Day1 exercise solution

Ali Movasati

Sept. 16th, 2024

```
# Set global code chunk options
knitr::opts_chunk$set(warning = FALSE)
```

```
# load required libraries
library(ggplot2)
library(magrittr)
library(dplyr)



# define functions
`%notin%` <- Negate(`%in%`)
```

Problem 1 (Resampling)

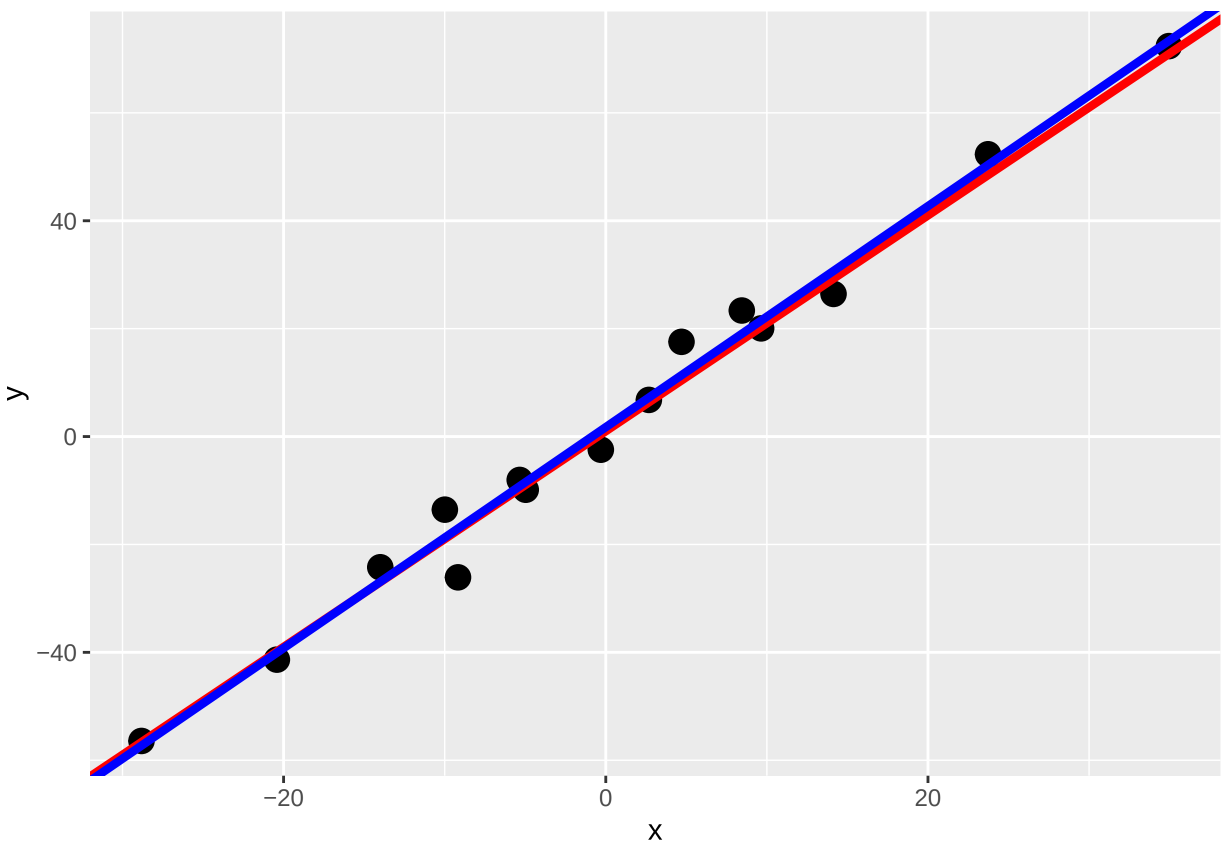
A)

```
# set parameters

n <- 15
mu <- 4
std_sig <- 4
std_err <- 2
beta0 <- 1
beta1 <- 2

# set computer seed for reproducibility
set.seed(333)

# generate independent and dependent variables
x <- rnorm(n , mu, std_sig^2)
y <- beta0 + beta1*x + rnorm(n, 0, std_err^2)

# generate linear model
model <- lm(y~x)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max
## -9.0483 -1.4329 -0.4266  2.3430  6.1880
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.73981    1.05416    1.65    0.123
## x            2.04829    0.06572   31.17 1.33e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.082 on 13 degrees of freedom
## Multiple R-squared:  0.9868, Adjusted R-squared:  0.9858
## F-statistic: 971.3 on 1 and 13 DF,  p-value: 1.332e-13
```

```r
# extract estimated parameters
est_intercept <- model$coefficients[1]
est_slope <- model$coefficients[2]

# generate plot

df <- data.frame(x = x, y = y)

df %>% ggplot(aes(x = x, y = y)) +
        geom_point(size = 4) +
        geom_abline(intercept = beta0, slope = beta1, color = "red", size = 1.5) +
        geom_abline(intercept = est_intercept, slope = est_slope, color = "blue", size = 1.5)
```

