a3

q2

```
data = read.csv("energyapp.csv")

get.reduced.model = function(model, i){
  # convenient helper to return the new model with ith feature removed
  # i can be vector or number

  # first column of data will be response variable, other columns are features of original
  # model, intercept wouldn't appear here as a feature
  data = model$model
  r = nrow(data)
  c = ncol(data)

  # special case if there is only 1 feature left
  if(c==2){
    return(lm(data[1:r,1]~1))
  }

  # we shouldn't receive a model with only intercept
  if(c==1){
    stop("get.reduced.model() recieved a model with intercept only")
  }

  # explanatory variable
  names = colnames(data)[2:c]
  # response variable
  yname = colnames(data)[1]
  formu = as.formula( paste(yname, "~", paste( names[-i], collapse = "+")))
  # new model
  m =  lm(formu , data=data)
  return(m)
}

backward.step = function(model, method, alpha=0.05){
  data = model$model
  r = nrow(data)
  c = ncol(data)
  if(method == "adjR2"){
    while(TRUE){
      oldR2 = summary(model)$adj.r.squared
      # put c-1 reduced model's adjRsqr in this newR2 vector
      newR2 = rep(0, c-1)
      for (i in 1:(c-1)){
        new.model = get.reduced.model(model, i)
        newR2[i] = summary(new.model)$adj.r.squared
      }
```

```
      # if reduced model doesn't have larger adjRsqr, we are done
      if (oldR2>max(newR2)){
        return(model)
      }else{
        # get rid of the feature, which with removing it, we obtain the largest adj R sqr
        model = get.reduced.model(model, which.max(newR2))
        c=c-1
      }
    }
  }
  else if(method == "AIC"){
    # we just use built in
    mod0 = lm(data[1:r, 1]~ 1)
    model = step(model, scope = list(lower = mod0, upper = model), direction = "backward", trace=0, k =
    return(model)
  }
  else if(method == "BIC"){
    # we just use built in, set k = log(r) where r is number of data
    mod0 = lm(data[1:r, 1]~ 1)
    model = step(model, scope = list(lower = mod0, upper = model), direction = "backward", trace=0, k =
    return(model)
  }else if (method == "pval"){
    while(TRUE){
      m = summary(model)$coefficients
      # start with 2 to get rid of intercept
      i = m[2:nrow(m), "Pr(>|t|)" ]
      # if all pvals are less than alpha, we are done
      if(max(i)<alpha){
        return(model)
      }
      # remove feature with largest pval
      ind = which.max(i)
      model = get.reduced.model(model, ind)
    }
  }else{
    stop("method is not one of specified")
  }
}
```

```
data = read.csv("energyapp.csv")

xtrain = data[1:500,]
xtest = data[501:nrow(data),]
```

(a)

Yes, there would be multicolliearity among the explanatory variables. For example, Temperature in different rooms of a house should have a positive correlation with each other. Same applies to Humidity.

If all features are applied, in the worst case, our design matrix's columns will be linearly dependent, we won't find a linear model. Otherwise we will have a extremely large confidence interval on prediction or on model fitting.

```
library("car")
```

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

```
model = lm(appEuse ~ . ,data=xtrain)


while (TRUE) {
  score = vif(model)
  if (max(score) <10){
    break
  }
  ind = which.max(score)
  model = get.reduced.model(model, ind)
}

vif(model)
```

```
##      lights        RH_1          T3        RH_3          T4          T5
##    1.883082    3.945864    2.216883    3.026102    2.800607    2.963431
##        RH_5        RH_6          T7          T8        RH_8          T9
##    2.264592    5.512718    4.694970    5.712952    3.241360    2.167350
## Press_mm_hg      RH_out   Windspeed  Visibility
##    4.347010    5.710715    2.064364    1.705817
```

```
length(model$coefficients)-1
```

```
## [1] 16
```

