Week 11 - Homework

STAT 420, Fall 2024, Banghao Chi

11/11/2024

Exercise 1 (`longley` Macroeconomic Data)

The built-in dataset `longley` contains macroeconomic data for predicting employment. We will attempt to model the `Employed` variable.

```r
View(longley)
?longley
```

(a) What is the largest correlation between any pair of predictors in the dataset?

```r
predictors = longley[, -which(names(longley) == "Employed")]
cor_matrix = cor(predictors)
max_cor = max(cor_matrix[lower.tri(cor_matrix)])
```

The largest correlation between any pair of the predictors in the dataset is 0.9953.

(b) Fit a model with `Employed` as the response and the remaining variables as predictors. Calculate and report the variance inflation factor (VIF) for each of the predictors. Which variable has the largest VIF? Do any of the VIFs suggest multicollinearity?

```r
library(car)
```

```
## Loading required package: carData
```

```r
full_model = lm(Employed ~ ., data = longley)
vif_values = vif(full_model)
vif_values
```

```
## GNP.deflator          GNP   Unemployed Armed.Forces   Population         Year
##      135.532     1788.513       33.619        3.589      399.151      758.981
```

As seen above, `GNP` has the larges VIF. Since the rule of thumb is that VIF values greater than 5 indicate problematic multicollinearity. AS all predictors except Armed.Forces (VIF = 3.589) have VIF values well above 5, the answer is yes, there are VIFs suggest multicollinearity.

(c) What proportion of the observed variation in `Population` is explained by a linear relationship with the other predictors?

```r
pop_model = lm(Population ~ GNP + Unemployed + Armed.Forces + Year + GNP.deflator, data = longley)
r_squared_pop = summary(pop_model)$r.squared
```

0.9975 of the observed variation in `Population` is explained by a linear relationship with the other predictors.

(d) Calculate the partial correlation coefficient for `Population` and `Employed` **with the effects of the other predictors removed**.

```r
model1 = lm(Employed ~ GNP + Unemployed + Armed.Forces + Year + GNP.deflator, data = longley)
resid1 = residuals(model1)

model2 = lm(Population ~ GNP + Unemployed + Armed.Forces + Year + GNP.deflator, data = longley)
resid2 = residuals(model2)

partial_cor = cor(resid1, resid2)
```

The partial correlation coefficient for `Population` and `Employed` **with the effects of the other predictors removed** is -0.0751.

(e) Fit a new model with `Employed` as the response and the predictors from the model in **(b)** that were significant. (Use $\alpha = 0.05$.) Calculate and report the variance inflation factor for each of the predictors. Which variable has the largest VIF? Do any of the VIFs suggest multicollinearity?

```r
summary(full_model)
```

```
##
## Call:
## lm(formula = Employed ~ ., data = longley)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4101 -0.1577 -0.0282  0.1016  0.4554
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -3.48e+03   8.90e+02   -3.91  0.00356 **
## GNP.deflator  1.51e-02   8.49e-02    0.18  0.86314
## GNP          -3.58e-02   3.35e-02   -1.07  0.31268
## Unemployed   -2.02e-02   4.88e-03   -4.14  0.00254 **
## Armed.Forces -1.03e-02   2.14e-03   -4.82  0.00094 ***
## Population   -5.11e-02   2.26e-01   -0.23  0.82621
## Year          1.83e+00   4.55e-01    4.02  0.00304 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.305 on 9 degrees of freedom
## Multiple R-squared:  0.995,  Adjusted R-squared:  0.992
## F-statistic:  330 on 6 and 9 DF,  p-value: 4.98e-10
```

As seen in the model summary, `Unemployed`, `Armed.Forces` and `Year` are significant.

```r
reduced_model = lm(Employed ~ Unemployed + Armed.Forces + Year, data = longley)
vif_values_reduced = vif(reduced_model)
vif_values_reduced
```

```
##   Unemployed Armed.Forces         Year
##        3.318        2.223        3.891
```

`Year` has the largest VIF, but there's no VIFs that suggest multicollinearity.

