Weeks 7+8 - Simulation Project

STAT 420, Fall 2024, Banghao Chi, banghao2

10/20/2024

Contents

Directions

This is an **individual** project similar to homework assignments. However, because of the project structure of your submission, collaboration should be more limited than an homework assignment to protect yourself from duplication. Discussion of question intent, coding problems/issues, and project administration may be discussed on the message board on a limited basis. However, sharing, copying, or providing any part of this project to another student is an infraction of the University's rules on academic integrity. Any violation will be punished as severely as possible.

- Your project must be submitted through Coursera. You are required to upload one `.zip` file, named `yourNetID-sim-proj.zip`, which contains:
  - Your RMarkdown file which should be saved as `yourNetID-sim-proj.Rmd`.
  - The result of knitting your RMarkdown file as `yourNetID-sim-proj.html`.
  - Any outside data provided as a `.csv` file. (In this case, `study_1.csv` and `study_2.csv`.)
- Your `.Rmd` file should be written such that, when stored in a folder with any data you are asked to import, it will knit properly without modification. If your `.zip` file is organized properly, this should not be an issue.
- Include your name and NetID in the final document, not only in your filenames.

This project consists of **three** simulation studies. MCS-DS and other campus graduate students must complete all three studies. Campus undergraduate students need to complete on the first two studies. There is no extra credit for undergraduates who complete the third study.

Unlike a homework assignment, these "exercises" are not broken down into parts (e.g., a, b, c), and so your analysis and report submission will not be partitioned into parts. Instead, your document should be organized more like a true project report, and it should use the overall format:

- Simulation Study 1
- Simulation Study 2
- Simulation Study 3 (*MCS and graduates only*)

Within each of the simulation studies, you should use the format:

- Introduction
- Methods

- Results
- Discussion

The **introduction** section should relay what you are attempting to accomplish. It should provide enough background to your work such that a reader would not need this directions document to understand what you are doing. Basically, assume the reader is mostly familiar with the concepts from the course, but not this project. [For the Midterm Assignment, the Introduction section is allowed to simply be the exercise statements as I have typed them later in this file. Yes, a direct copy.]

The **methods** section should contain the majority of your "work." This section will contain the bulk of the `R` code that is used to generate the results. Your `R` code is not expected to be perfect idiomatic `R`, but it is expected to be understood by a reader without too much effort. Use RMarkdown and code comments to your advantage to explain your code if needed. [For the Midterm Assignment, you may type a sentence or two explaining what you are attempting to do and then the code chunk that does it. Repeat as necessary to tell a coherent story about your method. That is, it would be clear to display your code in a few smaller chunks explaining along the way, as opposed to one giant chunk with a large paragraph trying to describe the whole thing at once.]

The **results** section should contain numerical or graphical summaries of your results as they pertain to the goal of each study. [For the Midterm Assignment, while the code chunks appear in Methods, the actual plots, numeric tables, etc. would appear in Results.]

The **discussion** section should contain discussion of your results. The discussion section should contain discussion of your results. Potential topics for discussion are suggested at the end of each simulation study section, but they are not meant to be an exhaustive list. These simulation studies are meant to be explorations into the principles of statistical modeling, so do not limit your responses to short, closed form answers as you do in homework assignments. Use the potential discussion questions as a starting point for your response. [For the Midterm Assignment, This is where you may give your summary of what you observe in the Results. The "Potential Topics" are meant to get your thoughts flowing and give you ideas about what to discuss. This section will be just your typed responses with no new code or results.]

- Your resulting `.html` file will be considered a self-contained "report," which is the material that will determine the majority of your grade. Be sure to visibly include all `R` code and output that is *relevant*. (You should not include irrelevant code you tried that resulted in error or did not answer the question correctly.)
- Grading will be based on a combination of completing the required tasks, discussion of results, `R` usage, RMarkdown usage, and neatness and organization. For full details see the provided rubric.
- At the beginning of *each* of the three simulation studies, set a seed equal to your birthday, as is done on homework. (It should be the first code run for each study.) These should be the only three times you set a seed.

```
birthday = 18760613
set.seed(birthday)
```

One Final Note: The simulations in this Midterm Assignment require combinations of several factors that result in a lot of computation. For example, in Simulation Study 1, the response vector that you generate will have $2(models)Ö3(sigmas)Ö2000(sims) = 12000$ simulated values. Expect that `R` may take longer to compile than your typical weekly Homework Assignment, especially your final report. I'll suggest two tips.