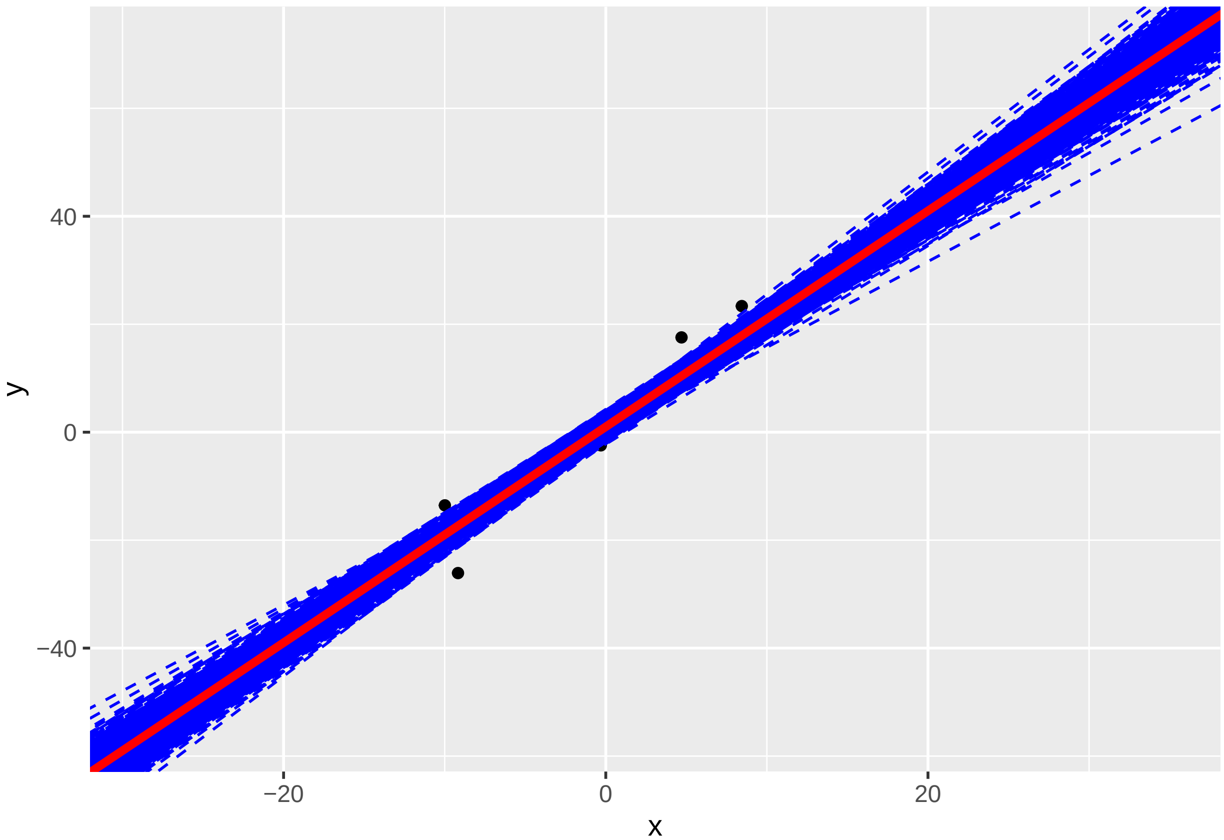


The estimated intercept and slope of regression line are 1.7398071 and 2.0482891; respectively!