(f) Use an F-test to compare the models in parts **(b)** and **(e)**. Report the following:

- The null hypothesis
- The test statistic
- The distribution of the test statistic under the null hypothesis
- The p-value

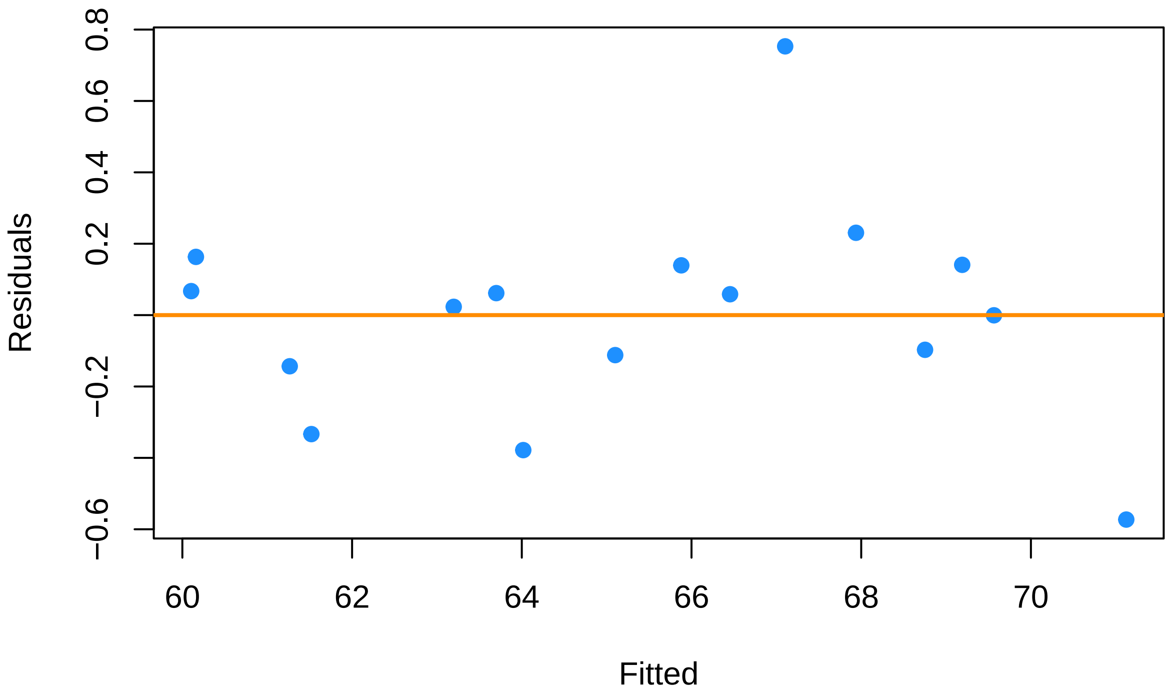
- A decision
- Which model you prefer, **(b)** or **(e)**

```
anova_result = anova(reduced_model, full_model)
anova_result
```

```
## Analysis of Variance Table
##
## Model 1: Employed ~ Unemployed + Armed.Forces + Year
## Model 2: Employed ~ GNP.deflator + GNP + Unemployed + Armed.Forces + Population +
##     Year
##   Res.Df   RSS Df Sum of Sq    F Pr(>F)
## 1     12 1.323
## 2      9 0.836  3     0.487 1.75   0.23
```

- The null hypothesis: The coefficients of the removed predictor is 0.
- The test statistic: F = 1.75
- The distribution of the test statistic under the null hypothesis: F ~ F(3, 9)
- The p-value: p = 0.23
- A decision
- Which model you prefer, **(b)** or **(e)**:
  - Since p-value = 0.23 > α = 0.05, we fail to reject the null hypothesis, meaning we prefer the reduced smaller model.

