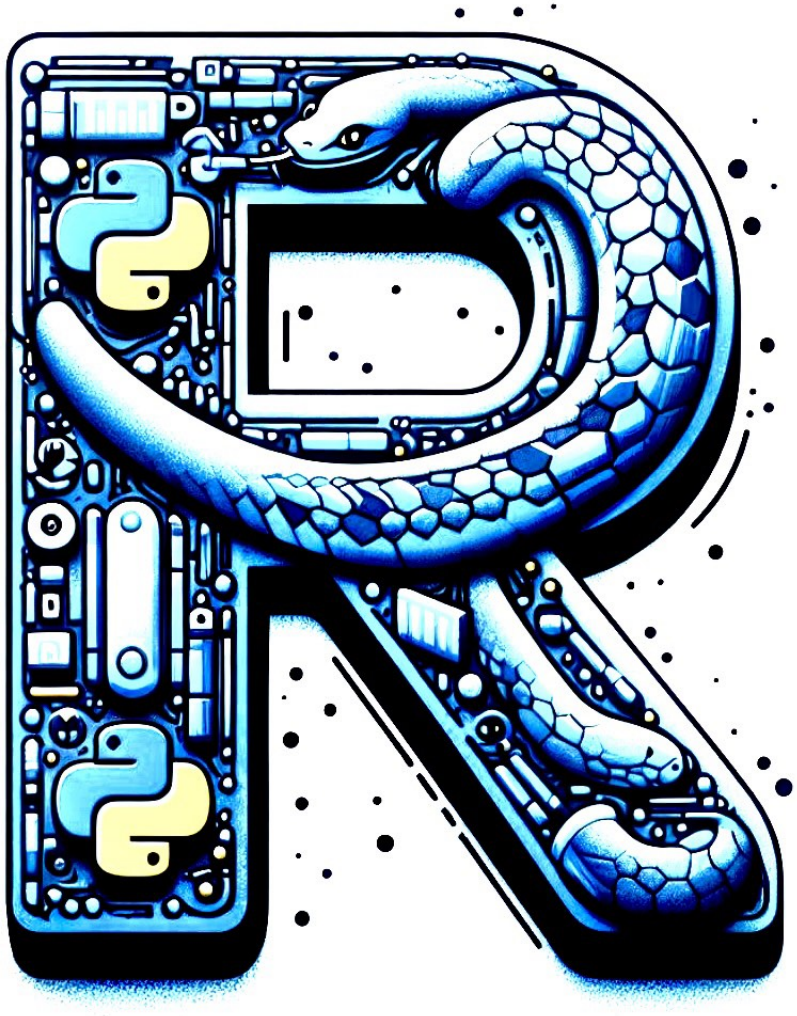
Consequences of Multicollinearity for OLS regression



Andrija Djurovic*

*www.linkedin.com/in/andrija-djurovic

The absence of multicollinearity is another assumption of OLS regression. Simply put, multicollinearity occurs when two or more independent variables in a regression model are (highly) correlated with each other. The consequences of violating this assumption can be severe including imprecise estimates, a large variance in estimates, and instability in the signs of the estimates. Practitioners typically view multicollinearity as a problem only when observing instability in the estimates' signs. The reduction of precision is often overlooked, although it warrants further investigation and potential model adjustments based on expert inputs.

The following example illustrates some consequences of multicollinearity at different levels of intercorrelation between two independent variables.

Let's begin in R by defining simulation parameters and a helper function for the data-generating process.

```
library(MASS)
library(dplyr)

#simulation parameters
n <- 45                                 #number of observations
B <- 1000                               #simulation replicates
beta1 <- 0.50                           #beta1 estimate
beta2 <- 0.30                           #beta2 estimate
rho <- c(0, 0.25, 0.50, 0.75, 0.95)     #correlation between x1 and x2

#helper function for simulation of the betas
sim.betas <- function(n, beta1, beta2, rho) {

      #covariance matrix
      cm <- matrix(data = c(1, beta1, beta2,
                            beta1, 1, rho,
                            beta2, rho, 1),
                   nrow = 3,
                   ncol = 3)

      #simulate data
      db <- mvrnorm(n = n,
                    mu = c(0, 0, 0),
                    Sigma = cm)
      db <- data.frame(db)
      names(db) <- c("y", "x1", "x2")
```

```
      #run the regression
      ols.reg <- lm(formula = y ~ x1 + x2,
                    data = db)

      #extract the estimates
      betas <- summary(ols.reg)$coefficients[-1, 1]

return(betas)
}
```