B)

In this part we will use **bootstrapping**. This method is useful when we do not know the parameters of the distribution from which the sample set is drawn from.

```
# set number of iterations
repeats <- 1000

# generate and store estimated bootstrapping values for intercept and slope of regression lines
model_coef_df <- data.frame()

for (i in 1:repeats){
    x_bts <- sample(x, n, replace = TRUE)
    y_bts <- beta0 + beta1*x_bts + rnorm(n, 0, std_err^2)
    model <- lm(y_bts~x_bts)
    est_intercept <- model$coefficients[1]
    est_slope <- model$coefficients[2]
    model_coef_df <- rbind(model_coef_df, c(est_intercept,est_slope))
}

colnames(model_coef_df) <- c("est_intercept", "est_slope")


# generate plot
p <- df %>% ggplot(aes(x = x, y = y)) +
        geom_point()

for (i in 1:nrow(model_coef_df)) {
  p <- p + geom_abline(intercept = model_coef_df$est_intercept[i], slope = model_coef_df$est_slope[i], 
}

p + geom_abline(intercept = beta0, slope = beta1, color = "red", size = 1.5)
```



It seems that our estimates of intercept and slope of regression line are accurate. Below is the numerical summary of each estimate based on 1000 bootstrapping:

Intercept:

```
summary(model_coef_df$est_intercept)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
## -2.2270  0.1955  0.9772  0.9424  1.6636  4.2363
```

Slope:

```
summary(model_coef_df$est_slope)
```

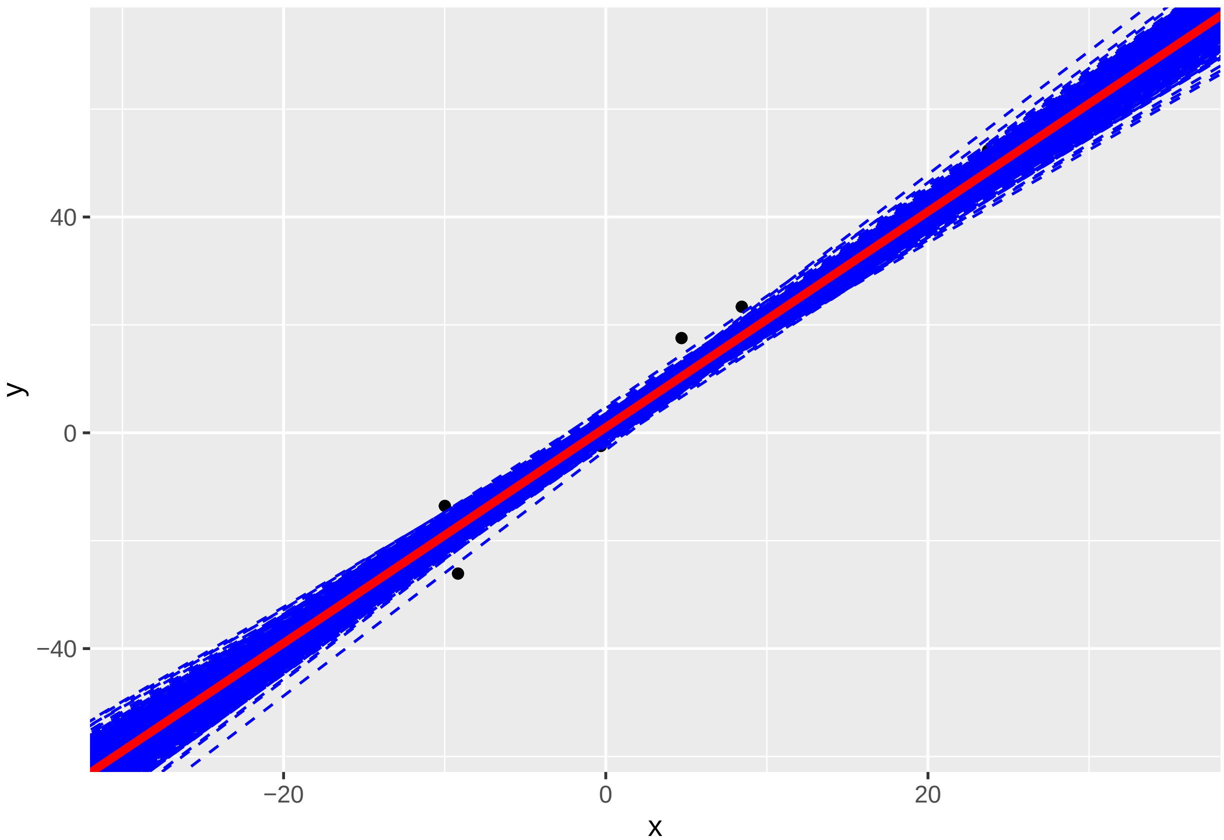
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   1.591   1.954   2.004   2.002   2.049   2.267
```

C)

In this part since we know the distribution parameters of the population, we can perform **resampling** by generating new numbers from the known distribution.

