2. Now we must figure out which number of terms to use in a final prediction. We need to tune this parameter. Use 10-fold cross-validation to train models and compute MSPE for values of nterms from among 1, 2, 3, 4, and 5, maintaining max.terms=5. Be sure to train each version of the model on each fold so that the comparison across the tuning parameters is easy.

(a) **Report the matrix of MSPEs from CV.** (There should be 10 rows and 5 columns)

```
### Split data
n = nrow(data)
p.train = 0.75
n.train = round(n * p.train)
n.valid = n - n.train
sets = c(rep(1, times = n.train), rep(2, times = n.valid))
sets.rand = shuffle(sets) # Our helper function

data.train = data[sets.rand == 1,]
data.valid = data[sets.rand == 2,]
Y.valid = data.valid$Ozone

############
### PPR ###
############

### To fit PPR, we need to do another round of CV. This time, do 5-fold
K.ppr = 10
n.train = nrow(data.train)
folds.ppr = get.folds(n.train, K.ppr)

### Container to store MSPEs for each number of terms on each sub-fold
MSPEs.ppr = array(0, dim = c(K.ppr, max.terms))

for(j in 1:K.ppr){
  ### Split the training data.
  ### Be careful! We are constructing an internal validation set by
  ### splitting the training set from outer CV.
  train.ppr = data.train[folds.ppr != j,]
  valid.ppr = data.train[folds.ppr == j,]
  Y.valid.ppr = valid.ppr$Ozone

  ### We need to fit several different PPR models, one for each number
  ### of terms. This means another for loop (make sure you use a different
  ### index variable for each loop).
  for(l in 1:max.terms){
    ### Fit model
    fit.ppr = ppr(Ozone ~ ., data = train.ppr,
                  max.terms = max.terms, nterms = l, sm.method = "gcvspline")

    ### Get predictions and MSPE
    pred.ppr = predict(fit.ppr, valid.ppr)
    MSPE.ppr = get.MSPE(Y.valid.ppr, pred.ppr) # Our helper function
```

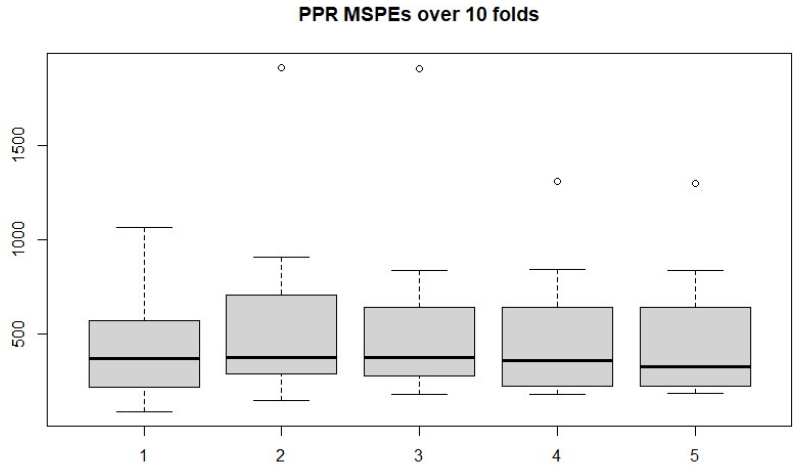
```
94
95      ### Store MSPE. Make sure the indices match for MSPEs.ppr
96      MSPEs.ppr[j, l] = MSPE.ppr
97    }
98  }
99
100 MSPEs.ppr
101
102 boxplot(MSPEs.ppr, main = paste0("PPR MSPEs over ", K, " folds"))
103
104 ### Calculate RMSPEs
105 all.RMSPEs = apply(MSPEs.ppr, 1, function(W){
106   best = min(W)
107   return(W / best)
108 })
109 all.RMSPEs = t(all.RMSPEs)
110
111 ### Make a boxplot of RMSPEs
112 boxplot(all.RMSPEs, main = paste0("CV RMSPEs over ", K, " folds"))
```

```
> MSPEs.ppr
           [,1]      [,2]      [,3]      [,4]      [,5]
 [1,]  623.89131  643.5699  638.2097  638.2097  638.2097
 [2,]  162.91871  359.1421  343.8283  362.3022  298.6712
 [3,]  218.78516  222.4944  222.4895  222.4895  222.4895
 [4,]  408.20916  285.3450  275.2477  262.0491  261.8709
 [5,]  545.62349  905.5683  836.2775  839.1623  835.9885
 [6,]   84.95486  147.8934  179.0570  175.6638  185.1357
 [7,]  569.83354  703.7716  638.0148  550.1438  504.3436
 [8,]  223.67817  371.6668  386.1637  197.0021  210.8126
 [9,] 1063.69841 1915.3978 1908.7334 1312.0695 1298.9598
[10,]  331.16617  380.0875  362.4160  347.5703  347.5084
```

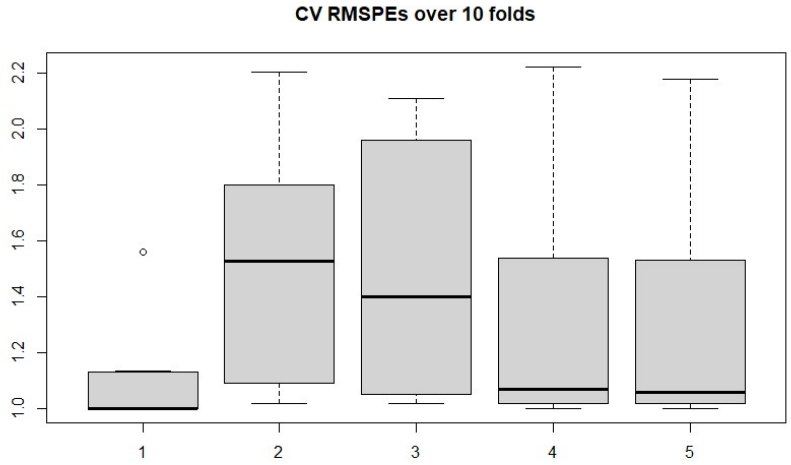
i. Comment on any consistent patterns you see in the comparison among numbers of terms. Specifically, are there one or more values that seem much better than others?

➔ Generally it looks like when nterms is 3, it has the smallest MSPE

(b) **Create and show the side-by-side boxplots of these 10 MSPEs for each number of terms (5 boxes)**



(c) **Repeat using relative MSPE**



(d) Based on what you have seen, **how many terms would you use?** If there is no clear winner, then choose the least complicated model than is among the top models.

I would use 1, because it's clearly better than other number of terms.