1. Use all-subsets regression.

    (a) **Report the variables in the best model of each size.**

```r
### The model.matrix lets us specify the regression formula we want
### to use, and outputs the corresponding model matrix
data.matrix = model.matrix(Ozone ~ .^2, data = data.train)

### We also need the response variable (i.e. alcohol)
Y.train = data.train$Ozone

### Now we can run all.subsets. There are a couple of extra inputs
### here. nvmax is the largest number of variables we are willing
### to include. 30 seems like plenty. intercept specifies whether we
### want regsubsets() to add an intercept term. Since model.matrix()
### already adds an intercept, we don't want regsubsets() to add
### another one.
all.subsets = regsubsets(x = data.matrix, y = Y.train, nvmax = 30)

### The output of regsubsets isn't really useful. We need to run the
### summary() function on it to get useful information
info.subsets = summary(all.subsets)

### The output of summary contains an array with columns corresponding
### to predictors and rows corresponding to model sizes. This array
### tells us which variables are included at each size.
all.subsets.models = info.subsets$which
all.subsets.models = all.subsets.models[, -1]
all.subsets.models
dim(all.subsets.models)
### We can get the AIC and BIC of each of these models by re-fitting
### the models and running extractAIC(). The extractAIC() function
### has an input called k, which is the coefficient on the penalty term.
```

```
> all.subsets.models
   (Intercept) Solar.R  Wind  Temp  TWcp TWrat Solar.R:Wind Solar.R:Temp Solar.R:TWcp Solar.R:TWrat Wind:Temp Wind:TWcp Wind:TWrat Temp:TWcp
1        FALSE   FALSE FALSE FALSE FALSE FALSE        FALSE        FALSE        FALSE          TRUE     FALSE     FALSE      FALSE     FALSE
2        FALSE   FALSE FALSE FALSE FALSE FALSE        FALSE        FALSE        FALSE          TRUE     FALSE     FALSE      FALSE     FALSE
3        FALSE   FALSE FALSE FALSE FALSE FALSE         TRUE        FALSE        FALSE          TRUE     FALSE     FALSE      FALSE     FALSE
4        FALSE   FALSE FALSE FALSE FALSE  TRUE        FALSE        FALSE        FALSE          TRUE     FALSE     FALSE      FALSE     FALSE
5        FALSE   FALSE FALSE  TRUE FALSE  TRUE        FALSE        FALSE        FALSE          TRUE     FALSE     FALSE      FALSE     FALSE
6        FALSE   FALSE  TRUE  TRUE FALSE  TRUE        FALSE        FALSE        FALSE          TRUE     FALSE     FALSE      FALSE     FALSE
7        FALSE   FALSE FALSE  TRUE FALSE  TRUE         TRUE         TRUE        FALSE          TRUE     FALSE     FALSE      FALSE     FALSE
8        FALSE    TRUE FALSE FALSE FALSE  TRUE         TRUE        FALSE         TRUE          TRUE     FALSE     FALSE       TRUE     FALSE
9        FALSE   FALSE FALSE FALSE FALSE  TRUE         TRUE         TRUE        FALSE          TRUE      TRUE     FALSE       TRUE      TRUE
10       FALSE   FALSE FALSE  TRUE  TRUE  TRUE         TRUE         TRUE        FALSE          TRUE     FALSE      TRUE      FALSE      TRUE
11       FALSE    TRUE FALSE  TRUE  TRUE  TRUE        FALSE         TRUE         TRUE          TRUE     FALSE      TRUE      FALSE      TRUE
12       FALSE    TRUE  TRUE  TRUE  TRUE  TRUE        FALSE         TRUE         TRUE          TRUE     FALSE      TRUE      FALSE      TRUE
13       FALSE    TRUE  TRUE  TRUE  TRUE  TRUE         TRUE         TRUE         TRUE          TRUE     FALSE      TRUE      FALSE      TRUE
   Temp:TWrat TWcp:TWrat
1       FALSE      FALSE
2       FALSE       TRUE
3       FALSE       TRUE
4        TRUE       TRUE
5        TRUE       TRUE
6        TRUE       TRUE
7        TRUE       TRUE
8        TRUE       TRUE
9        TRUE       TRUE
10       TRUE       TRUE
11       TRUE       TRUE
12       TRUE       TRUE
13       TRUE       TRUE
```

(b) Compute BIC on each of these models and **report the BIC values for the models.**

```
n.models = nrow(all.subsets.models) # Number of candidate models
all.AICs = rep(0, times = n.models) # Container to store AICs
all.BICs = all.AICs # Copy all.AICs to get a container for BICs

for(i in 1:n.models){
  ### We can actually supply a model matrix and response vector
  ### to lm, without using a data frame. Remember that our model matrix
  ### already has an intercept, so we need to make sure lm doesn't
  ### include another one. We do this by including -1 in the right side
  ### of the model formula.
  this.data.matrix = data.matrix[,all.subsets.models[i,]]
  fit = lm(Y.train ~ this.data.matrix - 1)

  ### Get the AIC using extractAIC(). This function takes a regression
  ### model as input, as well as (optionally) an input called k, which
  ### specifies the penalty on the number of variables in our model.
  ### The AIC value is in the second component of the output object.
  this.AIC = extractAIC(fit)[2]
  all.AICs[i] = this.AIC

  ### Get the BIC using extractAIC(). This time, we need to set k equal
  ### to the log of the number of observations used to fit our model
  this.BIC = extractAIC(fit, k = log(n.train))[2]
  all.BICs[i] = this.BIC
}
```

```
> all.BICs
[1] 527.0872 514.9455 513.8233 522.6431 501.1093 497.0320 501.0831 500.8298 493.0466 496.3106 498.2233 489.8231 494.1423
```