```
# set number of iterations
repeats <- 1000

# generate and store estimated bootstrapping values for intercept and slope of regression lines
model_coef_df <- data.frame()

for (i in 1:repeats){
    x_resampling <- rnorm(n , mu, std_sig^2)
```

```
    y_resampling <- beta0 + beta1*x_resampling + rnorm(n, 0, std_err^2)
    model <- lm(y_resampling~x_resampling)
    est_intercept <- model$coefficients[1]
    est_slope <- model$coefficients[2]
    model_coef_df <- rbind(model_coef_df, c(est_intercept,est_slope))
}

colnames(model_coef_df) <- c("est_intercept", "est_slope")


# generate plot
p <- df %>% ggplot(aes(x = x, y = y)) +
        geom_point()

for (i in 1:nrow(model_coef_df)) {
  p <- p + geom_abline(intercept = model_coef_df$est_intercept[i], slope = model_coef_df$est_slope[i], c
}

p + geom_abline(intercept = beta0, slope = beta1, color = "red", size = 1.5)
```



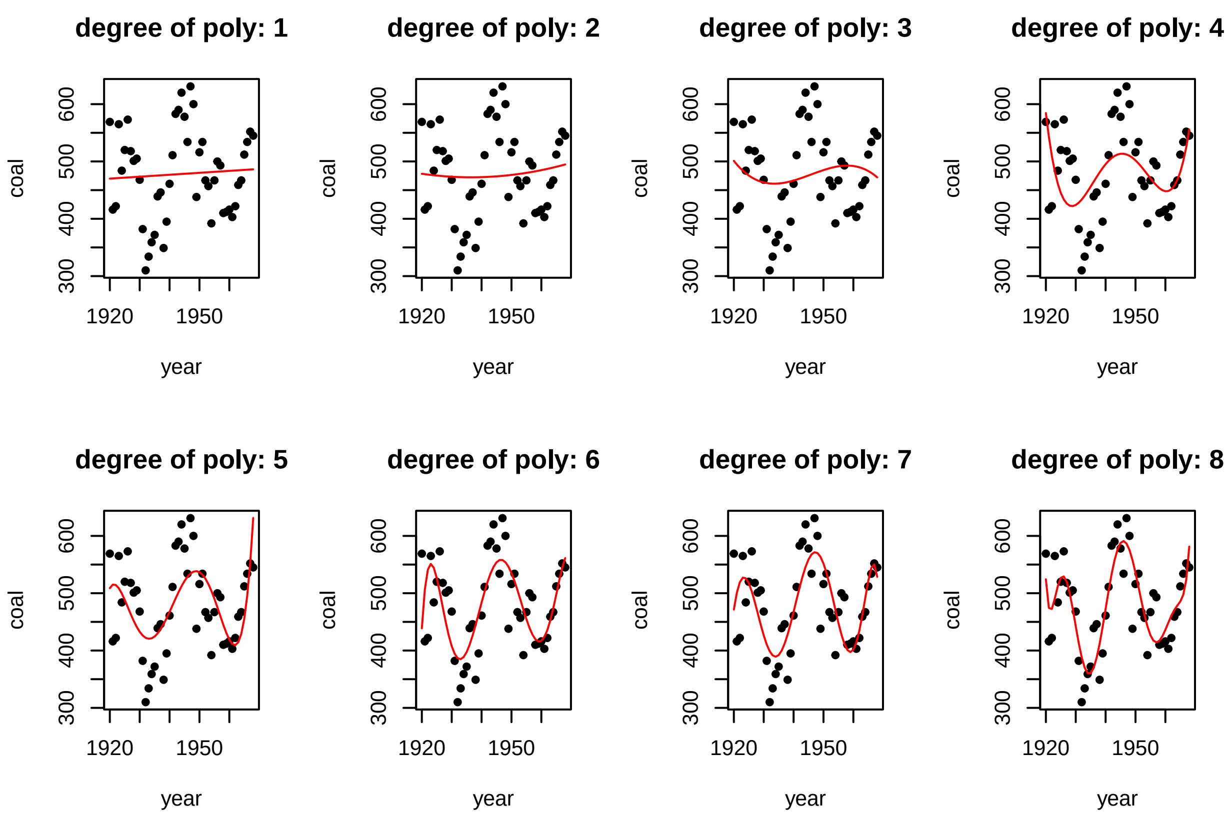
Both methods of bootstrapping and resampling in this case works because we have enough data points for bootstrapping and we know the parameters of the population for resampling method.

Problem 2

