Day1 exercise solution

Ali Movasati

Sept. 16th, 2024

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

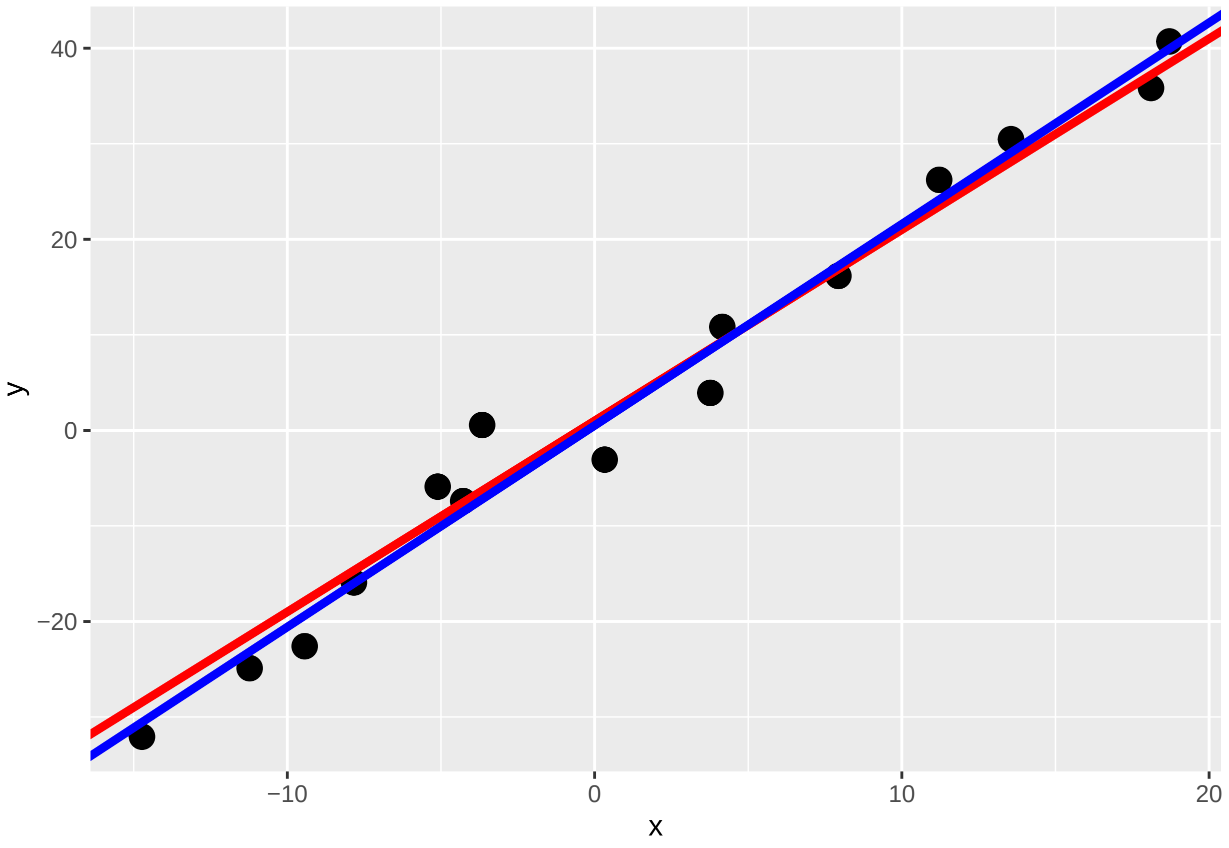
```r
# load required libraries
library(ggplot2)
library(magrittr)
library(dplyr)



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

Problem 1 (Resampling)

A)