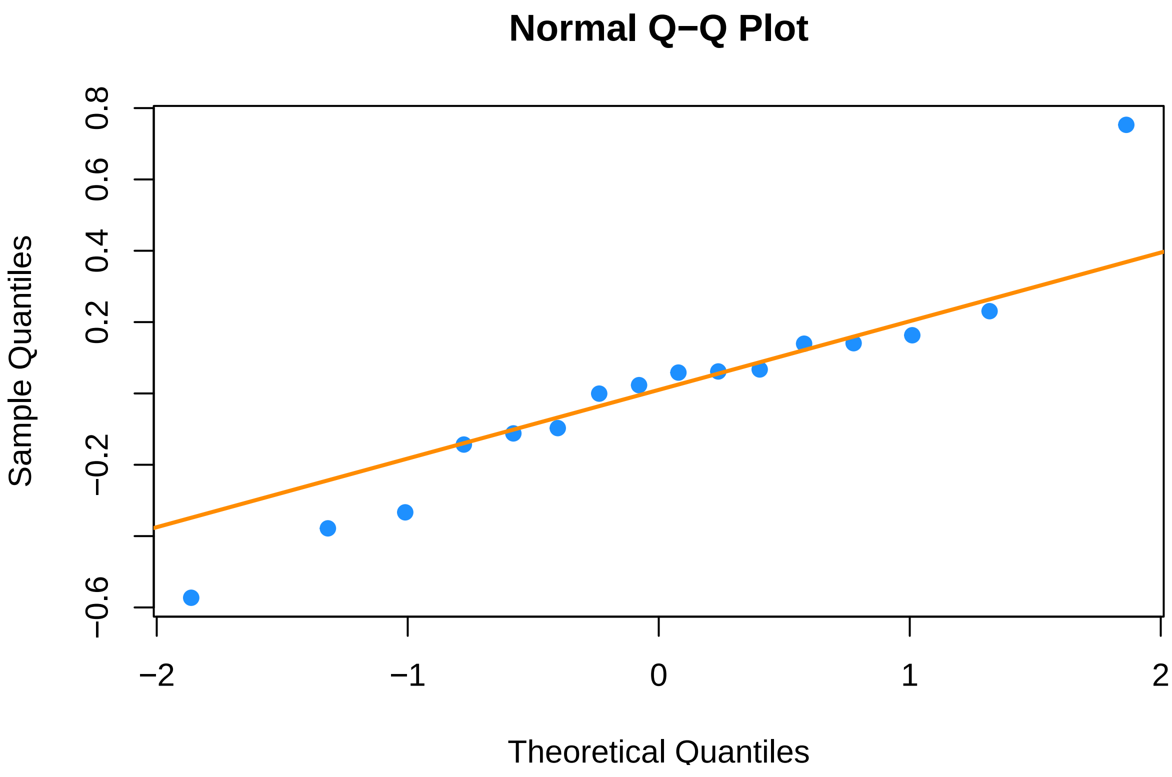
(g) Check the assumptions of the model chosen in part **(f)**. Do any assumptions appear to be violated?



```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```



```
##
##  studentized Breusch-Pagan test
##
## data:  reduced_model
## BP = 2.5, df = 3, p-value = 0.5
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid(reduced_model)
## W = 0.93, p-value = 0.2
```

Yes, from both plots and tests, the equal variance assumption and normality assumption both seem to be violated.

Exercise 2 (`Credit` Data)

For this exercise, use the `Credit` data from the `ISLR` package. Use the following code to remove the `ID` variable which is not useful for modeling.

```
library(ISLR)
data(Credit)
Credit = subset(Credit, select = -c(ID))
```

Use `?Credit` to learn about this dataset.

(a) Find a "good" model for `balance` using the available predictors. Use any methods seen in class except transformations of the response. The model should:

- Reach a LOOCV-RMSE below `140`
- Obtain an adjusted R^2 above `0.90`

- Fail to reject the Breusch-Pagan test with an α of 0.01
- Use fewer than 10 β parameters

Store your model in a variable called `mod_a`. Run the two given chunks to verify your model meets the requested criteria. If you cannot find a model that meets all criteria, partial credit will be given for meeting at least some of the criteria.

```
library(lmtest)

get_bp_decision = function(model, alpha) {
  decide = unname(bptest(model)$p.value < alpha)
  ifelse(decide, "Reject", "Fail to Reject")
}

get_sw_decision = function(model, alpha) {
  decide = unname(shapiro.test(resid(model))$p.value < alpha)
  ifelse(decide, "Reject", "Fail to Reject")
}

get_num_params = function(model) {
  length(coef(model))
}

get_loocv_rmse = function(model) {
  sqrt(mean((resid(model) / (1 - hatvalues(model))) ^ 2))
}

get_adj_r2 = function(model) {
  summary(model)$adj.r.squared
}
```

```
mod_a = lm(Balance ~ Limit + log(Income) + Cards + Age + Education + Gender + Student + Married, data =

get_loocv_rmse(mod_a)
```

```
## [1] 131.5
```

```
get_adj_r2(mod_a)
```

```
## [1] 0.9206
```

```
get_bp_decision(mod_a, alpha = 0.01)
```

```
## [1] "Fail to Reject"
```

```
get_num_params(mod_a)
```

```
## [1] 9
```

(b) Find another "good" model for `balance` using the available predictors. Use any methods seen in class except transformations of the response. The model should:

- Reach a LOOCV-RMSE below `130`
- Obtain an adjusted R^2 above `0.85`
- Fail to reject the Shapiro-Wilk test with an α of 0.01
- Use fewer than 25 β parameters

Store your model in a variable called `mod_b`. Run the two given chunks to verify your model meets the requested criteria. If you cannot find a model that meets all criteria, partial credit will be given for meeting at least some of the criteria.