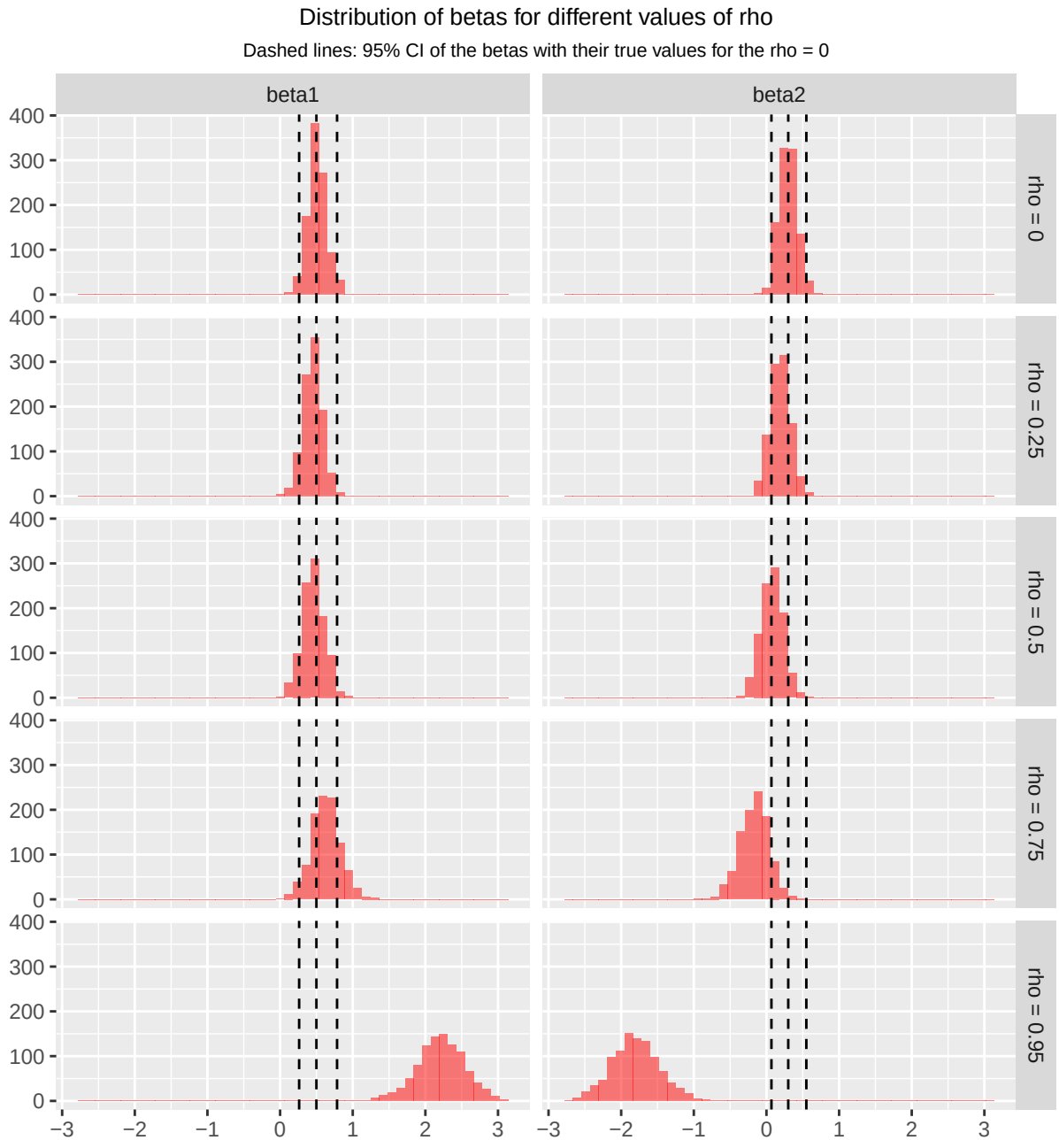
With the inputs prepared, we can now proceed to conduct the simulation.

```
#-------------------------run the simulations-------------------------#
#empty object to store the results
res <- vector("list", B)

for   (i in 1:B) {

      #random seed
      set.seed(i)

      #simulation
      sim.i <- sapply(X = rho,
                      FUN = function(x) {
                            sim.betas(n = n,
                                      beta1 = beta1,
                                      beta2 = beta2,
                                      rho = x)
                            }
                      )

      #store the results
      res.i <- cbind.data.frame(simulation = i,
                                rho = rho,
                                data.frame(t(sim.i)))
      res[[i]] <- res.i
      }

#concatenate the results
res <- do.call("rbind", res)
```

```r
#summarize the results
res %>%
group_by(rho) %>%
summarise(x1.low = quantile(x = x1, probs = 0.025),
          x1.cnt = quantile(x = x1, probs = 0.50),
          x1.upp = quantile(x = x1, probs = 0.975),
          x2.low = quantile(x = x2, probs = 0.025),
          x2.cnt = quantile(x = x2, probs = 0.50),
          x2.upp = quantile(x = x2, probs = 0.975),
          x1.std = sd(x1),
          x2.std = sd(x2)) %>%
as.data.frame()
```

```
##    rho    x1.low    x1.cnt    x1.upp      x2.low      x2.cnt     x2.upp
## 1 0.00 0.2622169 0.5024366 0.7842092  0.06852155  0.29413426  0.5491430
## 2 0.25 0.1874227 0.4511160 0.7123027 -0.07764253  0.18626673  0.4555672
## 3 0.50 0.1462334 0.4541766 0.7486144 -0.21831320  0.08039602  0.3835392
## 4 0.75 0.2452639 0.6292947 1.0465997 -0.57526176 -0.15771493  0.2144264
## 5 0.95 1.5086766 2.2185101 2.8365354 -2.44411043 -1.81307143 -1.1467552
##      x1.std    x2.std
## 1 0.1271422 0.1260787
## 2 0.1318679 0.1366010
## 3 0.1513431 0.1552601
## 4 0.2007366 0.1998318
## 5 0.3234497 0.3269169
```