```r
# set parameters

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




# 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)


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

# generate plot
```

```r
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)
```



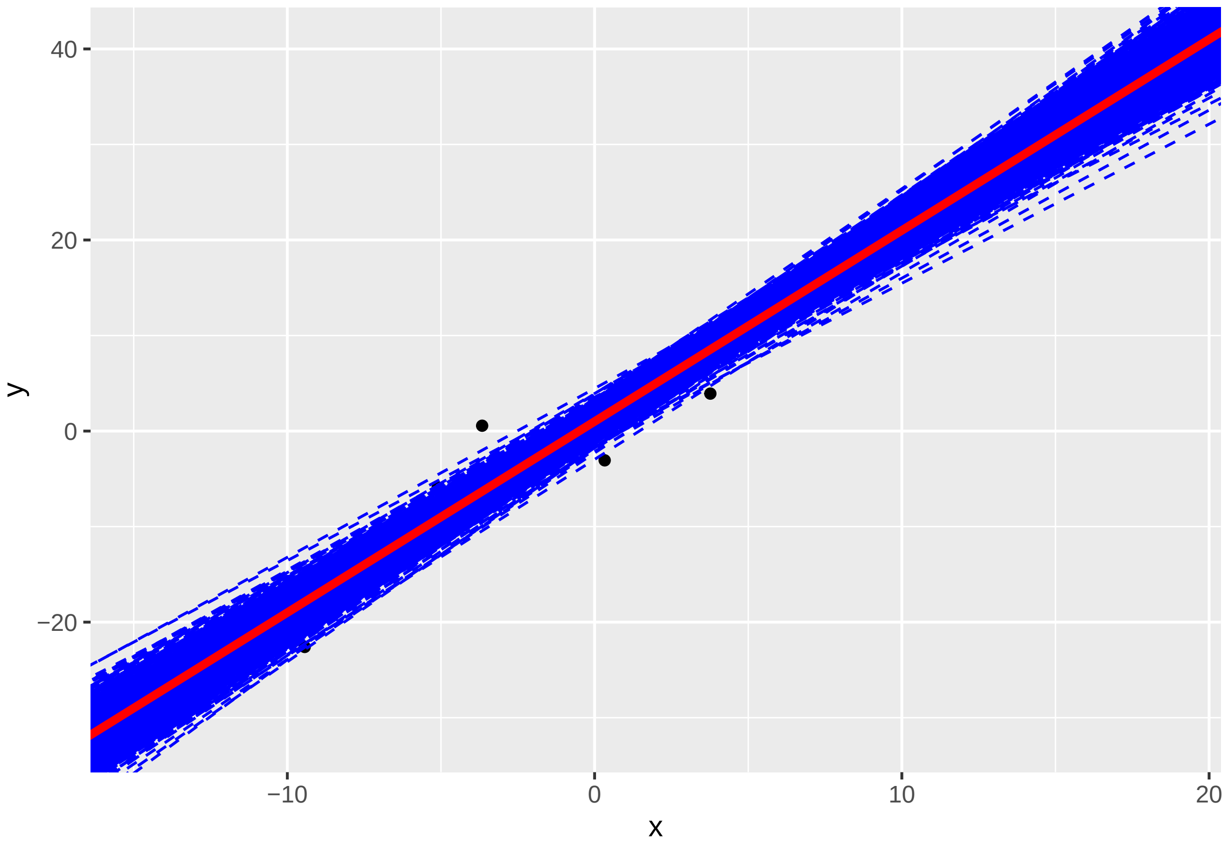
The estimated intercept and slope of regression line are 0.5012466 and 2.1085026; respectively!

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.

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

# generate and store estimated bootstrapping values for intercept and slope of regression lines
model_coef_df1 <- 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_df1 <- rbind(model_coef_df1, c(est_intercept,est_slope))
}
```

```r
colnames(model_coef_df1) <- c("est_intercept", "est_slope")


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

for (i in 1:nrow(model_coef_df1)) {
  p <- p + geom_abline(intercept = model_coef_df1$est_intercept[i], slope = model_coef_df1$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:

```r
summary(model_coef_df1$est_intercept)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.9704  0.4054  1.0581  1.0666  1.7710  4.4577
```

Slope:

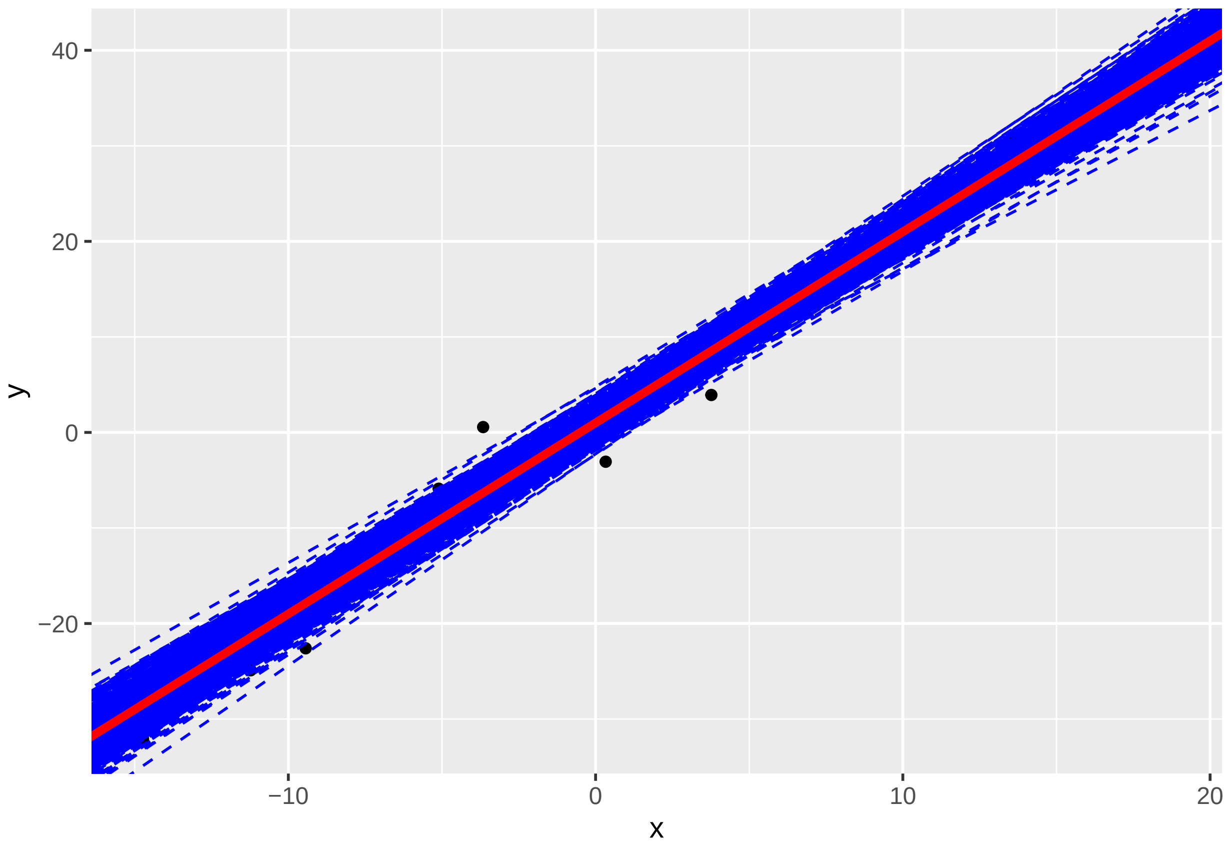
```r
summary(model_coef_df1$est_slope)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.648   1.931   2.000   2.001   2.073   2.377
```

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_df2 <- 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_df2 <- rbind(model_coef_df2, c(est_intercept,est_slope))
}

colnames(model_coef_df2) <- c("est_intercept", "est_slope")


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

for (i in 1:nrow(model_coef_df2)) {
  p <- p + geom_abline(intercept = model_coef_df2$est_intercept[i], slope = model_coef_df2$est_slope[i]
}

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.

