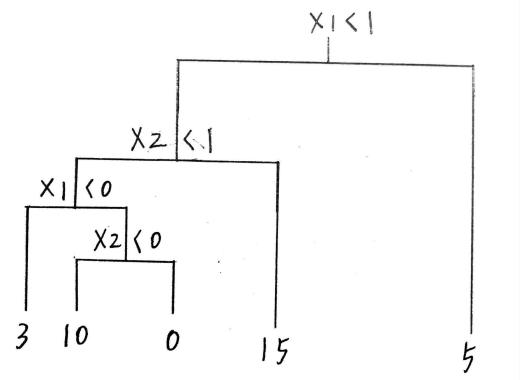
統計學習 作業六

106070038 科管院學士班 杜葳葳

4.

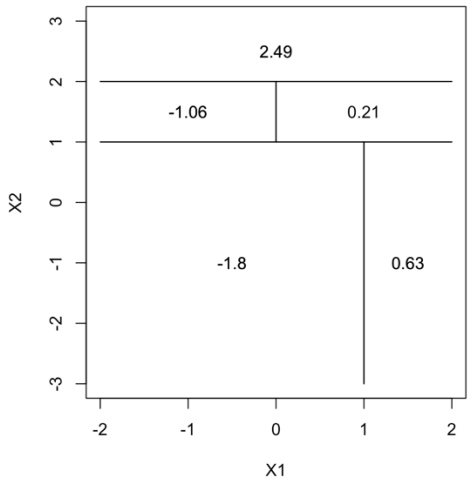
(a)

if x1 ≥ 1 then 5,
else if x2 ≥ 1 then 15,
else if x1 < 0 then 3,
else if x2 < 0 then 10, else 0



(b)

if x2 < 1, {
  if x1 < 1 then -1.8,
  else 0.63
} else {
  if x2 > 2 then 2.49,
  else if x1 < 0 then -1.06,
  else 0.21
}



8.

(a) Carseats 資料集：A simulated data set containing sales of child car seats at 400 different stores.

變數：

- Sales（銷量）
- CompPrice（競爭者價格）
- Income（收入等級）
- Advertising（廣告預算）
- Population（人口） (thousands)
- Price（價格）
- ShelveLoc（貨架陳列位置）
- Age（年齡）
- Education（教育程度）
- Urban（市區/鄉村）
- US（在美國/不在美國）

```
'data.frame':   400 obs. of  11 variables:
 $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
 $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
 $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
 $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
 $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
 $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
 $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
 $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
 $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
 $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
 $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

```
  Sales CompPrice Income Advertising Population Price ShelveLoc Age Education Urban  US
1  9.50       138     73          11        276   120       Bad  42        17   Yes Yes
2 11.22       111     48          16        260    83      Good  65        10   Yes Yes
3 10.06       113     35          10        269    80    Medium  59        12   Yes Yes
4  7.40       117    100           4        466    97    Medium  55        14   Yes Yes
5  4.15       141     64           3        340   128       Bad  38        13   Yes  No
6 10.81       124    113          13        501    72       Bad  78        16    No Yes
```

計算各變數的最大、最小值、四分位數、中位數、平均

```
     Sales          CompPrice       Income        Advertising       Population   
 Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000   Min.   : 10.0  
 1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000   1st Qu.:139.0  
 Median : 7.490   Median :125   Median : 69.00   Median : 5.000   Median :272.0  
 Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635   Mean   :264.8  
 3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000   3rd Qu.:398.5  
 Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000   Max.   :509.0  
     Price        ShelveLoc        Age          Education    Urban       US     
 Min.   : 24.0   Bad   : 96   Min.   :25.00   Min.   :10.0   No :118   No :142  
 1st Qu.:100.0   Good  : 85   1st Qu.:39.75   1st Qu.:12.0   Yes:282   Yes:258  
 Median :117.0   Medium:219   Median :54.50   Median :14.0                      
 Mean   :115.8                Mean   :53.32   Mean   :13.9                      
 3rd Qu.:131.0                3rd Qu.:66.00   3rd Qu.:16.0                      
 Max.   :191.0                Max.   :80.00   Max.   :18.0
```

除了 Urban 和 US 是類別型變數外，其他都是連續型的變數。

以 **4 : 1** 的比例切成訓練集和測試集

訓練集