```r
library(lmtest)

get_bp_decision = function(model, alpha) {
  decide = unname(bptest(model)$p.value < alpha)
  ifelse(decide, "Reject", "Fail to Reject")
}

get_sw_decision = function(model, alpha) {
  decide = unname(shapiro.test(resid(model))$p.value < alpha)
  ifelse(decide, "Reject", "Fail to Reject")
}

get_num_params = function(model) {
  length(coef(model))
}

get_loocv_rmse = function(model) {
  sqrt(mean((resid(model) / (1 - hatvalues(model))) ^ 2))
}

get_adj_r2 = function(model) {
  summary(model)$adj.r.squared
}
```

```r
mod_b = lm(Balance ~ Income + Limit + Rating + Cards + Age + Education + Gender + Student + Married + E

get_loocv_rmse(mod_b)
```

```
## [1] 65.87
```

```r
get_adj_r2(mod_b)
```

```
## [1] 0.9821
```

```r
get_sw_decision(mod_b, alpha = 0.01)
```

```
## [1] "Fail to Reject"
```

```r
get_num_params(mod_b)
```

```
## [1] 24
```

Exercise 3 (Sacramento Housing Data)

For this exercise, use the `Sacramento` data from the `caret` package. Use the following code to perform some preprocessing of the data.

```r
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```r
library(ggplot2)
data(Sacramento)
sac_data = Sacramento
```

```r
sac_data$limits = factor(ifelse(sac_data$city == "SACRAMENTO", "in", "out"))
sac_data = subset(sac_data, select = -c(city, zip))
```

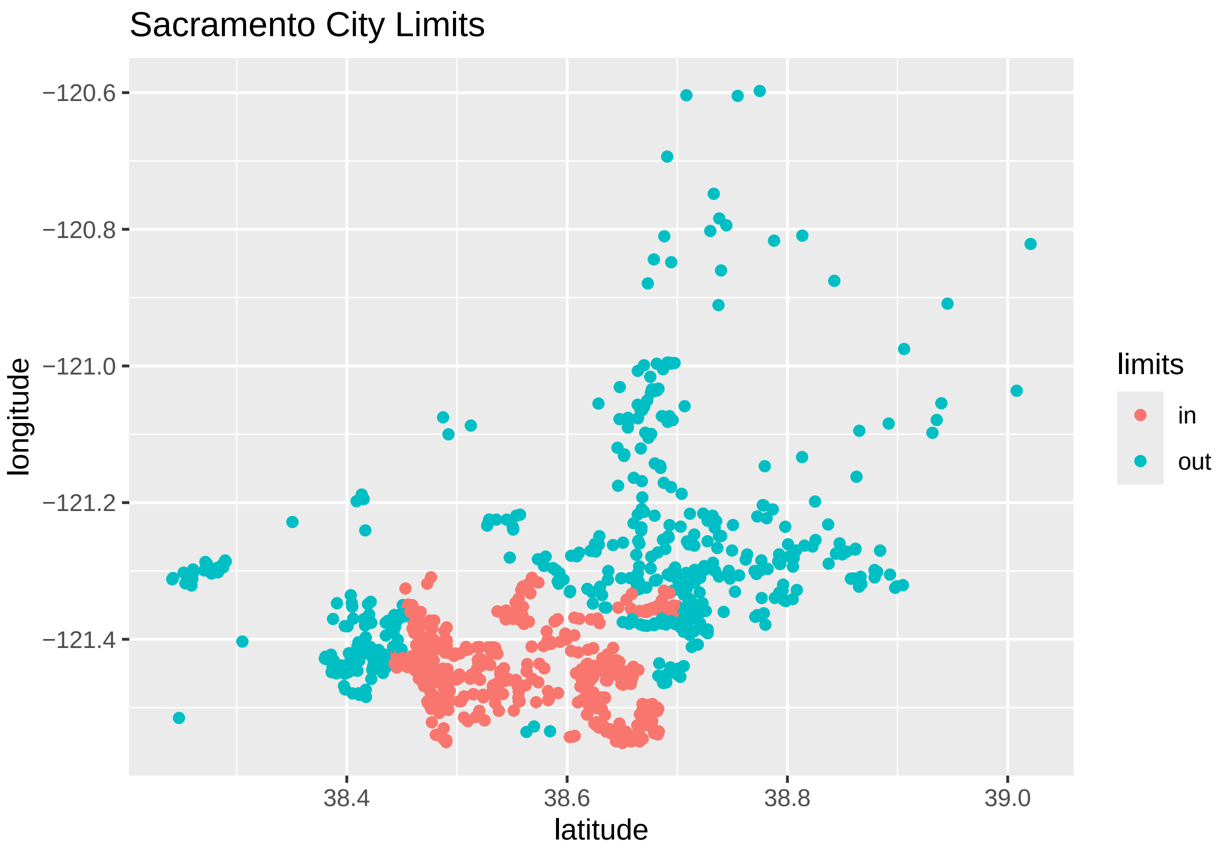
Instead of using the `city` or `zip` variables that exist in the dataset, we will simply create a variable (`limits`) indicating whether or not a house is technically within the city limits of Sacramento. (We do this because they would both be factor variables with a **large** number of levels. This is a choice that is made due to laziness, not necessarily because it is justified. Think about what issues these variables might cause.)

Use `?Sacramento` to learn more about this dataset.

A plot of longitude versus latitude gives us a sense of where the city limits are.