To better understand the findings above, let's represent the results visually.



Concluding the exercise, we'll replicate the same simulation using `Python`.

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

#simulation parameters
```

```python
n = 45                                          #number of observations
B = 1000                                        #simulation replicates
beta1 = 0.50                                    #beta1 estimate
beta2 = 0.30                                    #beta2 estimate
rho = [0, 0.25, 0.50, 0.75, 0.95]               #correlation between x1 and x2

#helper function for simulation of the betas
def sim_betas(n, beta1, beta2, rho):

    #covariance matrix
    cm = np.array([[1, beta1, beta2],
                   [beta1, 1, rho],
                   [beta2, rho, 1]])

    #simulate data
    db = np.random.multivariate_normal(mean = [0, 0, 0],
                                       cov = cm,
                                       size = n)
    db = pd.DataFrame(data = db,
                      columns = ["y", "x1", "x2"])

    #run the regression
    ols_reg = smf.ols(formula = "y ~ x1 + x2",
                      data = db).fit()

    #extract the estimates
    betas = ols_reg.params[1:]

    return betas

#-------------------------run the simulations-------------------------#
#empty object to store the results
res = [None]*B

#run the simulations
for i in range(1, B + 1):

    #set random seed
    np.random.seed(i)
```

```
    #simulate data for different rho values
    sim_results = [sim_betas(n, beta1, beta2, rho_i) for rho_i in rho]

    #store the results
    res_i = pd.DataFrame(data = sim_results)
    res_i["simulation"] = i
    res_i["rho"] = rho
    res[i - 1] = res_i

#concatenate the results
res = pd.concat(objs = res,
                ignore_index = True)

#summarize the results
res.groupby("rho").agg(
    x1_low = ("x1", lambda x: np.percentile(a = x, q = 2.5)),
    x1_cnt = ("x1", lambda x: np.percentile(a = x, q = 50)),
    x1_upp = ("x1", lambda x: np.percentile(a = x, q = 97.5)),
    x2_low = ("x2", lambda x: np.percentile(a = x, q = 2.5)),
    x2_cnt = ("x2", lambda x: np.percentile(a = x, q = 50)),
    x2_upp = ("x2", lambda x: np.percentile(a = x, q = 97.5)),
    x1_std = ("x1", lambda x: np.std(a = x, ddof = 1)),
    x2_std = ("x2", lambda x: np.std(a = x, ddof = 1))
    ).reset_index()
```

```
##     rho    x1_low    x1_cnt    x1_upp    x2_low    x2_cnt    x2_upp    x1_std  \
## 0  0.00  0.247497  0.503320  0.742984  0.049488  0.306543  0.550458  0.126296
## 1  0.25  0.189725  0.460893  0.720391 -0.077847  0.185654  0.462899  0.136916
## 2  0.50  0.163336  0.468381  0.783256 -0.227433  0.054696  0.372123  0.155673
## 3  0.75  0.238313  0.629706  1.050971 -0.581171 -0.174227  0.203122  0.200822
## 4  0.95  1.492846  2.201046  2.857388 -2.489935 -1.794232 -1.131327  0.340420
##
##      x2_std
## 0  0.127056
## 1  0.139141
## 2  0.155170
## 3  0.196688
## 4  0.334148
```