```
'data.frame':   320 obs. of  11 variables:
 $ Sales      : num  7.52 5.25 9.95 4.97 4.1 ...
 $ CompPrice  : num  123 131 132 162 121 141 127 145 124 89 ...
 $ Income     : num  39 55 33 67 78 60 44 69 44 81 ...
 $ Advertising: num  5 0 7 0 4 19 13 19 0 15 ...
 $ Population : num  499 26 35 27 413 319 160 501 125 237 ...
 $ Price      : num  98 110 97 160 130 92 123 105 107 99 ...
 $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 3 1 3 3 1 2 2 3 3 2 ...
 $ Age        : num  34 79 60 77 46 44 63 45 80 74 ...
 $ Education  : num  15 12 11 17 10 11 18 11 11 12 ...
 $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 1 2 1 2 2 2 2 2 ...
 $ US         : Factor w/ 2 levels "No","Yes": 1 2 2 2 2 2 2 2 1 2 ...
```

測試集

```
'data.frame':   80 obs. of  11 variables:
 $ Sales      : num  10.81 9.01 10.96 13.91 8.73 ...
 $ CompPrice  : num  124 121 115 110 129 121 103 136 131 157 ...
 $ Income     : num  113 78 28 110 76 31 74 58 84 53 ...
 $ Advertising: num  13 9 11 0 16 0 0 16 11 0 ...
 $ Population : num  501 150 29 408 58 292 359 241 29 403 ...
 $ Price      : num  72 100 86 68 121 109 97 131 96 124 ...
 $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 1 2 2 3 3 1 3 3 1 ...
 $ Age        : num  78 26 53 46 69 79 55 44 44 58 ...
 $ Education  : num  16 10 18 17 12 10 11 18 17 16 ...
 $ Urban      : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 2 2 2 1 2 ...
 $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 2 1 ...
```

(b)

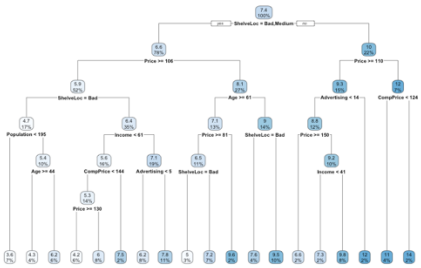