```r
qplot(y = longitude, x = latitude, data = sac_data,
      col = limits, main = "Sacramento City Limits ")
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



After these modifications, we test-train split the data.

```r
set.seed(420)
sac_trn_idx  = sample(nrow(sac_data), size = trunc(0.80 * nrow(sac_data)))
sac_trn_data = sac_data[sac_trn_idx, ]
sac_tst_data = sac_data[-sac_trn_idx, ]
```

The training data should be used for all model fitting. Our goal is to find a model that is useful for predicting home prices.

(a) Find a "good" model for `price`. Use any methods seen in class. The model should reach a LOOCV-RMSE below 77,500 in the training data. Do not use any transformations of the response variable.

```
calc_loocv_rmse = function(model) {
  sqrt(mean((resid(model) / (1 - hatvalues(model)))^2))
}

model = lm(price ~ . + I(sqft^2) + I(beds * baths), data = sac_trn_data)

rmse = calc_loocv_rmse(model)
rmse
```

```
## [1] 77472
```

(b) Is a model that achieves a LOOCV-RMSE below 77,500 useful in this case? That is, is an average error of 77,500 low enough when predicting home prices? To further investigate, use the held-out test data and your model from part **(a)** to do two things:

- Calculate the average percent error:

$$\frac{1}{n}\sum_i \frac{|\text{predicted}_i - \text{actual}_i|}{\text{predicted}_i} \times 100$$

- Plot the predicted versus the actual values and add the line $y = x$.

Based on all of this information, argue whether or not this model is useful.

