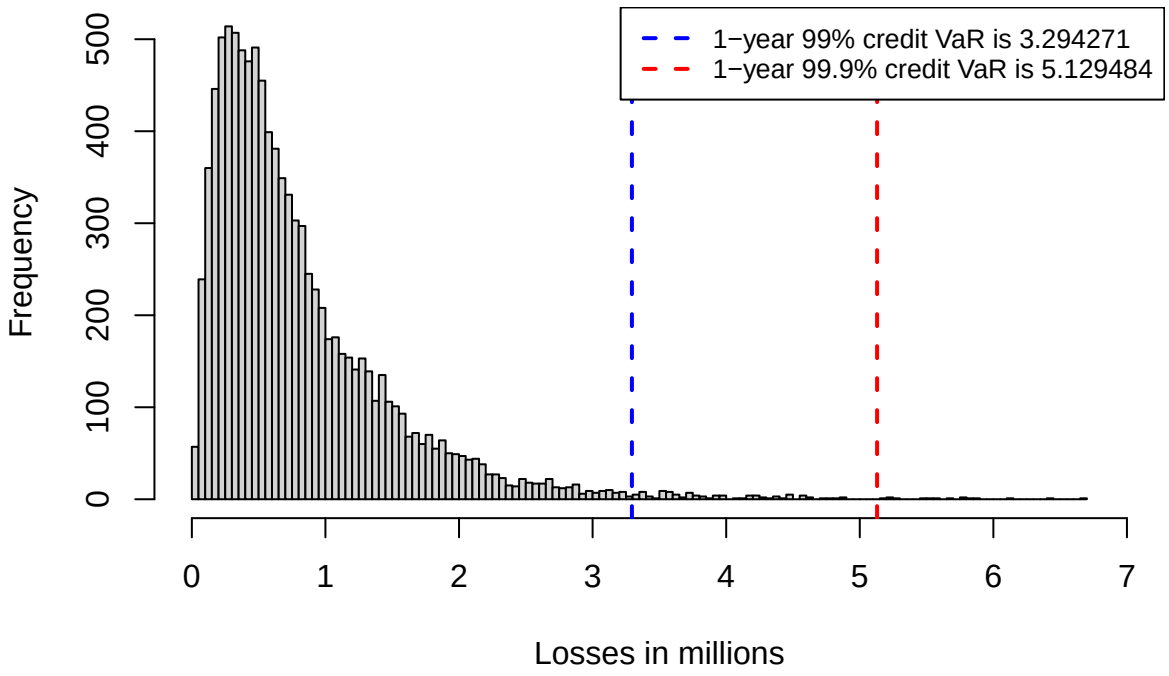


Figure 4.0.1: Distribution of losses.

```
dat <- data.frame(Loss, 1)
ggplot(dat, aes(x = Loss, fill = "Losses in millions")) +
  geom_density(color = "darkblue", fill = "lightblue") +
geom_vline(aes(xintercept = CVaR.999),
               color = "red", linetype = "dashed", size = 1) +
  geom_vline(aes(xintercept = CVaR.99 ),
               color = "blue", linetype = "dashed", size = 1)
```

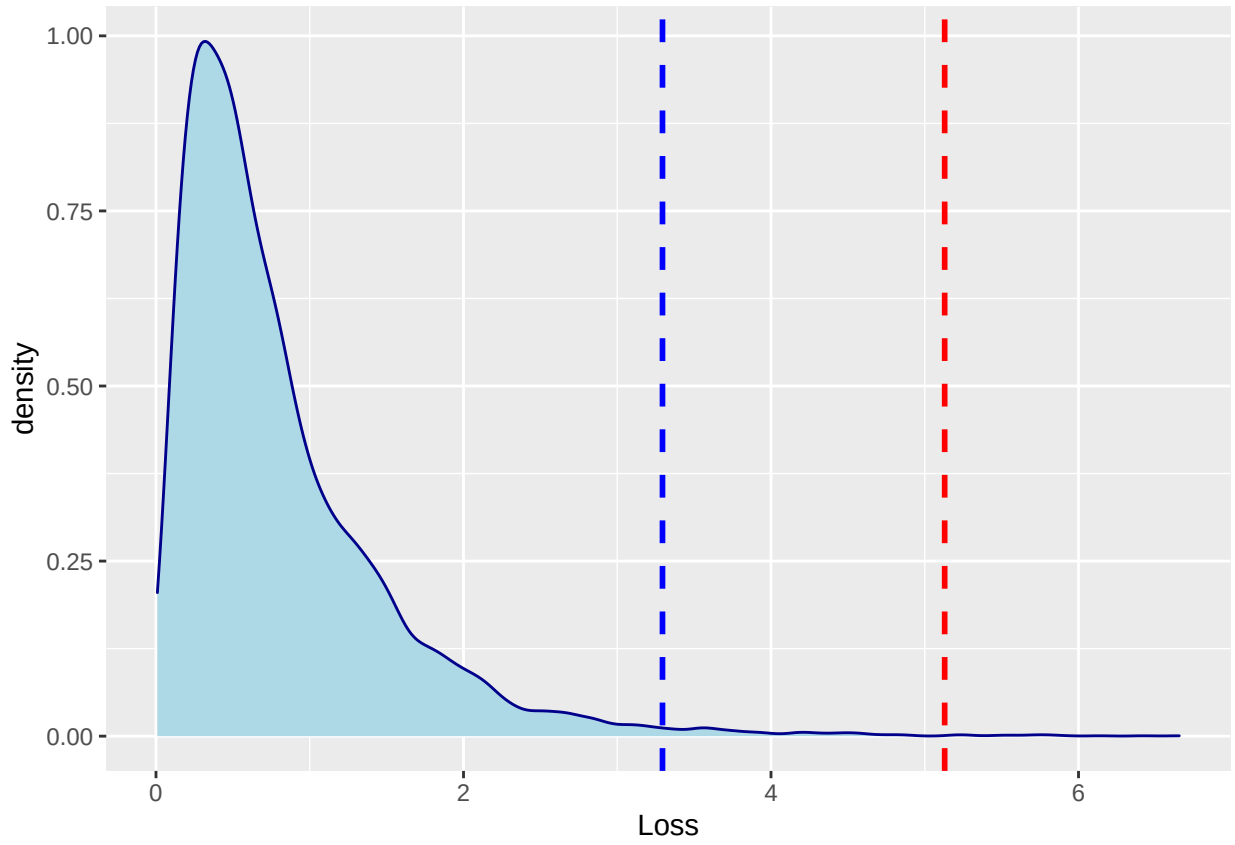


Figure 4.0.2: Distribution of losses.

Nice.

References.

John C. Hull. *Options, futures, and other derivatives*. Prentice Hall, 9th ed. edition, 2015.

John C. Hull. *Machine Learning in Business: An Introduction to the World of Data Science*. Amazon Distribution, 2020.

Donald Ervin Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.