- Tip 1: Make a separate `R` script, notebook, or Rmd for each Simulation Study. Thus, while you are working through one study, you are not making `R` try to compile the code for the other studies that you are not actively working on.
- Tip 2: Start with a smaller number of simulations until you work out the bugs. For example, 200 simulations in Study 1 instead of 2000. The end Results will not be correct initially, but `R` will compile faster while you are still figuring out the code in the Methods. Once you've nailed that, update your code with the correct number of simulations.

Good luck!

Simulation Study 1: Significance of Regression

Introduction

In this simulation study we will investigate the significance of regression test. We will simulate from two different models:

1. The **"significant"** model

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$ and

- $\beta_0 = 3$,
- $\beta_1 = 1$,
- $\beta_2 = 1$,
- $\beta_3 = 1$.

2. The **"non-significant"** model

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$ and

- $\beta_0 = 3$,
- $\beta_1 = 0$,
- $\beta_2 = 0$,
- $\beta_3 = 0$.

For both, we will consider a sample size of 25 and three possible levels of noise. That is, three values of σ.

- $n = 25$
- $\sigma \in (1, 5, 10)$

Use simulation to obtain an empirical distribution for each of the following values, for each of the three values of σ, for both models.

- The F **statistic** for the significance of regression test.
- The **p-value** for the significance of regression test
- R^2

For each model and σ combination, use 2000 simulations. For each simulation, fit a regression model of the same form used to perform the simulation.

Use the data found in `study_1.csv` for the values of the predictors. These should be kept constant for the entirety of this study. The `y` values in this data are a blank placeholder.

Done correctly, you will have simulated the `y` vector $2(models)Ö3(sigmas)Ö2000(sims) = 12000$ times.

Methods

Before we begin, we need to first do the following two things:

- Set the seed with our birthday date.

```
birthday = 20021023
set.seed(birthday)
```

- Import our data.

```r
ss1_data = read.csv("study_1.csv")
```

After that, we can start to setup the overall environment with the following config according to our introduction section:

```r
n = 25
sigmas = c(1, 5, 10)
num_sims = 2000
```

We also need to prepare a predefined data structure to store all our results for the two different models in all simulations and sigmas:

```r
results_sig = array(0, dim = c(num_sims, length(sigmas), 3))
results_nonsig = array(0, dim = c(num_sims, length(sigmas), 3))
```

We will use array to store the value, as this kind of data structure allows us to store multiple values for a specific position within the data structure.

We'll also need to extract the predictor variable from the original data to use them later in our simulation process:

```r
x1 = ss1_data$x1
x2 = ss1_data$x2
x3 = ss1_data$x3
```

Here, we are going to define some functions that we can re-use multiple times to make the overall simulations process neat. Since we will definitely have simulated response to do the empirical experiments, we first define the function that return simulated response:

```r
sim_data = function(n, beta0, beta1, beta2, beta3, sigma, x1, x2, x3) {
  epsilon = rnorm(n, mean = 0, sd = sigma)
  beta0 + beta1*x1 + beta2*x2 + beta3*x3 + epsilon
}
```

This function uses `rnorm` to generate normal distributed noise and add it to the mean response formula to form the simulated response.

We'll also need another function to calculate the statistics value such as F-statistics, p-value and r squared of different models under different simulation turns and sigmas:

```r
cal_stats = function(y, x1, x2, x3) {
  model = lm(y ~ x1 + x2 + x3)
  f_stat = summary(model)$fstatistic[1]
  p_value = pf(f_stat, 3, 21, lower.tail = FALSE)
  r_squared = summary(model)$r.squared
  c(f_stat, p_value, r_squared)
}
```

In this function, we use `lm()` to apply MLR model to the data. We also get the r squared and f statistics from the model summary, calculate p-value according to the f statistics, and finally return all three statistics.

Also, when we calculate the p-value, the two degrees of freedom that we need are 3 and 21 respectively, since for the the formula for these two are $p-1$ and $n-p$, and p here represents the number of β, which is 4, and n represents the number of samples, which is 25. Therefore, the respective two degrees of freedom are 3 and 21.

