Week 9 - Homework

STAT 420, Summer 2018, Connor Segneri - segneri3

Contents

Exercise 1 (`longley` Macroeconomic Data)

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

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

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

```
library(knitr)
library(kableExtra)

round(cor(longley), 3) %>%
  kable() %>%
  kable_styling()
```

	GNP.deflator	GNP	Unemployed	Armed.Forces	Population	Year	Employed
GNP.deflator	1.000	0.992	0.621	0.465	0.979	0.991	0.971
GNP	0.992	1.000	0.604	0.446	0.991	0.995	0.984
Unemployed	0.621	0.604	1.000	-0.177	0.687	0.668	0.502
Armed.Forces	0.465	0.446	-0.177	1.000	0.364	0.417	0.457
Population	0.979	0.991	0.687	0.364	1.000	0.994	0.960
Year	0.991	0.995	0.668	0.417	0.994	1.000	0.971
Employed	0.971	0.984	0.502	0.457	0.960	0.971	1.000

Based on the table above, the highest correlation is between `Year` and `GNP`.

(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?

```
library(car)

model_b = lm(Employed ~ ., data = longley)
vif(model_b)
```

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

The variable with the highest vif of `1788.5135` is `GNP`. There is a huge multicollinearity issue with the model, as any of the vif values above 5 are cause for concern.

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

```
population_model = lm(Population ~ . - Employed, data = longley)
```

The r-squared value of the model above explains the proportion of observed variation in the predictor `Population` explained by the other predictors. This r-squared value is `0.9975`.

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

```
cor(resid(model_b), resid(population_model))
```

```
## [1] -2.54e-17
```

(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?

```
summary(model_b)$coefficients
```

```
##                 Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  -3482.25863 890.420384 -3.9108 0.0035604
## GNP.deflator     0.01506   0.084915  0.1774 0.8631408
## GNP             -0.03582   0.033491 -1.0695 0.3126811
## Unemployed      -0.02020   0.004884 -4.1364 0.0025351
## Armed.Forces    -0.01033   0.002143 -4.8220 0.0009444
## Population      -0.05110   0.226073 -0.2261 0.8262118
## Year             1.82915   0.455478  4.0159 0.0030368
```

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

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

The variable with the largest vif value is `Year` with a vif of `3.8909`. All of these vif values are below five, and do not 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(model_b, model_e)
```

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

The null hypothesis for the anova test above is if the additional variables in model 1 do not bring significant value to the model.

The test statistic is `1.7465`.

The distribution of the test statistic under the null hypothesis is null.

The p-value is `0.227`.

A decision under any reasonable α is to reject the null hypothesis. The additional variables in model one do improve the fit of the model.

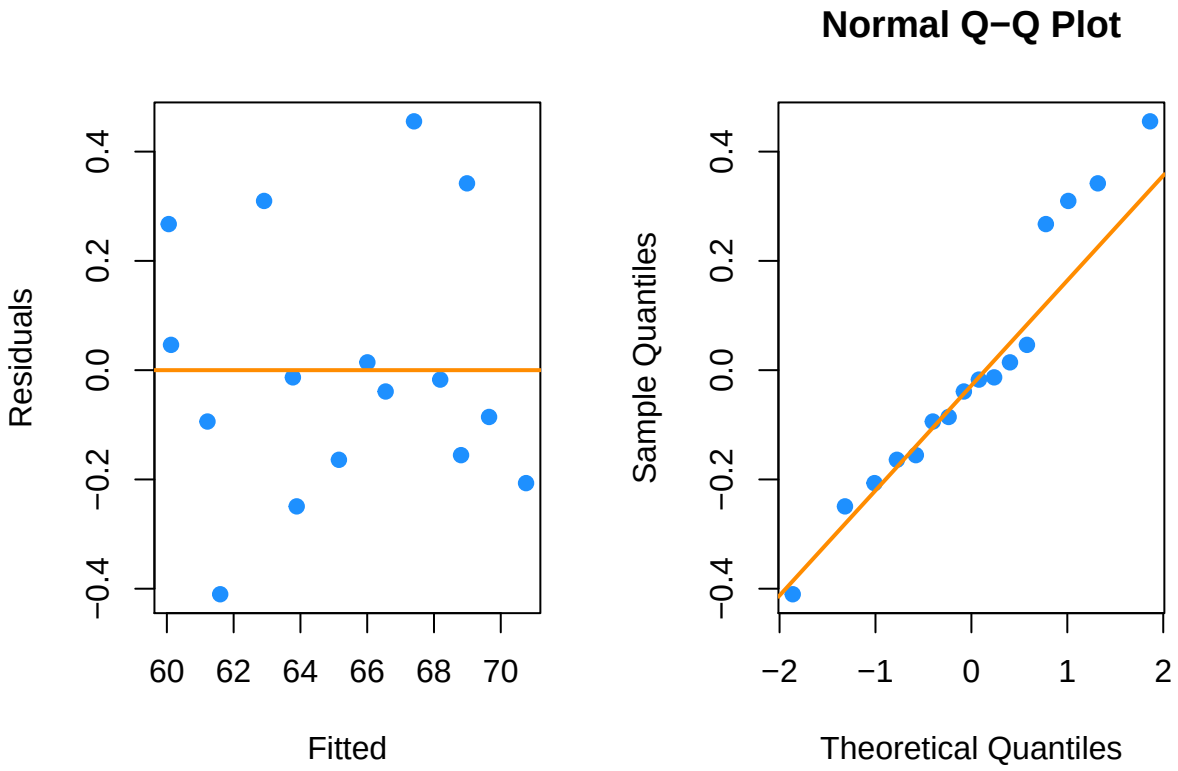
Based on just the results of this hypothesis, I would choose the first model with all of the predictors.

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