- 控制 cp：alpha 越大，選到的子樹越小

cp 控制 alpha 值，當 alpha 越大，選到的子樹越小，樹的層數越少、節點數也越少

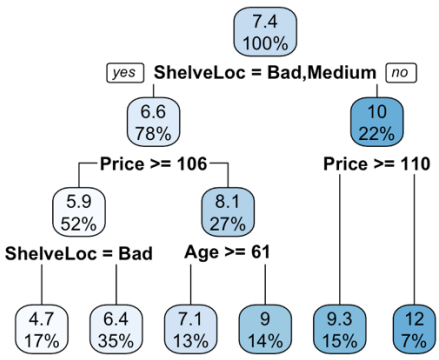
用 training set 訓練，並實驗 testing set 在不同 cp 值之下的 MSE，可觀察到當 cp 大於 0.05 時，MSE 皆固定在 5.28738。而當 cp 在 0.03 附近時，有最佳的 MSE。

cp	0.01	0.03	0.05
MSE	4.405849	3.959836	5.28738

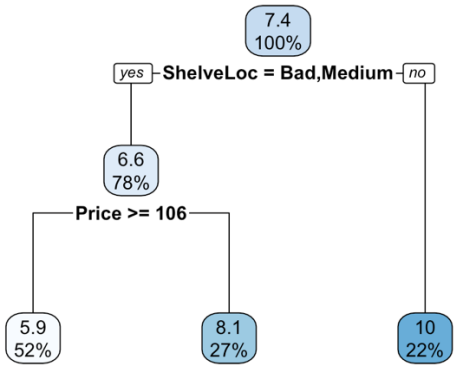
cp = 0.01



cp = 0.03



cp = 0.05



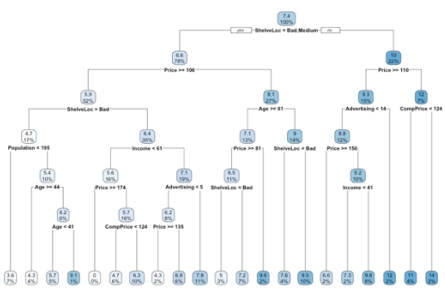
- 控制 minbucket：控制在末端的 node 上最少要幾個 data

當 minbucket 增加時，因為末端的節點數變多，樹的結構變得相對簡單，故層數也減少。

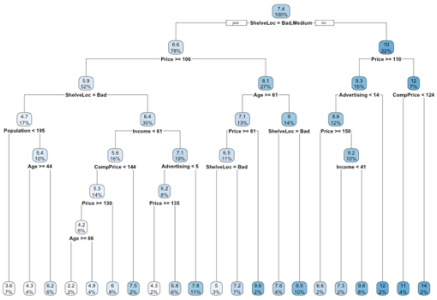
用 training set 訓練，並實驗 testing set 在不同 minbucket 之下的 MSE，可觀察到當 minbucket 在 15 附近時，有最佳的 MSE。

minbucket	1	5	10	15	20
MSE	4.123522	4.284842	3.973578	3.888589	4.13882

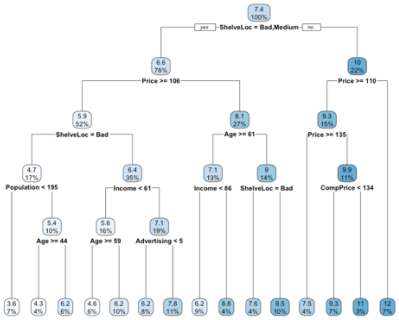
minbucket = 1



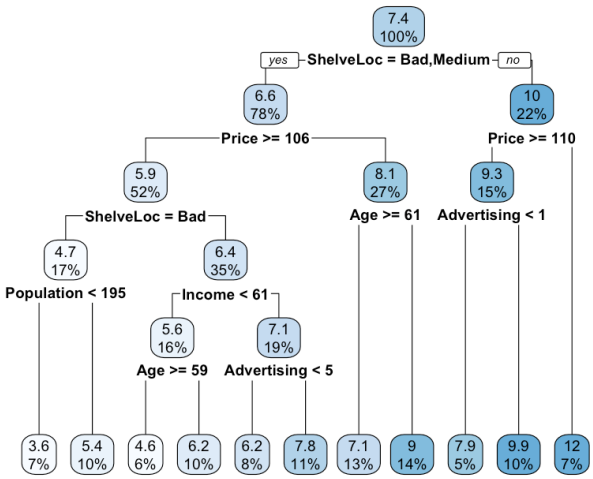
minbucket = 5



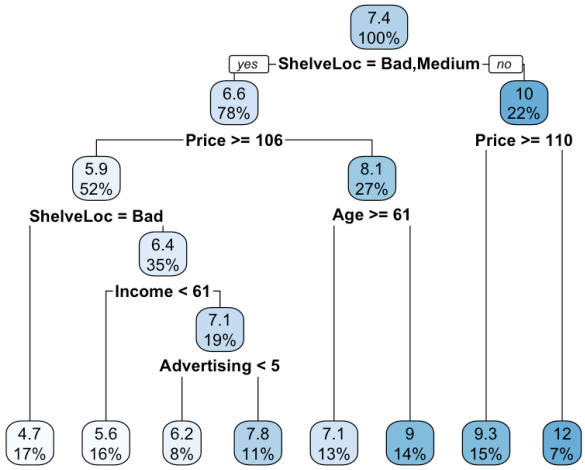
minbucket = 10



minbucket = 15



minbucket = 20

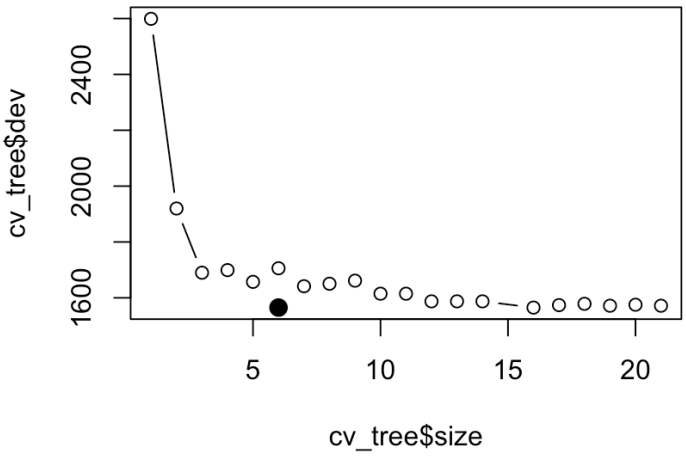


(c)

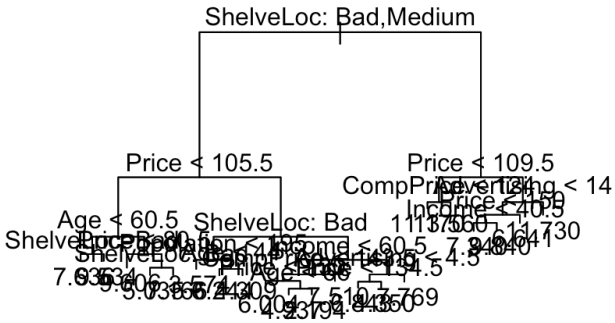
用 training set 做訓練，並利用未修剪的 tree 對 testing set 做預測，得到 MSE = 4.284842，

接著利用 Cross Validation 找最佳樹的大小，#Fold = 5 時，有最佳大小的樹（如下圖所示，圓形實心點為最佳樹的大小= 6）

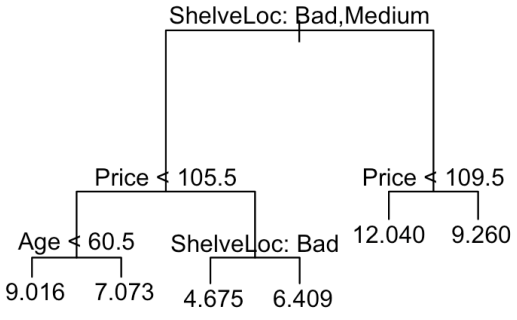
於是將樹修剪(prune)為只剩 6 個終端節點，得到 MSE = 3.959836，比尚未修剪的好。



尚未修剪的 tree：



修剪後的 tree：



綜合以上可知，修剪過的樹的 MSE < 尚未修剪的樹的 MSE 小，因此修剪過比較好。

(d)

bagging 是將樣本重複抽樣(取後放回)，產生多個子資料集後，依序建立多個模型，最後再將所有模型的結果彙整在一起，有助於降低 variance。

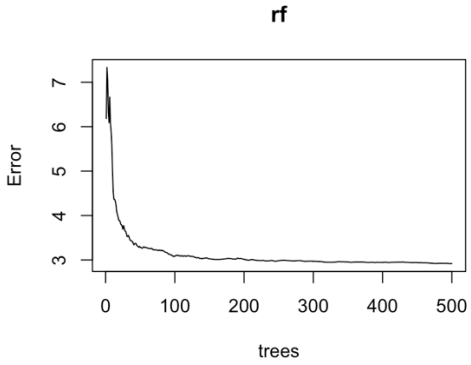
使用 bagging、抽 100 次 boostrap sample 建模，取 100 次平均計算 MSE = 2.416937

下面是各個變數的重要性，可發現 ShelveLoc 和 Price 是兩個最重要的變數