16 coefficients are left (excluding intercept)

this is preferable because if among explanatory variables a,b,c,d,e,f,

only a,b,c has strong correlation, we should leave only one of "a/b/c" left in our model (our model would have feature, for example, a,d,e,f, instead of just d,e,f.

however, a,b,c will all have large vif. Therefor we cannot delete large vif simultaneously.

```
after = function(model){
  return(list(adj.r.sqr = summary(model)$adj.r.squared,
              AIC=AIC(model), BIC=BIC(model)))
}

result = list()
```

```
model = lm(appEuse ~ . ,data=xtrain)


m1=backward.step(model, "adjR2")
result[[1]] = after(m1)
m2=backward.step(model, "AIC")
result[[2]] = after(m2)
m3=backward.step(model, "BIC")
result[[3]] = after(m3)
m4=backward.step(model, "pval")
result[[4]] = after(m4)


m=matrix(0, nrow = 4, ncol = 3)

for (i in 1:4){
    m[i,1] = result[[i]]$adj.r.sqr
    m[i,2] = result[[i]]$AIC
    m[i,3] = result[[i]]$BIC
}
library(knitr)
colnames(m)=c("adjr.sqaure", "AIC", "BIC")
rownames(m)=c("adjr.sqaure", "AIC", "BIC", "pval")

kable(m)
```

	adjr.sqaure	AIC	BIC
adjr.sqaure	0.3180099	6113.886	6206.608
AIC	0.3160125	6112.470	6192.548
BIC	0.3082916	6116.153	6187.802
pval	0.3134025	6113.411	6189.274

we observe the model that select feature based on certain criteria will result in a model that do well on that feature.

Selecting by p value result in 2nd place in all criteria, which is impressive

there is not significantly large difference among these results though. there is no method that ends up with 2 1st places or have 2 last places among the 3 criteria.

```
y = xtest[,"appEuse"]

y1=predict(m1,newdata=xtest)
mse1 = mean((y1-y)^2)

y2=predict(m2,newdata=xtest)
mse2 = mean((y2-y)^2)

y3=predict(m3,newdata=xtest)
mse3 = mean((y3-y)^2)

y4=predict(m4,newdata=xtest)
mse4 = mean((y4-y)^2)
```

```
#adj.r.sqr
mse1
```

```
## [1] 48556.36
```

```
#AIC
mse2
```

```
## [1] 44641.56
```

```
#BIC
mse3
```

```
## [1] 55663.33
```

```
#pval
mse4
```

```
## [1] 55055.78
```

we observe fitting well on the train data does not mean good performance on test set.

In this case, AIC result in better performance. p-value performs bad as expected, as p-value doesn't consider the effect of overfitting.

quite surprisingly, BIC has a quite large MSE, this might be due to chance.

```r
set.seed(20704870)
n=100
v=50
ran = rnorm(n*v)
fakedata = matrix(ran, nrow = n, ncol = v)
colnames(fakedata) <- paste0("Grbg_", 1:v)
fakedata=data.frame(fakedata)

y = -1+ rnorm(n,0,2)
data = cbind(fakedata, y)

model = lm(y~Grbg_1+Grbg_2+Grbg_3+Grbg_4+Grbg_5 , data=data )

summary(model)
```