« Comments »

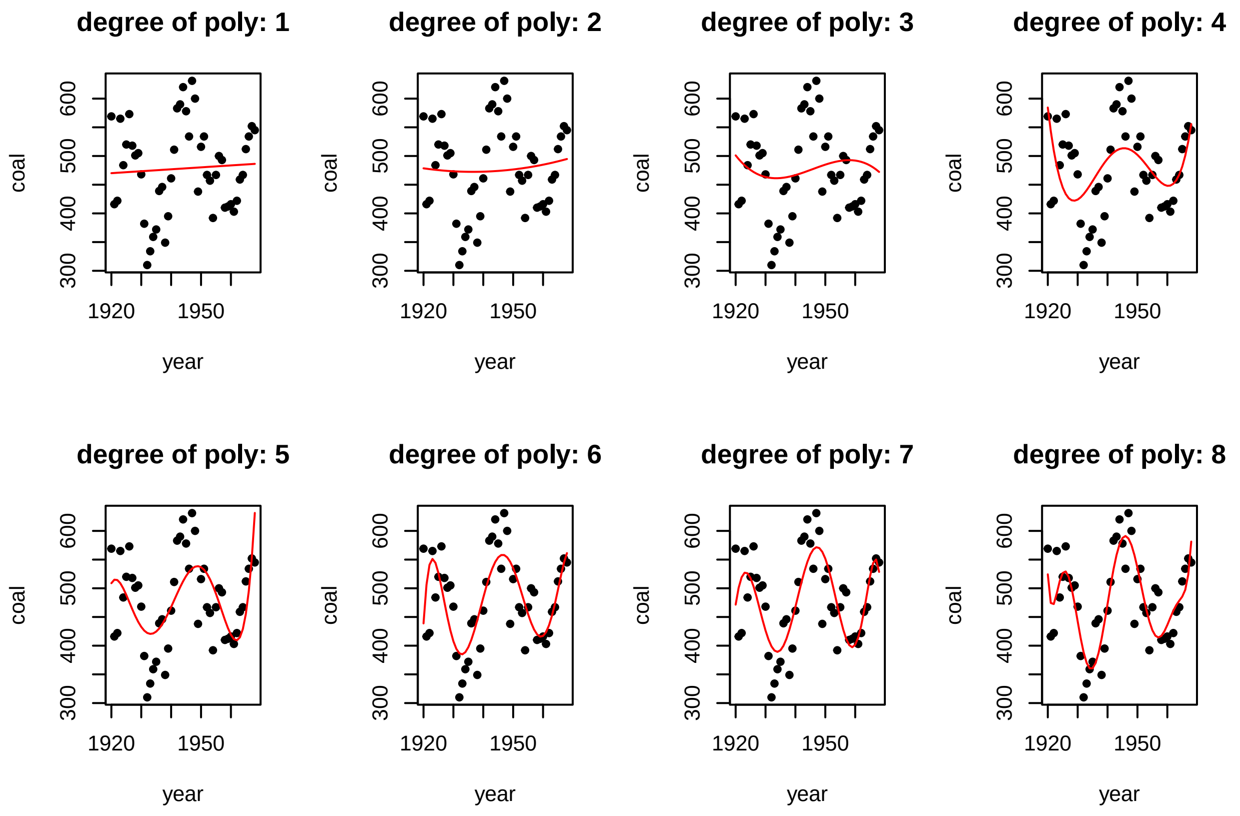
- In A section we estimate the intercept and slope of the linear regression. The estimates are slightly different from true parameters of the model due to the variation that is caused by the error term of the model.
- In section B, we used **bootstrapping** to obtain distribution estimates for the model parameters. The estimate for the intercept is 1.0665787 with 95% CI [-0.6622554,2.8725903] and for the slope 2.0011305 with 95% CI [1.8227062,2.1759897]
- In section C, we used **resampling** to obtain distribution estimates for the model parameters. The estimate for the intercept is 1.0081708 with 95% CI [-0.701548,2.7384577] and for the slope 1.9968857 with 95% CI [1.8813402,2.1131599]

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)
```