After setting up two functions, we can actually get started with the for loop to process different simulations with different models and sigmas:

```r
for (sigma_index in 1:length(sigmas)) {
  for (sim in 1:num_sims) {
```

```
    y_sig = sim_data(n, 3, 1, 1, 1, sigmas[sigma_index], x1, x2, x3)
    results_sig[sim, sigma_index, ] = cal_stats(y_sig, x1, x2, x3)

    y_nonsig = sim_data(n, 3, 0, 0, 0, sigmas[sigma_index], x1, x2, x3)
    results_nonsig[sim, sigma_index, ] = cal_stats(y_nonsig, x1, x2, x3)
  }
}
```

In this code chunk, we use two for loops to loop through all the situations: the outer loop is responsible for simulating with different sigmas, while the inner loop is there to simulate 2000 times for the same model. Within the loop, we first use n, real model parameters values, sigma and x to get the simulated response from true model using `sim_data()` function. Then, the simulated response will be input into `cal_stats` function along with our x columns to get the three statistics that we are interested in, and we store them in the pre-defined data structure.

Results

To make a grid as suggested in the additional tip, we will utilize both `ggplot2` and `gridExtra` packages, where `ggplot2` is mainly used for plotting while `gridExtra` is used for grid layout of the plots:

```
library(ggplot2)
library(gridExtra)
```

Then, we can define a function dedicated for plotting for

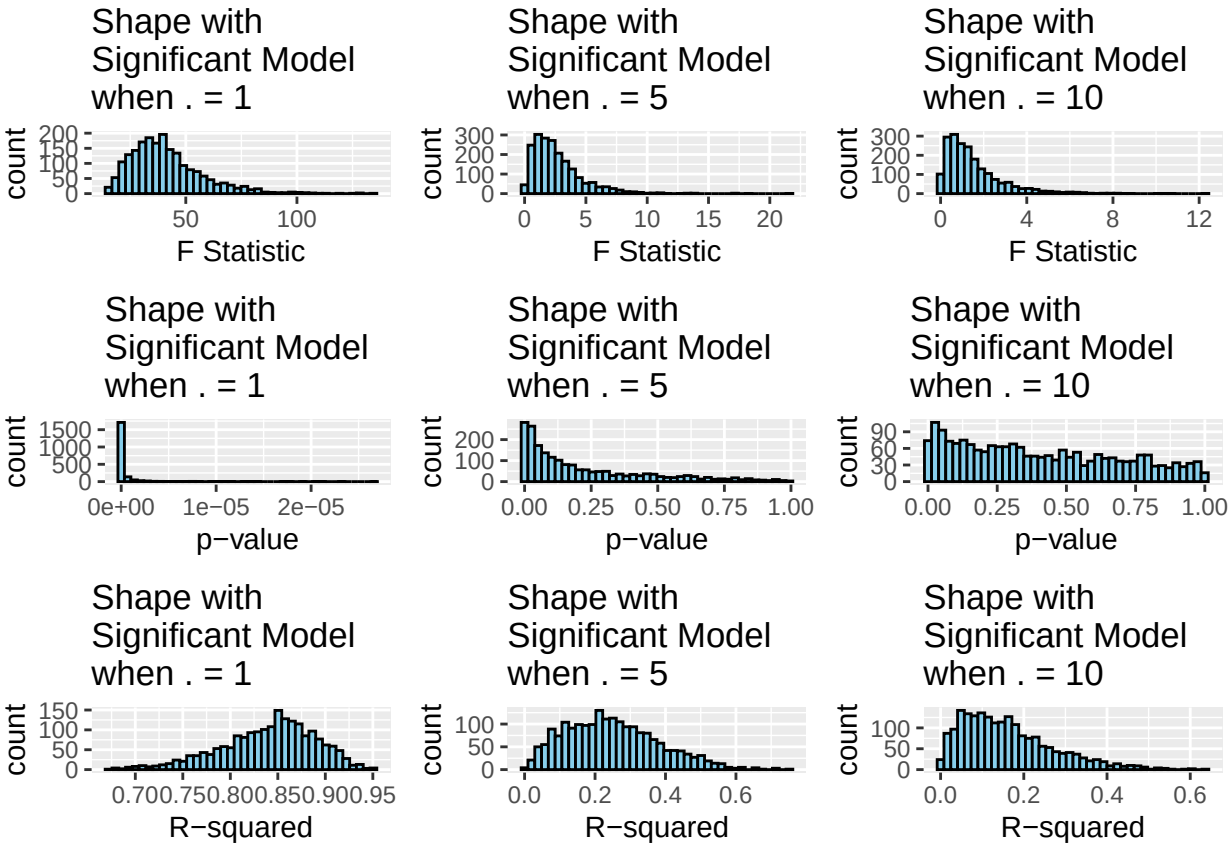
```
plot_histograms = function(data, title, x_label) {
  plots = list()
  for (i in 1:3) {
    for (j in 1:length(sigmas)) {
      df = data.frame(value = data[, j, i])
      p = ggplot(df, aes(x = value)) +
        geom_histogram(bins = 40, fill = "skyblue", color = "black") +
        ggtitle(paste(title, "\nwhen sigma =", sigmas[j])) +
        xlab(x_label[i])
      plots[[length(plots) + 1]] = p
    }
  }
  grid.arrange(grobs = plots, ncol = 3)
}
```