```
plot_fitted_resid = function(model, pointcol = "dodgerblue", linecol = "darkorange") {
  plot(fitted(model), resid(model),
       col = pointcol, pch = 20, cex = 1.5,
       xlab = "Fitted", ylab = "Residuals")
  abline(h = 0, col = linecol, lwd = 2)
}

plot_qq = function(model, pointcol = "dodgerblue", linecol = "darkorange") {
  qqnorm(resid(model), col = pointcol, pch = 20, cex = 1.5)
  qqline(resid(model), col = linecol, lwd = 2)
}
```

```
par(mfrow = c(1, 2))
plot_fitted_resid(model_b)
plot_qq(model_b)
```



The fitted versus residuals plot appears to portray a non-constant variance. Also the normality assumptions seems slightly suspect based ont he normal q-q plot.

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 `135`
- 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
}
```

```
model_full = lm(Balance ~ ., data = Credit)
mod_a = step(model_full, direction = "backward", trace = 0)
```

```
get_loocv_rmse(mod_a)
```

```
## [1] 99.54
```

```
get_adj_r2(mod_a)
```

```
## [1] 0.954
```

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

```
## [1] "Reject"
```

```
get_num_params(mod_a)
```

```
## [1] 7
```

(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 `125`
- Obtain an adjusted R^2 above `0.91`
- 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.

```
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
}
```

```
model_full = lm(Balance ~ .^2, data = Credit)
mod_b = step(model_full, direction = "backward", k = log(dim(Credit)[1]), trace = 0)
```

```
get_loocv_rmse(mod_b)
```

```
## [1] 66.22
```

```
get_adj_r2(mod_b)
```

```
## [1] 0.9809
```

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

```
## [1] "Reject"
```

```
get_num_params(mod_b)
```

```
## [1] 11
```

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.