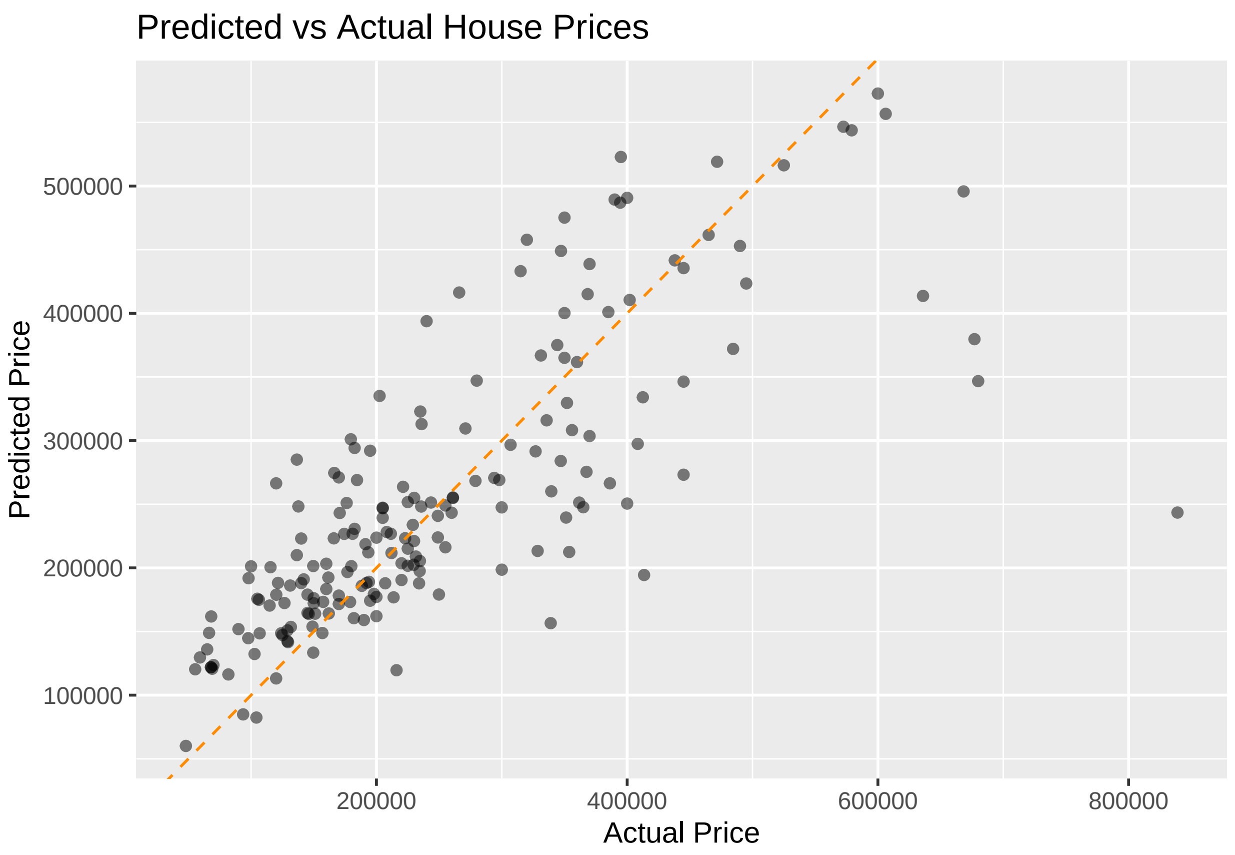
```
predictions = predict(model, newdata = sac_tst_data)
actual = sac_tst_data$price

avg_percent_error = mean(abs(predictions - actual) / predictions * 100)
avg_percent_error
```

```
## [1] 24.56
```

```
pred_vs_actual = ggplot(data.frame(predicted = predictions, actual = actual), aes(x = actual, y = predi
  geom_point(alpha = 0.5) +
  geom_abline(intercept = 0, slope = 1, color = "darkorange", linetype = "dashed") +
  labs(x = "Actual Price", y = "Predicted Price",
       title = "Predicted vs Actual House Prices")

print(pred_vs_actual)
```



As we can see from both the average percent error and the plot, the model might be more useful for rough estimates or identifying general price ranges rather than precise valuations since it has error roughly around 24.5% and the point deviates from the orange line.

Therefore, While the model meets the technical RMSE threshold, I would argue that this model might not be useful in predicting precise house prices.

Exercise 4 (Does It Work?)

In this exercise, we will investigate how well backwards AIC and BIC actually perform. For either to be "working" correctly, they should result in a low number of both **false positives** and **false negatives**. In model selection,

- **False Positive**, FP: Incorrectly including a variable in the model. Including a *non-significant* variable
- **False Negative**, FN: Incorrectly excluding a variable in the model. Excluding a *significant* variable

Consider the **true** model

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \beta_6 x_6 + \beta_7 x_7 + \beta_8 x_8 + \beta_9 x_9 + \beta_{10} x_{10} + \epsilon$$

where $\epsilon \sim N(0, \sigma^2 = 4)$. The true values of the β parameters are given in the `R` code below.

```
beta_0  = 1
beta_1  = -1
beta_2  = 2
beta_3  = -2
beta_4  = 1
```

```r
beta_5  = 1
beta_6  = 0
beta_7  = 0
beta_8  = 0
beta_9  = 0
beta_10 = 0
sigma = 2
```

Then, as we have specified them, some variables are significant, and some are not. We store their names in `R` variables for use later.

```r
not_sig  = c("x_6", "x_7", "x_8", "x_9", "x_10")
signif = c("x_1", "x_2", "x_3", "x_4", "x_5")
```

We now simulate values for these `x` variables, which we will use throughout part **(a)**.

```r
set.seed(420)
n = 100
x_1  = runif(n, 0, 10)
x_2  = runif(n, 0, 10)
x_3  = runif(n, 0, 10)
x_4  = runif(n, 0, 10)
x_5  = runif(n, 0, 10)
x_6  = runif(n, 0, 10)
x_7  = runif(n, 0, 10)
x_8  = runif(n, 0, 10)
x_9  = runif(n, 0, 10)
x_10 = runif(n, 0, 10)
```