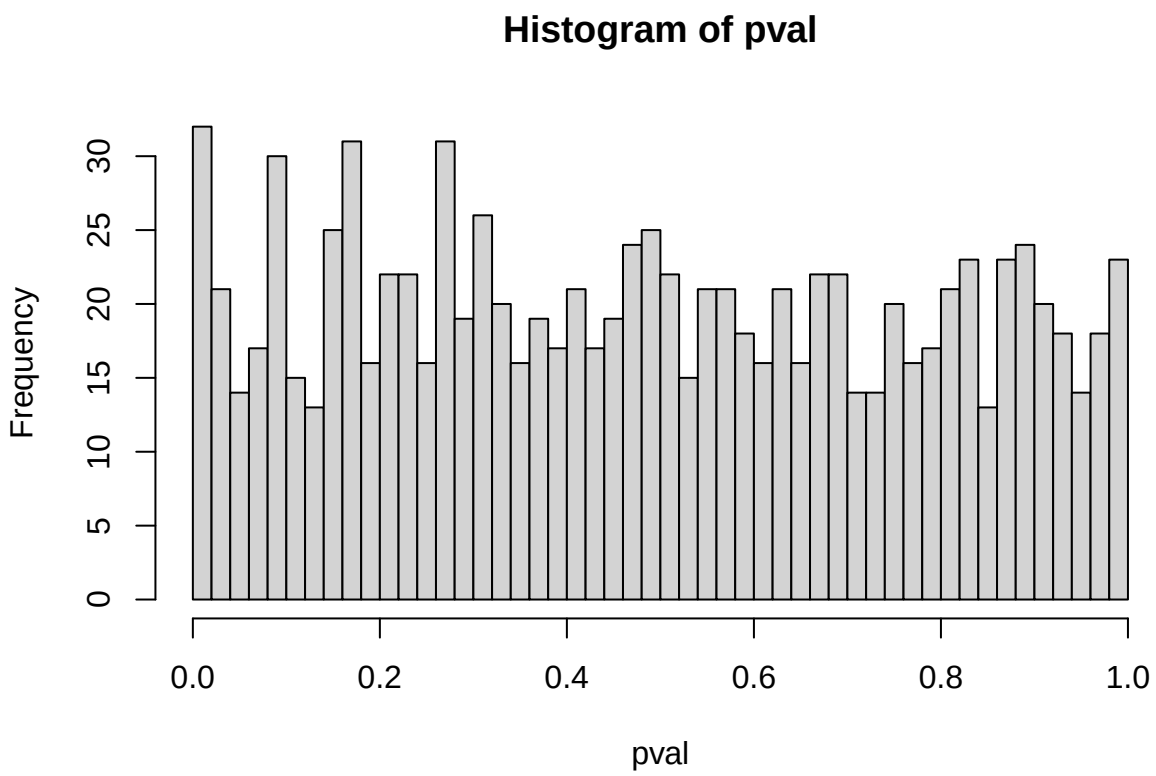
```
##
## Call:
## lm(formula = y ~ Grbg_1 + Grbg_2 + Grbg_3 + Grbg_4 + Grbg_5,
##     data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.3779 -1.3670 -0.1522  1.4521  5.6469
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.26832    0.20125  -6.302 9.42e-09 ***
## Grbg_1      -0.41377    0.19068  -2.170   0.0325 *
## Grbg_2       0.01706    0.18888   0.090   0.9282
## Grbg_3      -0.06458    0.19988  -0.323   0.7473
## Grbg_4       0.18659    0.21129   0.883   0.3794
## Grbg_5       0.02465    0.20547   0.120   0.9048
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.997 on 94 degrees of freedom
## Multiple R-squared:  0.05162,    Adjusted R-squared:  0.001177
## F-statistic: 1.023 on 5 and 94 DF,  p-value: 0.4085
```

```r
#v
M=1000
data = fakedata[,1:v]
pval = rep(0, M)
for (i in 1:M){
  y = -1+ rnorm(n,0,2)
  newdata = cbind(data, y)

  colnames(newdata) <- c(paste0("Grbg_", 1:v),"y")
  newdata = data.frame(newdata)
  model = lm(y~Grbg_1+Grbg_2+Grbg_3+Grbg_4+Grbg_5 , data=newdata)
  s=summary(model)
  f=s$fstatistic
  pval[i] = 1-pf(f[1],f[2],f[3])
```

```
}

hist(pval, breaks = 50)
```




```
# prop that got rejected
length(pval[pval<0.05])/M
```

```
## [1] 0.06
```