Here, we first use a list to store complex objects which later will be known as ggplot objects. Since `c()` is only suitable for simple type, we choose to use `list()` here.

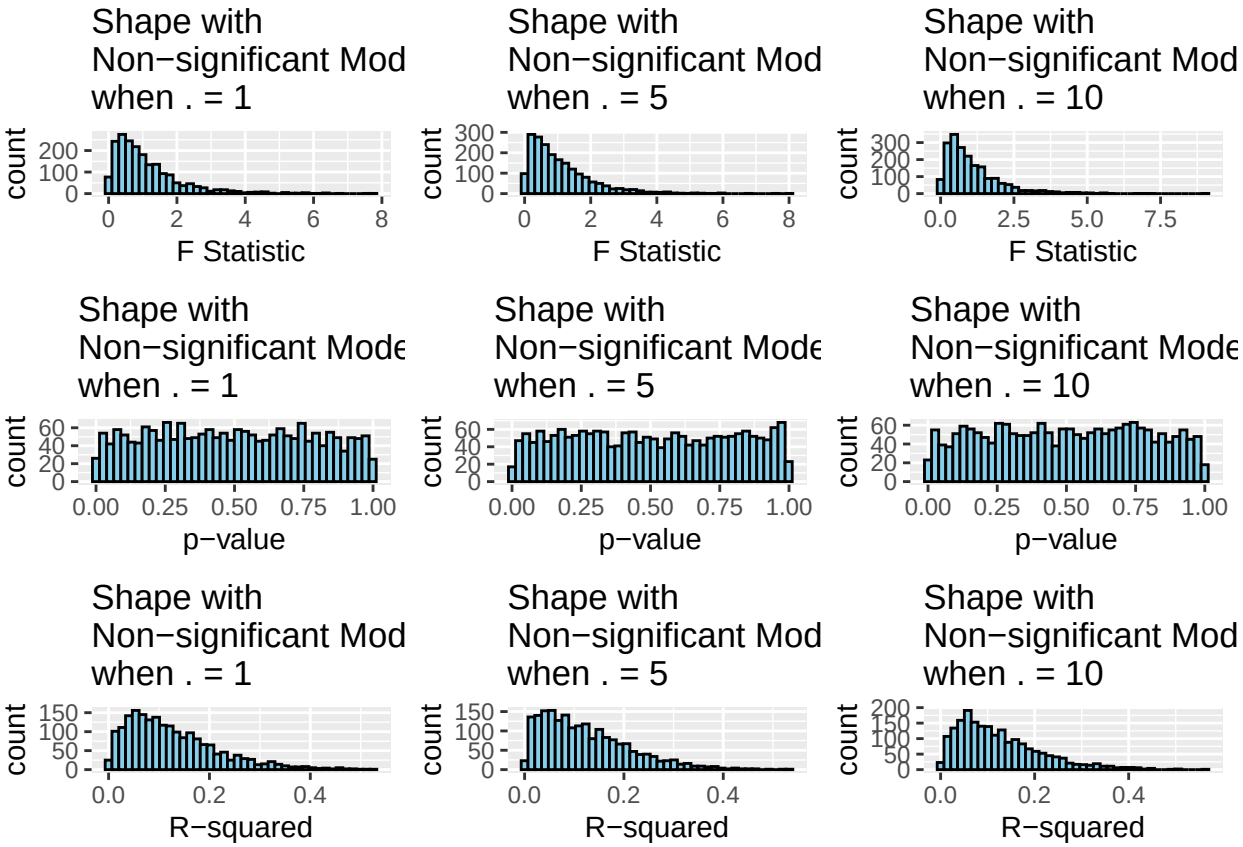
In the above code chunk, we also have two for loops, the outer one is responsible for each statistics(e.g., f-statistics, p-value, r squared), the inner one is for each sigma. Inside the for loop, the simulated results with a specific sigma and statistic value are extracted, which will be used later in plotting. In the contruction of ggplot objects, we specify the data, histogram config, title and x label, and finally we append it to the `plots` list for grid layout arrangement.