We then combine these into a data frame and simulate `y` according to the true model.

```r
sim_data_1 = data.frame(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10,
  y = beta_0 + beta_1 * x_1 + beta_2 * x_2 + beta_3 * x_3 + beta_4 * x_4 +
      beta_5 * x_5 + rnorm(n, 0 , sigma)
)
```

We do a quick check to make sure everything looks correct.

```r
head(sim_data_1)
```

```
##     x_1   x_2    x_3    x_4    x_5   x_6    x_7    x_8   x_9   x_10       y
## 1 6.055 4.088 8.7894 1.8180 0.8198 8.146 9.7305 9.6673 6.915 4.5523 -11.627
## 2 9.703 3.634 5.0768 5.5784 6.3193 6.033 3.2301 2.6707 2.214 0.4861  -0.147
## 3 1.745 3.899 0.5431 4.5068 1.0834 3.427 3.2223 5.2746 8.242 7.2310  15.145
## 4 4.758 5.315 7.6257 0.1287 9.4057 6.168 0.2472 6.5325 2.102 4.5814   2.404
## 5 7.245 7.225 9.5763 3.0398 0.4194 5.937 9.2169 4.6228 2.527 9.2349  -7.910
## 6 8.761 5.177 1.7983 0.5949 9.2944 9.392 1.0017 0.4476 5.508 5.9687   9.764
```

Now, we fit an incorrect model.

```r
fit = lm(y ~ x_1 + x_2 + x_6 + x_7, data = sim_data_1)
coef(fit)
```

```
## (Intercept)         x_1         x_2         x_6         x_7
##     -1.3758     -0.3572      2.1040      0.1344     -0.3367
```

Notice, we have coefficients for `x_1`, `x_2`, `x_6`, and `x_7`. This means that `x_6` and `x_7` are false positives, while `x_3`, `x_4`, and `x_5` are false negatives.

To detect the false negatives, use:

```r
# which are false negatives?
!(signif %in% names(coef(fit)))
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

To detect the false positives, use:

```r
# which are false positives?
names(coef(fit)) %in% not_sig
```

```
## [1] FALSE FALSE FALSE  TRUE  TRUE
```

Note that in both cases, you could `sum()` the result to obtain the number of false negatives or positives.

(a) Set a seed equal to your birthday; then, using the given data for each `x` variable above in `sim_data_1`, simulate the response variable `y` 300 times. Each time,

- Fit an additive model using each of the `x` variables.
- Perform variable selection using backwards AIC.
- Perform variable selection using backwards BIC.
- Calculate and store the number of false negatives for the models chosen by AIC and BIC.
- Calculate and store the number of false positives for the models chosen by AIC and BIC.

Calculate the rate of false positives and negatives for both AIC and BIC. Compare the rates between the two methods. Arrange your results in a well formatted table.

```r
calculate_errors = function(selected_vars, signif, not_sig) {
  selected_vars = selected_vars[selected_vars != "(Intercept)"]

  fn = sum(!(signif %in% selected_vars))

  fp = sum(selected_vars %in% not_sig)

  return(c(fp = fp, fn = fn))
}
```

```r
sim_a = function(seed, n_sims = 300) {
  set.seed(seed)
  n = 100

  x_1  = runif(n, 0, 10)
  x_2  = runif(n, 0, 10)
  x_3  = runif(n, 0, 10)
  x_4  = runif(n, 0, 10)
  x_5  = runif(n, 0, 10)
  x_6  = runif(n, 0, 10)
  x_7  = runif(n, 0, 10)
  x_8  = runif(n, 0, 10)
  x_9  = runif(n, 0, 10)
  x_10 = runif(n, 0, 10)

  results_aic = matrix(0, nrow = n_sims, ncol = 2)
  results_bic = matrix(0, nrow = n_sims, ncol = 2)

  for(i in 1:n_sims) {
    y = beta_0 + beta_1 * x_1 + beta_2 * x_2 + beta_3 * x_3 +
        beta_4 * x_4 + beta_5 * x_5 + rnorm(n, 0, sigma)
    sim_data = data.frame(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10, y)

    full_model = lm(y ~ ., data = sim_data)

    aic_model = step(full_model, direction = "backward", trace = 0)
    bic_model = step(full_model, direction = "backward", k = log(n), trace = 0)