```r
# load and prepare data
library(fma)


coal <- as.numeric(bicoal)
year <- c(time(bicoal))
coal_df <- data.frame(coal = coal, year = year)
```

A)

```r
# generate plot of fitted polynomial line with degrees between 1-8
#pdf("/Users/alimos313/Documents/studies/phd/university/courses/stat-modelling/day1/figs/polynomial.pdf
par(mfrow=c(2,4))
for (i in seq(1, 8, by=1)) {
    model <- lm(coal ~ poly(year, i), data = coal_df)
    plot(year, coal, pch=20, main = paste0("degree of poly: ", i))
    lines(year, model$fitted.values, col ="red")
}
```




```r
dev.off()
```

```
## null device
##           1
```

```r
# set paramters
degrees <- 1:8
k_folds <- 5
```

```
# calculate and store rss values

rss_df <- data.frame()

for (i in degrees){
    for (j in 1:100){
        fold_indices <- sample(rep(1:k_folds, length.out = nrow(coal_df)))
        for (k in 1:k_folds){
            training_indices <- which(fold_indices != k)
            model <- lm(coal ~ poly(year, degrees[i]), data = coal_df, subset = training_indices)
            res_valid <- predict(model, coal_df[-training_indices, ])
            rss <- sum((res_valid - coal_df$coal[-training_indices])^2)
            rss_df <- rbind(rss_df, c(i, j, k, rss))
        }
    }
}

colnames(rss_df) <- c("degree", "rep", "fold_nr", "rss")


# print average rss per polynomial degree
rss_summary <- rss_df %>% group_by(degree, fold_nr) %>% summarize(mean_rss = mean(rss), .groups = "drop

paste0("The polynomial model with degree ", rss_summary$degree[which.min(rss_summary$mean_rss)], " has
```

```
## [1] "The polynomial model with degree 6 has the lowest mean of RSS of 56828.55"
```

B)

Here we are asked to split the dataset into a test set before generating a model. The splitting method however is not random, and we are reserving the first and last 5 years present in the data for the test set.

```
# reserve test set
test_set_indices <- c(1:5, (nrow(coal_df)-4):nrow(coal_df))

# use the rest of data points for model creation and evaluation
model_set_indices <- which(1:49 %notin% test_set_indices)

# degrees to be checked to find the optimal model
degrees <- 1:10

# iterate the 5-fold cross validation step and store the rss
rss_df2 <- data.frame()

set.seed(333)
for (i in degrees){
    for (j in 1:100){
            training_set_indices <- sample(model_set_indices,26)
            model <- lm(coal ~ poly(year, degrees[i]), data = coal_df, subset = training_set_indices)
            res_valid <- predict(model, coal_df[-c(test_set_indices,training_set_indices), ])
            rss <- sum((res_valid - coal_df$coal[-c(test_set_indices,training_set_indices)])^2)
            rss_df2 <- rbind(rss_df2, c(i, j, rss))
    }
```

```
}

colnames(rss_df2) <- c("degree", "rep", "rss")

# find the polynomial degree with the lowest rss average
rss_summary2 <- rss_df2 %>% group_by(degree) %>% summarize(mean_rss = mean(rss), .groups = "drop") %>% 

paste0("The polynomial model with degree ", rss_summary2$degree[which.min(rss_summary2$mean_rss)], " has
```

```
## [1] "The polynomial model with degree 6 has the lowest mean of RSS of 50374.97"
```

```
opt_degree <- rss_summary2$degree[which.min(rss_summary2$mean_rss)]

# recreate the model with the optimal polynomial degree and predict function for the test set!
model <- lm(coal ~ poly(year, opt_degree), data = coal_df, subset = model_set_indices)
res_valid <- predict(model, coal_df[test_set_indices,])
rss <- sum((res_valid - coal_df$coal[test_set_indices])^2)
```

The RSS of the "optimal model" which is a polynomial model with degree of 6 is 3.472958×10^6

Non-random splitting of the data is not a good practice and can introduced biases in evaluating the performance of the model.