After setting up the function for plotting, we can call the function with the respective data, title and x labels:

```
x_labels = c("F Statistic", "p-value", "R-squared")
plot_histograms(results_sig, "Shape with \nSignificant Model", x_labels)
```




```
plot_histograms(results_nonsig, "Shape with \nNon-significant Model", x_labels)
```



Discussion

The formula for F-statistic is:

$$F = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2/(p-1)}{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2/(n-p)}$$

The formula for R^2 is:

$$R^2 = \frac{\sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

- True distribution:
  - For MLR model, F-statistics follows the F-distribution no matter what model it is with degrees of freedom $p-1$ and $n-p$.
  - For MLR model, p-value is uniformly distributed with non-significant model, since the whole point of using the correct distribution (normal, t, f, chisq, etc.) is to transform from the test statistic to a uniform p-value. (referenced from Stack Overflow).
  - For MLR model, the distribution of R^2, as we can tell from the formulas above, follows the scaled F-distribution.
- Empirical distribution:
  - From the plot, F-statistics follows the F-distribution no matter what model it is.
  - From the plot, p-value:
    - Clusters near zero for the significant model, indicating strong evidence against the null hypothesis.
    - Is uniformly distributed with non-significant model, matching theoretical expectations.
  - From the plot, it is likely that R^2 follows a F-distribution when it's non-significant model.
- Changing with sigma:
  - For F-statistics:
    - When it's significant model, the whole distribution shifts left as sigma gets bigger.

    - When it's non-significant model, the distribution barely changes.
  - For p-value:
    - When it's significant model, the whole distribution shifts right as sigma gets bigger.
    - When it's non-significant model, the distribution remains uniform regardless of sigma.
  - For R^2:
    - When it's significant model, the whole distribution shifts left as sigma gets bigger.
    - When it's non-significant model, the distribution barely changes.

Simulation Study 2: Using RMSE for Selection?

Introduction

In homework we saw how Test RMSE can be used to select the "best" model. In this simulation study we will investigate how well this procedure works. Since splitting the data is random, we don't expect it to work correctly each time. We could get unlucky. But averaged over many attempts, we should expect it to select the appropriate model.

We will simulate from the model

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4} + \beta_5 x_{i5} + \beta_6 x_{i6} + \epsilon_i$$

where $\epsilon_i \sim N(0, \sigma^2)$ and

- $\beta_0 = 0$,
- $\beta_1 = 3$,
- $\beta_2 = -4$,
- $\beta_3 = 1.6$,
- $\beta_4 = -1.1$,
- $\beta_5 = 0.7$,
- $\beta_6 = 0.5$.

We will consider a sample size of 500 and three possible levels of noise. That is, three values of σ.

- $n = 500$
- $\sigma \in (1, 2, 4)$

Use the data found in `study_2.csv` for the values of the predictors. These should be kept constant for the entirety of this study. The `y` values in this data are a blank placeholder.

Each time you simulate the data, randomly split the data into train and test sets of equal sizes (250 observations for training, 250 observations for testing).

For each, fit **nine** models, with forms:

- `y ~ x1`
- `y ~ x1 + x2`
- `y ~ x1 + x2 + x3`
- `y ~ x1 + x2 + x3 + x4`
- `y ~ x1 + x2 + x3 + x4 + x5`
- `y ~ x1 + x2 + x3 + x4 + x5 + x6`, the correct form of the model as noted above
- `y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7`
- `y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8`
- `y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9`

For each model, calculate Train and Test RMSE.

$$\text{RMSE}(\text{model, data}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Repeat this process with 1000 simulations for each of the 3 values of σ. For each value of σ, create a plot that shows how average Train RMSE and average Test RMSE changes as a function of model size. Also show the number of times the model of each size was chosen for each value of σ.

Done correctly, you will have simulated the y vector $3Ö1000 = 3000$ times. You will have fit $9Ö3Ö1000 = 27000$ models. A minimal result would use 3 plots. Additional plots may also be useful.