    results_aic[i,] = calculate_errors(names(coef(aic_model)), signif, not_sig)
    results_bic[i,] = calculate_errors(names(coef(bic_model)), signif, not_sig)
  }

  data.frame(
    Method = c("AIC", "BIC"),
    FP_Rate = c(mean(results_aic[,1])/length(not_sig),
                mean(results_bic[,1])/length(not_sig)),
    FN_Rate = c(mean(results_aic[,2])/length(signif),
                mean(results_bic[,2])/length(signif))
  )
}

sim_a(1023)
```

```
##   Method FP_Rate FN_Rate
## 1    AIC 0.18333       0
## 2    BIC 0.03733       0
```

(b) Set a seed equal to your birthday; then, using the given data for each `x` variable below in `sim_data_2`, simulate the response variable `y` 300 times. Each time,

- Fit an additive model using each of the `x` variables.
- Perform variable selection using backwards AIC.
- Perform variable selection using backwards BIC.
- Calculate and store the number of false negatives for the models chosen by AIC and BIC.
- Calculate and store the number of false positives for the models chosen by AIC and BIC.

Calculate the rate of false positives and negatives for both AIC and BIC. Compare the rates between the two methods. Arrange your results in a well formatted table. Also compare to your answers in part **(a)** and suggest a reason for any differences.

```
set.seed(94)
x_1  = runif(n, 0, 10)
x_2  = runif(n, 0, 10)
x_3  = runif(n, 0, 10)
x_4  = runif(n, 0, 10)
x_5  = runif(n, 0, 10)
x_6  = runif(n, 0, 10)
x_7  = runif(n, 0, 10)
x_8  = x_1 + rnorm(n, 0, 0.1)
x_9  = x_1 + rnorm(n, 0, 0.1)
x_10 = x_2 + rnorm(n, 0, 0.1)

sim_data_2 = data.frame(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10,
  y = beta_0 + beta_1 * x_1 + beta_2 * x_2 + beta_3 * x_3 + beta_4 * x_4 +
      beta_5 * x_5 + rnorm(n, 0 , sigma)
)
```

```r
sim_b = function(seed, n_sims = 300) {
  set.seed(seed)
  n = 100

  x_1  = runif(n, 0, 10)
  x_2  = runif(n, 0, 10)
  x_3  = runif(n, 0, 10)
  x_4  = runif(n, 0, 10)
  x_5  = runif(n, 0, 10)
  x_6  = runif(n, 0, 10)
  x_7  = runif(n, 0, 10)
  x_8  = x_1 + rnorm(n, 0, 0.1)
  x_9  = x_1 + rnorm(n, 0, 0.1)
  x_10 = x_2 + rnorm(n, 0, 0.1)

  results_aic = matrix(0, nrow = n_sims, ncol = 2)
  results_bic = matrix(0, nrow = n_sims, ncol = 2)

  for(i in 1:n_sims) {
    y = beta_0 + beta_1 * x_1 + beta_2 * x_2 + beta_3 * x_3 +
        beta_4 * x_4 + beta_5 * x_5 + rnorm(n, 0, sigma)

    sim_data = data.frame(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_10, y)

    full_model = lm(y ~ ., data = sim_data)

    aic_model = step(full_model, direction = "backward", trace = 0)
    bic_model = step(full_model, direction = "backward", k = log(n), trace = 0)

    results_aic[i,] = calculate_errors(names(coef(aic_model)), signif, not_sig)
    results_bic[i,] = calculate_errors(names(coef(bic_model)), signif, not_sig)
  }

  data.frame(
    Method = c("AIC", "BIC"),
    FP_Rate = c(mean(results_aic[,1])/length(not_sig),
                mean(results_bic[,1])/length(not_sig)),
    FN_Rate = c(mean(results_aic[,2])/length(signif),
                mean(results_bic[,2])/length(signif))
  )
}

sim_b(1023)
```

```
##   Method FP_Rate FN_Rate
## 1    AIC  0.3173  0.1600
## 2    BIC  0.2127  0.1713
```

- Differences: Both rates in part (b) are larger than those from part (a) no matter the criterion is AIC or BIC.
- Reasons: In Part B, x__8 and x__9 are highly correlated with x__1, and x__10 is correlated with x__2. This multicollinearity makes it harder for both methods to distinguish between significant and non-significant variables, since when predictors are correlated, they can “mask” each other’s importance, leading to both more false positives (including correlated non-significant variables) and false negatives (excluding

significant variables due to their correlation with included variables).