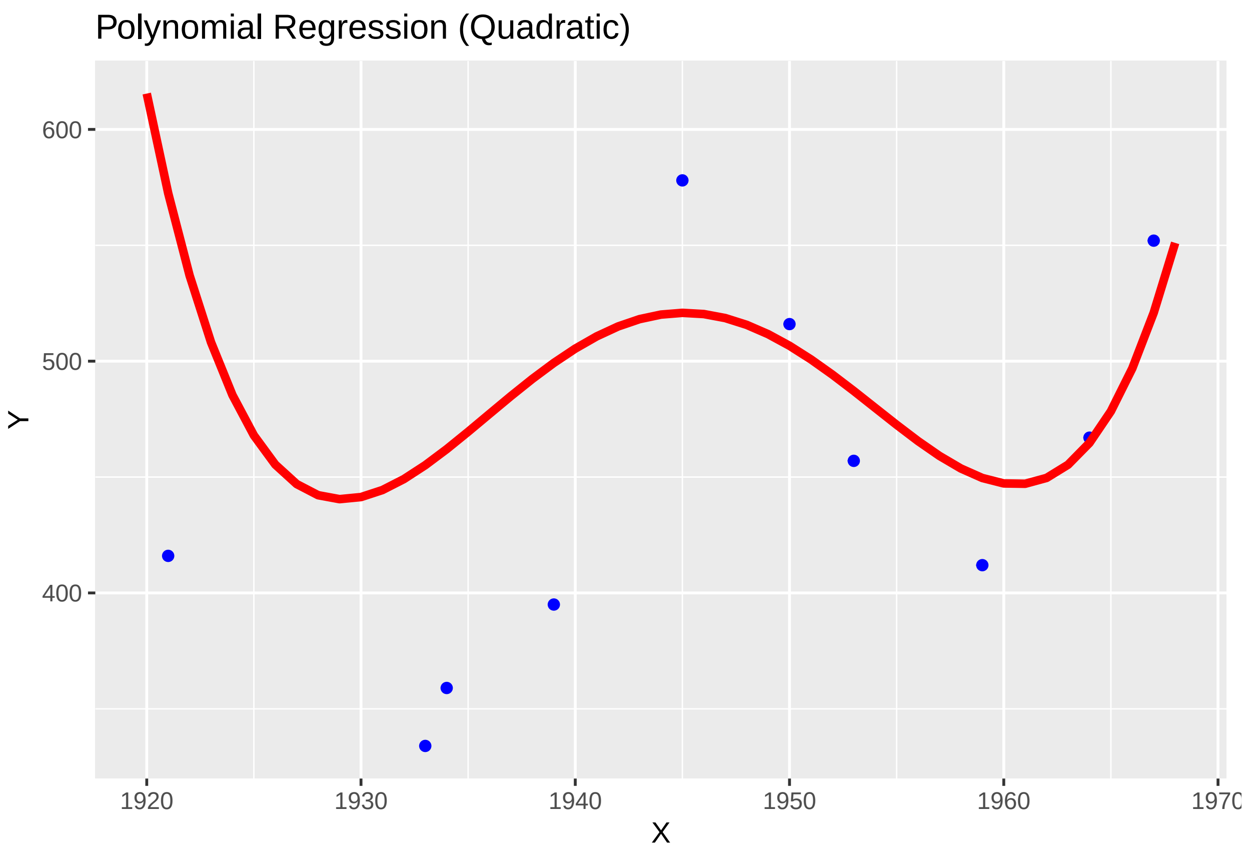
A)

```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()
```

```
## pdf
##   2
```