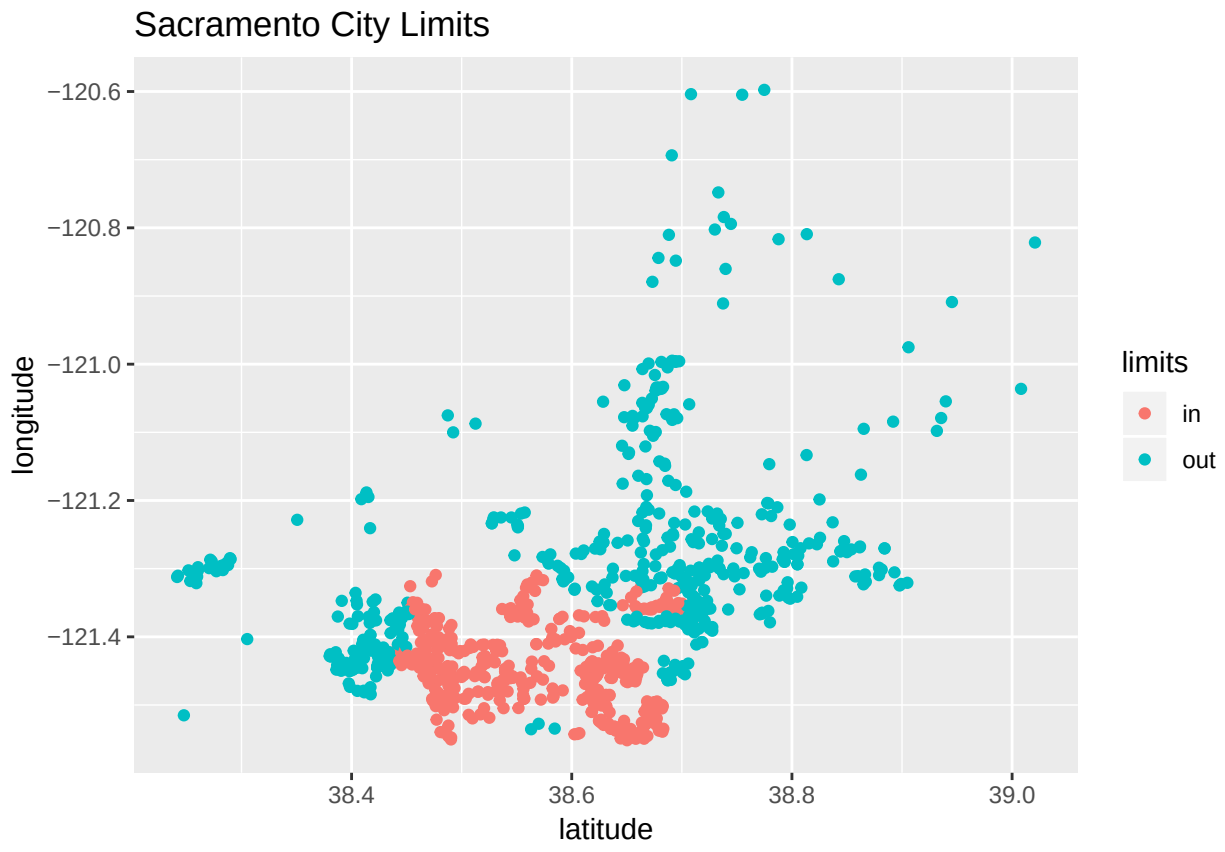
```
library(caret)
library(ggplot2)
data(Sacramento)
sac_data = Sacramento
sac_data$limits = factor(ifelse(sac_data$city == "SACRAMENTO", "in", "out"))
sac_data = subset(sac_data, select = -c(city, zip))
```

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.

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



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

```
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.

```
model_big = lm(price ~ .^2+I(beds^2)+I(baths^2)+I(sqft^2), data = sac_trn_data)
model_back_aic = step(model_big, direction = "backward", trace = 0)

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

The LOOCV-RMSE of this model is `77044.8998`.

(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:

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

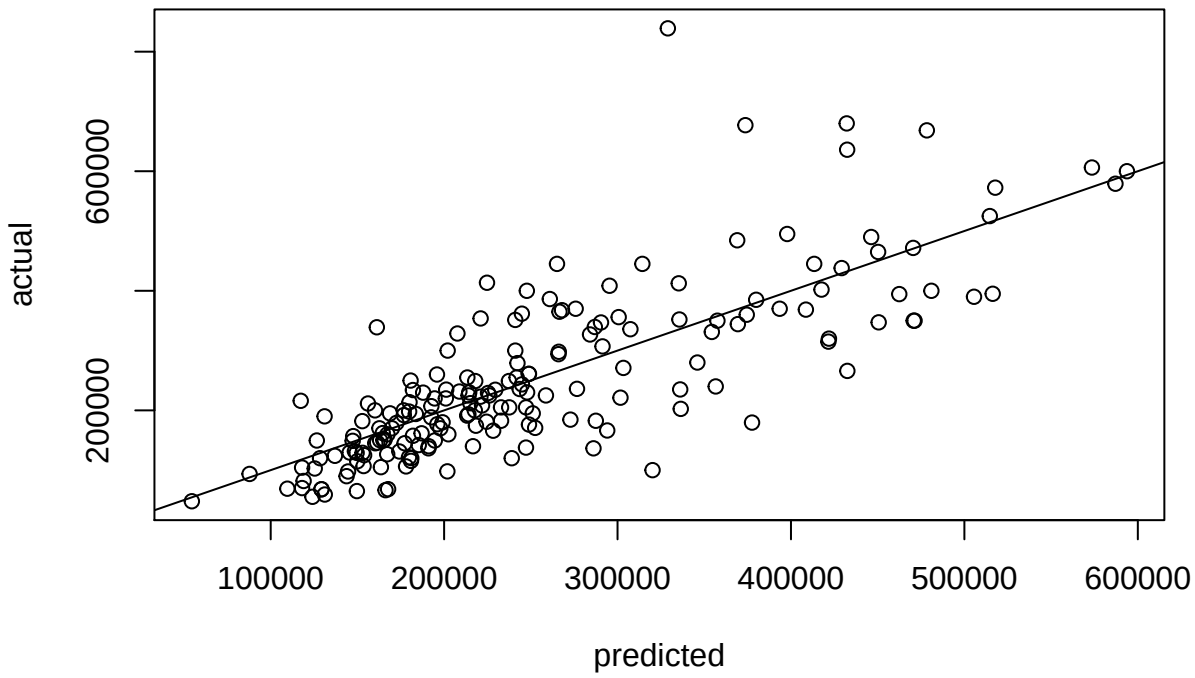
- Plot the predicted versus the actual values and add the line $y = x$.