iv) $H_0 : Grbg_i = 0, i \in [1, 5]$: has pval 0.4085, we do not reject H_0

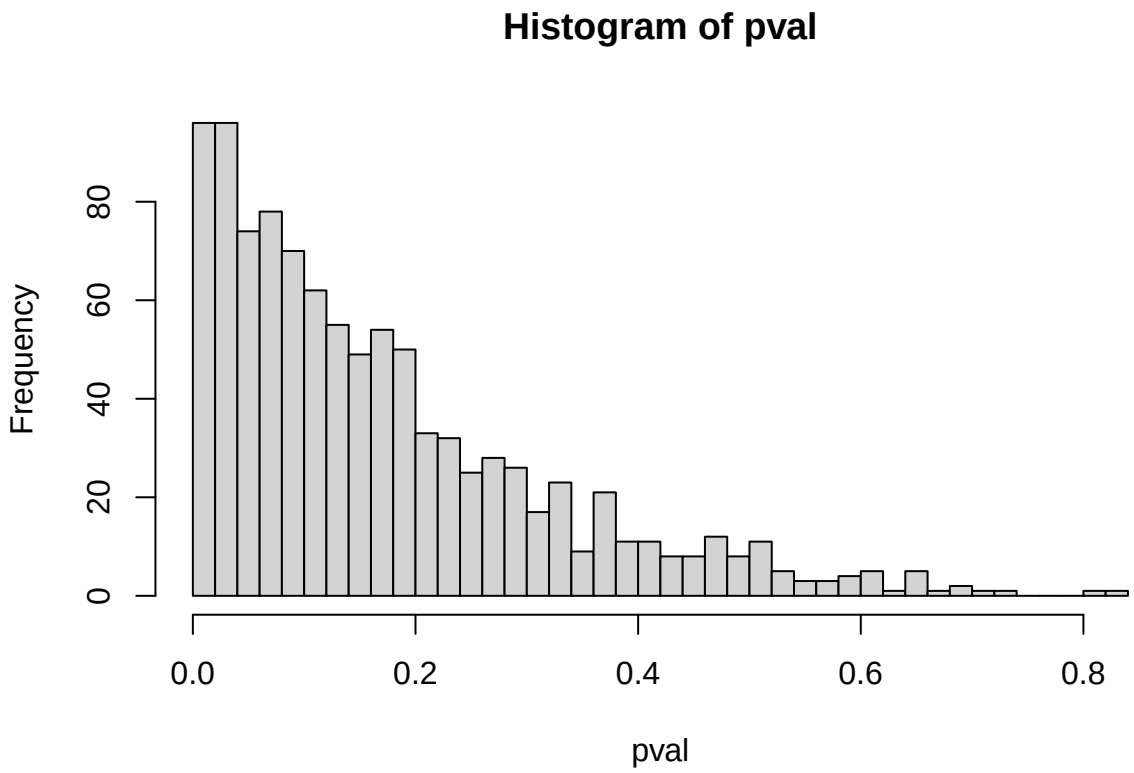
```
#B
M=1000
data = fakedata

pval = rep(0,M)
adj.r.sqr = rep(0,5)
temppval = rep(0,5)
for (i in 1:M){
  y = -1+ rnorm(n,0,2)
  newdata = cbind(data, y)


  for (j in 1:5){
    formu = as.formula( paste( "y~",paste0("Grbg_", (j*5-4):(j*5), collapse = "+" )))
    model = lm(formu , data=newdata)
    s=summary(model)
```

```
    adj.r.sqr[j] = s$adj.r.squared
    f=s$fstatistic
    temppval[j] = 1-pf(f[1],f[2],f[3])
  }
  ind = which.max(adj.r.sqr)
  pval[i] = temppval[ind]
}

hist(pval, breaks = 50)
```




```
# prop that got rejected
length(pval[pval<0.05])/M
```

```
## [1] 0.23
```