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

# set computer seed for reproducibility
set.seed(4)


# calculate and store rss values

rss_df <- data.frame()

for (i in 1:100){
    fold_indices <- sample(rep(1:k_folds, length.out = nrow(coal_df)))
    for (j in degrees){
        training_indices <- which(fold_indices != 1)
        model <- lm(coal ~ poly(year, j), 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, rss))
    }
}


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

```r
# print average rss per polynomial degree
rss_summary <- rss_df %>% group_by(degree) %>% summarize(median_rss = median(rss))

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

```
## [1] "The polynomial model with degree 7 has the lowest mean of RSS of 40504.64"
```

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.

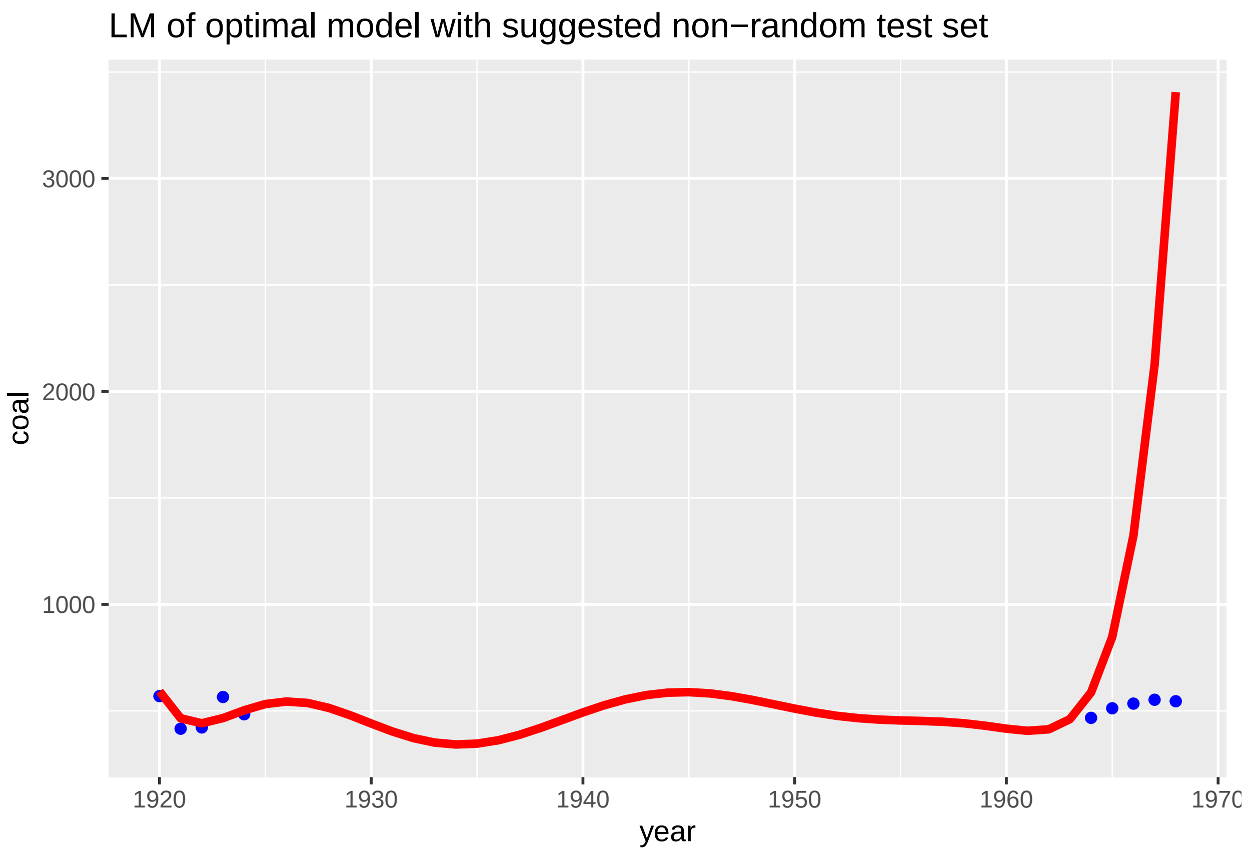
```r
# reserve test set
test_set_indices <- c(1:5, (nrow(coal_df)-4):nrow(coal_df))


# 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 <- 1:8

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


for (i in 1:100){
    for (j in degrees){
            training_set_indices <- sample(model_set_indices,26)
            model <- lm(coal ~ poly(year, j), 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("rep", "degree", "rss")

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

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

```
## [1] "The polynomial model with degree 8 has the lowest mean of RSS of 36671.22"
```

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

```r
# 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)



# visualization
```

```r
y_predicted <- predict(model, coal_df)
predicted_data <- data.frame(year = year, predicted = y_predicted)


ggplot(coal_df[test_set_indices,], aes(x = year, y = coal)) +
    geom_point(color = "blue") +   # Scatter plot
    geom_line(data = predicted_data, aes(x = year, y = y_predicted), color = "red", size = 1.5) +      #
    labs(title = "LM of optimal model with suggested non-random test set",
        x = "year", y = "coal")
```