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

```
actual = sac_tst_data$price
predicted = predict(model_back_aic, newdata = sac_tst_data)

average_percent_error = (1/dim(sac_trn_data)[1]) * sum( abs(predicted - actual) / predicted ) * 100

plot(predicted, actual)
abline(a=0, b=1)
```



The average percent error is `5.9952%`.

The model is useful because the average percent error is so low. While the variance of the points in the plot above is higher than we would like, it still follows the x=y line, meaning that the predicted and the actual values trend one to one.

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
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.

```
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)**.

```
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.

```
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.

```
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.

```
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:

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

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

To detect the false positives, use:

```
# 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.

```
set.seed(19951015)
aic_false = data.frame(negatives = rep(0,300), positives = rep(0,300))
bic_false = data.frame(negatives = rep(0,300), positives = rep(0,300))
for (i in 1:300) {
  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 = 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)
  )
  model_a = lm(y ~ ., data = sim_data)
  back_aic = step(model_a, direction = "backward", trace = 0)
  back_bic = step(model_a, direction = "backward", trace = 0, k = log(dim(sim_data)[1]))

  aic_false$negatives[i] = sum(!(signif %in% names(coef(back_aic))))
```

```
  aic_false$positives[i] = sum(names(coef(back_aic)) %in% not_sig)
  bic_false$negatives[i] = sum(!(signif %in% names(coef(back_bic))))
  bic_false$positives[i] = sum(names(coef(back_bic)) %in% not_sig)
}

results = data.frame(
  aic = c(
    mean(aic_false$negatives) / 300,
    mean(aic_false$positives) / 300
  ),
  bic = c(
    mean(bic_false$negatives) / 300,
    mean(bic_false$positives) / 300
  )
)
rownames(results) = c("negatives", "positives")
results %>%
  kable() %>%
  kable_styling() %>%
  add_header_above(c("AIC and BIC False Positive and False Negative Rates" = 3))
```

AIC and BIC False Positive and False Negative Rates		
	aic	bic
negatives	0.0000	0.0000
positives	0.0031	0.0008

(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(420)
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)
```

```r
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)
)

set.seed(19951015)
aic_false = data.frame(negatives = rep(0,300), positives = rep(0,300))
bic_false = data.frame(negatives = rep(0,300), positives = rep(0,300))
for (i in 1:300) {
  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)
    )
  model_a = lm(y ~ ., data = sim_data)
  back_aic = step(model_a, direction = "backward", trace = 0)
  back_bic = step(model_a, direction = "backward", trace = 0, k = log(dim(sim_data)[1]))

  aic_false$negatives[i] = sum(!(signif %in% names(coef(back_aic))))
  aic_false$positives[i] = sum(names(coef(back_aic)) %in% not_sig)
  bic_false$negatives[i] = sum(!(signif %in% names(coef(back_bic))))
  bic_false$positives[i] = sum(names(coef(back_bic)) %in% not_sig)
}

results = data.frame(
  aic = c(
    mean(aic_false$negatives) / 300,
    mean(aic_false$positives) / 300
  ),
  bic = c(
    mean(bic_false$negatives) / 300,
    mean(bic_false$positives) / 300
  )
)
rownames(results) = c("negatives", "positives")
results %>%
  kable() %>%
  kable_styling() %>%
  add_header_above(c("AIC and BIC False Positive and False Negative Rates" = 3))
```

AIC and BIC False Positive and False Negative Rates		
	aic	bic
negatives	0	0
positives	0	0

This time there were no false positives or negatives. This might be because three of the non-significant variables `x_8`, `x_9`, and `x_10`, were variables of the other significant variables, `x_1` and `x_2`. This would cause AIC and BIC to weed these non-significant variables out early on because it would detect collinearity.