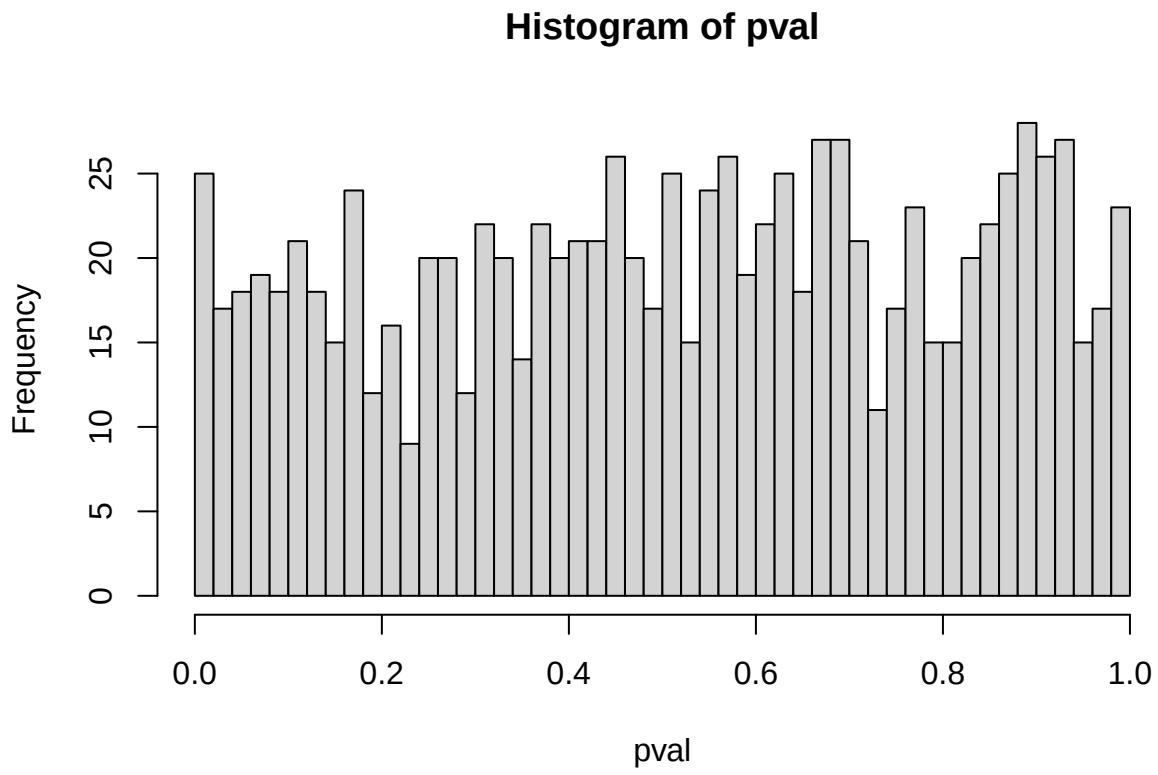
C) we find that if we let our data choose the model with lowest pvalue, we end up with lower pvalue than the true possibility that we observe the data given H_0 is true.

Therefor p-value makes less sense when we are choosing our model depending on our data.

```
#D
M=1000
data = fakedata

pval = rep(0,M)
for (i in 1:M){
  y = -1+ rnorm(n,0,2)
  data = cbind(data, y)
```

```
  model = lm(y~. , data=data)
  mod0 = lm(y~1)
  model = step(model, scope = list(lower = mod0, upper = model), direction = "forward", trace=0, k = 2)
  s=summary(model)
  f=s$fstatistic
  pval[i] = 1-pf(f[1],f[2],f[3])
  data = data[,1:v]
}
hist(pval, breaks = 50)
```




```
# prop that got rejected
length(pval[pval<0.05])/M
```

```
## [1] 0.051
```

we are glad to find forward selection does make the concept of p-value consistent even when the model selected depends on the data.

This is because forward selection use AIC, which consider effect of overestimating.