```
ShelveLoc       Price Advertising   CompPrice  Population
699.10139   563.53086   371.50737   292.45918   259.59054
Education         Age      Income          US
157.92502   135.10996    81.68312    52.91628
```

(e) Random Forest

同樣使用 training set 建樹，用 testing set 做預測、MSE = 2.516136，MSE 比(c)小題下降、模型較好

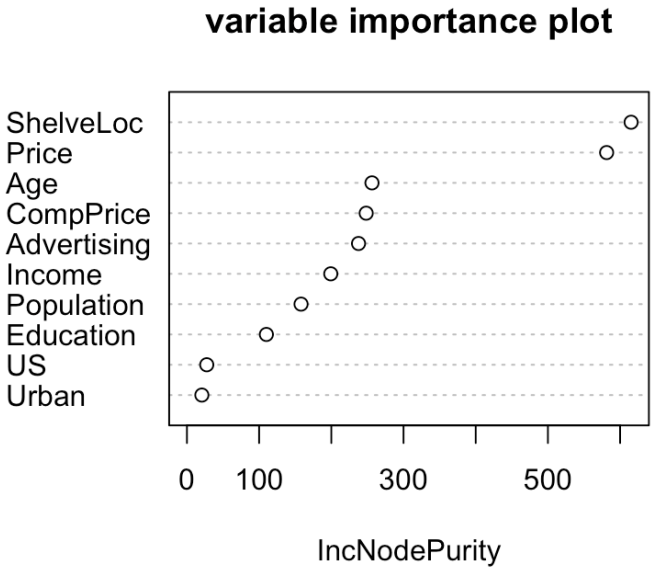