Methods

Before we begin, we need to first do the following two things:

- Set the seed with our birthday date.

```
birthday = 20021023
set.seed(birthday)
```

- Import our data.

```
ss2_data = read.csv("study_2.csv")
```

After that, we can start to setup the overall environment with the following config according to our introduction section:

```
n = 500
sigmas = c(1, 2, 4)
num_sims = 1000
betas = c(0, 3, -4, 1.6, -1.1, 0.7, 0.5, 0, 0, 0)
x_data = ss2_data[, 2:ncol(ss2_data)]
```

We also need to prepare a predefined data structure to store all our results for all 9 different models in all simulations, sigmas and datasets(train and test, which are number of 2) for visualizaton purpose:

```
results = array(0, dim = c(num_sims, 2, 9, length(sigmas)))
```

During each simulation under a specific sigma, we'll also be counting the number of times that which model is the best according to the RMSE on the test dataset and store it in another data structure to decide the best model in the future step:

```
model_selections = matrix(0, nrow = length(sigmas), ncol = 9)
```

After that, we are going to define some functions that we can re-use multiple times to make the overall simulations process neat. Since we will definitely have simulated response to do the empirical experiments, we first define the function that return simulated response:

```
sim_data = function(n, betas, sigma, x_data) {
  epsilon = rnorm(n, mean = 0, sd = sigma)
  betas[1] + betas[2] * x_data$x1 + betas[3] * x_data$x2 + betas[4] * x_data$x3 + betas[5] * x_data$x4 +
}
```

This function uses `rnorm` to generate normal distributed noise and add it to the mean response formula to form the simulated response.

We'll also need another function to calculate the statistics value, in this case RMSE, under different simulation turns and sigmas given `y` and `y_hat`:

```
rmse = function(y, y_hat) {
  sqrt(mean((y - y_hat)^2))
}
```

In this function, we calculate the RMSE given the actual `y` and the simulated `y_hat` according the the formula in the introduction section.

Finally, we'll create a function that fit different models with the inputed training dataset, test dataset, simulated y to get the results and count the times of best model:

```
fit_models = function(train_data, test_data, y_train, y_test) {
  formulas = list(
    y ~ x1,
    y ~ x1 + x2,
    y ~ x1 + x2 + x3,
    y ~ x1 + x2 + x3 + x4,
    y ~ x1 + x2 + x3 + x4 + x5,
    y ~ x1 + x2 + x3 + x4 + x5 + x6,
    y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7,
    y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8,
    y ~ x1 + x2 + x3 + x4 + x5 + x6 + x7 + x8 + x9
  )

  results = data.frame(
    train_rmse = rep(0, 9),
    test_rmse = rep(0, 9)
  )

  for (i in 1:length(formulas)) {
    model = lm(formulas[[i]], data = data.frame(y = y_train, train_data))

    y_hat_train = predict(model, newdata = train_data)
    results$train_rmse[i] = rmse(y_train, y_hat_train)

    y_hat_test = predict(model, newdata = test_data)
    results$test_rmse[i] = rmse(y_test, y_hat_test)
  }

  return(results)
}
```

Within the function, we first use a list for storing diffferent formulas for the true models(same reason as the utilization of `list()` in simulation study 1). After that, we create a pre-defined dataframe to store the RMSE results for both train and test dataset.

Then we comes to the for loop, which is designed to fit every model, calculate the RMSE, store it in the data structure and finally it return it. Within the for loop, we first use `lm()` to apply MLR model given the response and the predictor variable. After that, we use the applied model to predict the value given the train or test data to get the predicted `y_hat`, and finally calculate, store, and return the RMSE data.

After setting up the functions, can actually get started with the main for loop to process different simulations with different models and sigmas:

```
for (sigma_index in 1:length(sigmas)) {
  sigma = sigmas[sigma_index]
```

```
  for (sim in 1:num_sims) {
    y = sim_data(n, betas, sigma, x_data)

    train_idx = sample(1:n, n/2)
    train_data = x_data[train_idx, ]
    test_data = x_data[-train_idx, ]
    y_train = y[train_idx]
    y_test = y[-train_idx]

    sim_results = fit_models(train_data, test_data, y_train, y_test)

    results[sim, 1, , sigma_index] = sim_results$train_rmse
    results[sim, 2, , sigma_index] = sim_results$test_rmse

    best_model = which.min(sim_results$test_rmse)
    model_selections[sigma_index, best_model] = model_selections[sigma_index, best_model] + 1
  }
}
```

Here we also have two for loops, outer for different sigma, inner for different simulations. Inside two for loops, we simulate the response by calling `sim_data()` that we defined. We then use sample to randomly split the `x_data` and `y` in to two groups using `sample()` method. After that, we call `fit_models` to get the simulated results of all models and store them in the pre-defined data structure. Finally, we count the number and store the count in another pre-defined data structure for future use.