(c) Identify the best model. **What variables are in it?**

```
> which.min(all.BICs)
[1] 12
```

12th model includes "Solar.R", "Wind", "Temp", "TWcp", "TWrat", "Solar.R:Temp", "Solar.R:TWcp" "Solar.R:TWrat", "Wind:TWcp", "Temp:TWcp", "Temp:TWrat", and "TWcp:TWrat"

2. Use the hybrid stepwise algorithm that is the default in the step() function. **Report the model that it chooses as "best."**

```
#(b) Compute BIC on each of these models and report the BIC values for the
#models.

#####Use stepwise
fit.start = lm(Ozone ~ 1, data = data.train)
fit.end = lm(Ozone ~ .^2, data = data.train)

step.AIC = step(fit.start, list(upper = fit.end), k = 2)
step.BIC = step(fit.start, list(upper = fit.end), k = log(n.train), trace = 0)

pred.step.AIC = predict(step.AIC, data.valid)
pred.step.BIC = predict(step.BIC, data.valid)

err.step.AIC = get.MSPE(Y.valid, pred.step.AIC)
err.step.BIC = get.MSPE(Y.valid, pred.step.BIC)
```

```
> summary(step.BIC)

Call:
lm(formula = Ozone ~ TWrat + Temp + Solar.R + TWrat:Solar.R +
    TWrat:Temp, data = data.train)

Residuals:
    Min      1Q  Median      3Q     Max
-37.563 -12.930  -3.048  10.033  46.888

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)   -1.584e+02  3.400e+01  -4.658 1.31e-05 ***
TWrat          1.068e+01  4.508e+00   2.370 0.020312 *
Temp           2.555e+00  4.351e-01   5.872 1.03e-07 ***
Solar.R       -1.705e-01  6.466e-02  -2.636 0.010129 *
TWrat:Solar.R  3.060e-02  7.479e-03   4.091 0.000105 ***
TWrat:Temp    -1.613e-01  4.965e-02  -3.248 0.001724 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.88 on 77 degrees of freedom
Multiple R-squared:  0.7313,	Adjusted R-squared:  0.7138
F-statistic: 41.91 on 5 and 77 DF,  p-value: < 2.2e-16
```

The model that use TWrat, Temp, Solar.R, TWrat:Solar.R, and Twrat:Temp as variables

3. Use 10-fold CV to estimate the MSPE for the stepwise model selection process. That is,

```
#(c) Identify the best model. What variables are in it?

set.seed(2928893)
### First we need to set the number of folds
K = 10

### Construct folds
### Don't attach fold labels to dataset because we would just have
### to remove this later
n = nrow(data)
n.fold = n/K # Approximate number of observations per fold
n.fold = ceiling(n.fold)
ordered.ids = rep(1:10, each = n.fold)
ordered.ids = ordered.ids[1:n]
fold.ids = shuffle(ordered.ids)


### Create a container to store CV MSPEs
### One column per model, and one row per fold
CV.models = c("stepwise.AIC", "stepwise.BIC")
errs.CV = array(0, dim = c(K,length(CV.models)))
colnames(errs.CV) = CV.models

for(i in 1:K){
  print(paste0(i, " of ", K))

  ### Construct training and validation sets by either removing
  ### or extracting the current fold.
  ### Also, get the response vectors
  data.train = data[fold.ids != i,]
  data.valid = data[fold.ids == i,]
  Y.train = data.train$Ozone
  Y.valid = data.train$Ozone


  ###########################################
  ### Stepwise selection via AIC and BIC ###
  ###########################################

  fit.start = lm(Ozone ~ 1, data = data.train)
  fit.end = lm(Ozone ~ .^2, data = data.train)

  ### These functions will run several times each. We don't need
  ### to print out all the details, so set trace = 0.
  step.AIC = step(fit.start, list(upper = fit.end), k=2,
                  trace = 0)
  step.BIC = step(fit.start, list(upper = fit.end), k = log(n.train),
                  trace = 0)
  print(summary(step.BIC))

  pred.step.AIC = predict(step.AIC, data.valid)
  pred.step.BIC = predict(step.BIC, data.valid)
```

```r
  err.step.AIC = get.MSPE(Y.valid, pred.step.AIC)
  err.step.BIC = get.MSPE(Y.valid, pred.step.BIC)

  ### Store errors in errs.CV, which has two dimensions, so
  ### we need two indices
  errs.CV[i, "stepwise.AIC"] = err.step.AIC
  errs.CV[i, "stepwise.BIC"] = err.step.BIC
}
```

```
> errs.CV
      stepwise.AIC stepwise.BIC
 [1,]     1445.724     1379.311
 [2,]     2929.978     3469.082
 [3,]     1787.337     1720.089
 [4,]     1320.081     1320.081
 [5,]     2407.516     2407.516
 [6,]     2302.269     2263.954
 [7,]     2003.969     2066.328
 [8,]     1529.626     1544.587
 [9,]     2007.500     2007.500
[10,]     2224.739     2563.569
```

4th model shows the smallest BIC with 1320.081.