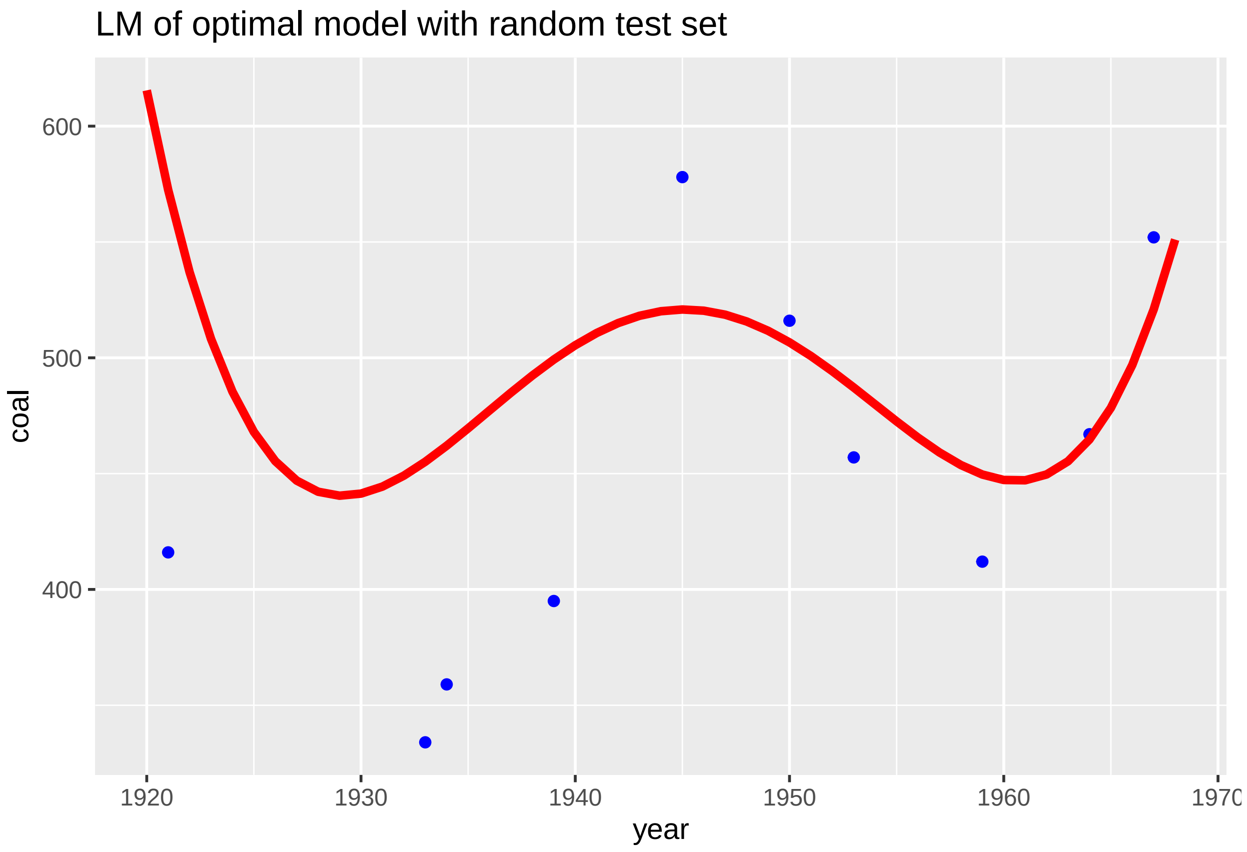


The RSS of the "optimal model" which is a polynomial model with degree of 8 is 1.1420707×10^7 when tested on the test data set.

« Comments »

- Cross validation is a method to measure model performance. In this problem we fitted polynomial models with different degrees to our data and measured RSS by cross validation. The lower the RSS, the "better" the model performs. Based on that criteria, the polynomial model with degree of 8 was found to perform the best.
- In the second part of the problem, we reserved a test set before doing the same procedure and at the end measured the performance of the model on the test Non-random splitting of the test set is not a good practice and can introduced biases in evaluating the performance of the model. As in this case, the model performance on the test set based on the RSS value is a lot poorer than what we would have accepted from cross validation. Probably the non-random selection of the test set also impacted our decision on the optimal model. Therefore, it would be a better approach to randomly split the data, ensuring that both the training and test sets are representative of the underlying distribution.

If we do the same procedure but with random selection of the test set:



The RSS of the “optimal model” when model set and test set are selected randomly is 6.7298838×10^4 when tested on the test data set which is considerably lower than non-random selection method.