Results

Since we won't look at all the values of RMSE, we average the RMSE scores of different models with different sigmas on different datset to get one single value:

```
avg_train_rmse = apply(results[, 1, , ], c(2, 3), mean)
avg_test_rmse = apply(results[, 2, , ], c(2, 3), mean)
```

Same use of two libraries:

```
library(ggplot2)
library(gridExtra)
```

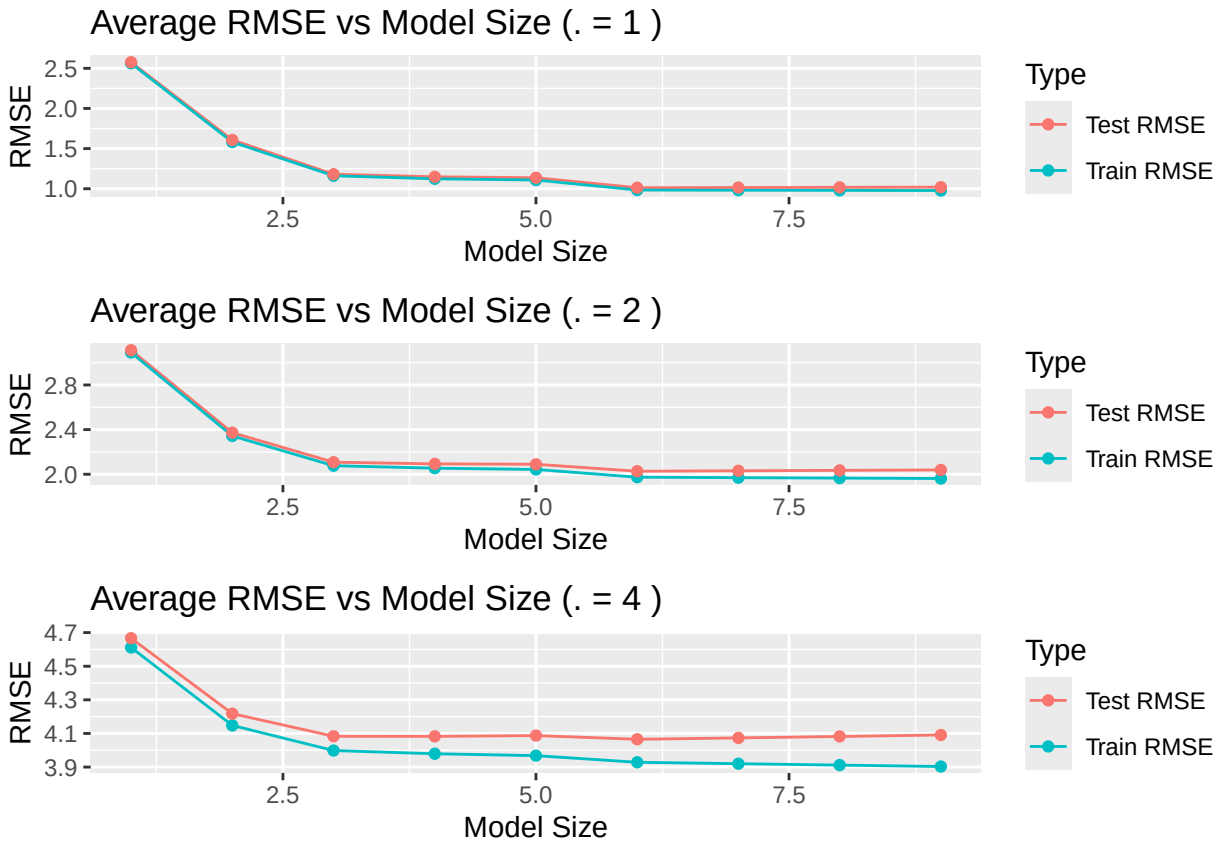
Then we can create a function dedicated for plotting RMSE by model for a specific sigma value:

```
create_rmse_plot = function(sigma_idx, sigma_val) {
  plot_data = data.frame(
    model_size = rep(1:9, 2),
    rmse = c(avg_train_rmse[, sigma_idx], avg_test_rmse[, sigma_idx]),
    type = factor(rep(c("Train RMSE", "Test RMSE"), each = 9))
  )

  ggplot(plot_data, aes(x = model_size, y = rmse, color = type)) +
    geom_line() +
    geom_point() +
    labs(title = paste("Average RMSE vs Model Size (sigma =", sigma_val, ")"),
         x = "Model Size",
         y = "RMSE",
         color = "Type")
}
```

First, a data frame for plotting is created. Then we plot the data using ggplot.

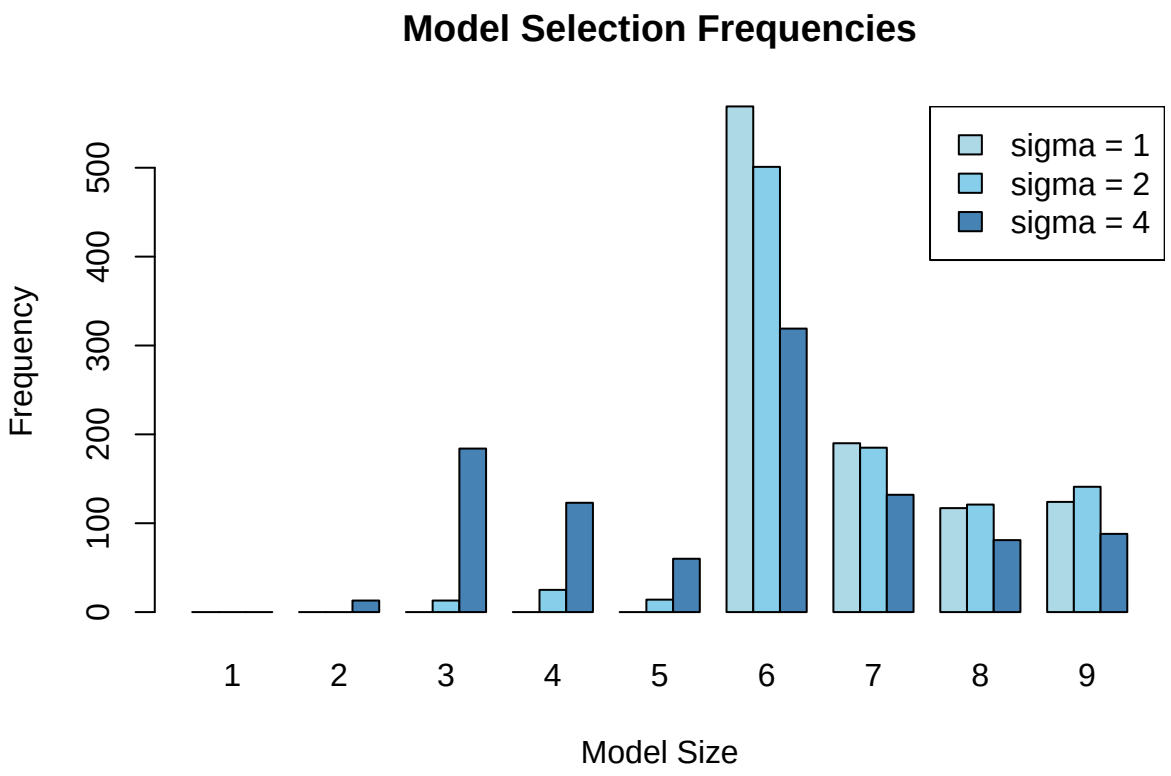
After the definition of the function, we can plot for all three different sigmas and place them horizontally for comparison:

```
plots = list()

for (i in 1:length(sigmas)) {
    plots[[i]] = create_rmse_plot(i, sigmas[i])
}

grid.arrange(grobs = plots, ncol = 1)
```



We can also plot the model selections:

```
barplot(model_selections,
        beside = TRUE,
        col = c("lightblue", "skyblue", "steelblue"),
        xlab = "Model Size",
        ylab = "Frequency",
        main = "Model Selection Frequencies",
        names.arg = 1:9)
legend("topright",
       legend = paste("sigma =", sigmas),
       fill = c("lightblue", "skyblue", "steelblue"))
```



Discussion

- It does not always select the correct model, but on average, it selects the correct model, which can be seen in both the plotting of model selection and averaged RMSE.
- When noise gets bigger:
  - Model selection is less accurate.
  - General RMSE increases among all models
  - The gap between training and test RMSE increases
  - Train RMSE shows more optimistic results than test RMSE