```
Call:
 randomForest(formula = Sales ~ ., data = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 2.919329
                    % Var explained: 63.92
```

用 importance 看各個變數的重要性，可發現 ShelveLoc 和 Price 是兩個最重要的變數，
US 和 Urban 則為最不重要的兩個變數。

```
             IncNodePurity
CompPrice        248.28101
Income           199.24256
Advertising      237.68791
Population       158.06084
Price            581.33142
ShelveLoc        615.24361
Age              256.31313
Education        110.05427
Urban             20.67165
US                27.45737
```



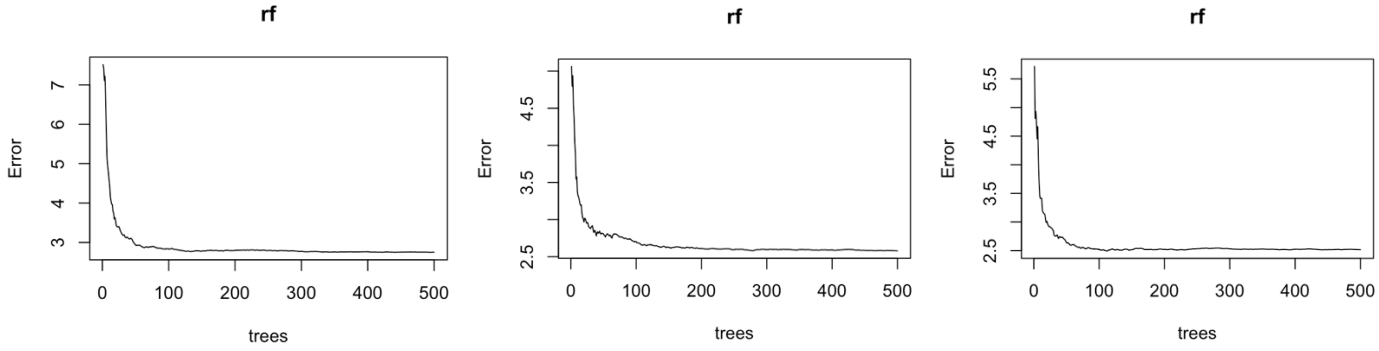
- 控制 mtry

當 mtry 增加時，每次切割考慮的變數個數變多。

用 training set 訓練，並實驗 testing set 在不同 mtry 之下的 MSE，可觀察到當 mtry 在 10 附近時，有最佳的 MSE。

mtry	4	6	10
MSE	2.314774	2.156937	2.04352

mtry = 4 mtry = 6 mtry = 10



9.

(a)

OJ Dataset：1070 筆消費者購買 CH 或 MM 柳丁汁的資料

每筆資料包含 18 個變數，如下：

```
'data.frame':   1070 obs. of  18 variables:
 $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 1 1 1 1 1 ...
 $ WeekofPurchase: num  237 239 245 227 228 230 232 234 235 238 ...
 $ StoreID       : num  1 1 1 1 7 7 7 7 7 7 ...
 $ PriceCH       : num  1.75 1.75 1.86 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
 $ PriceMM       : num  1.99 1.99 2.09 1.69 1.69 1.99 1.99 1.99 1.99 1.99 ...
 $ DiscCH        : num  0 0 0.17 0 0 0 0 0 0 0 ...
 $ DiscMM        : num  0 0.3 0 0 0 0 0.4 0.4 0.4 0.4 ...
 $ SpecialCH     : num  0 0 0 0 0 0 1 1 0 0 ...
 $ SpecialMM     : num  0 1 0 0 0 1 1 0 0 0 ...
 $ LoyalCH       : num  0.5 0.6 0.68 0.4 0.957 ...
 $ SalePriceMM   : num  1.99 1.69 2.09 1.69 1.69 1.99 1.59 1.59 1.59 1.59 ...
 $ SalePriceCH   : num  1.75 1.75 1.69 1.69 1.69 1.69 1.69 1.75 1.75 1.75 ...
 $ PriceDiff     : num  0.24 -0.06 0.4 0 0 0.3 -0.1 -0.16 -0.16 -0.16 ...
 $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 1 1 2 2 2 2 2 2 ...
 $ PctDiscMM     : num  0 0.151 0 0 0 ...
 $ PctDiscCH     : num  0 0 0.0914 0 0 ...
 $ ListPriceDiff : num  0.24 0.24 0.23 0 0 0.3 0.3 0.24 0.24 0.24 ...
 $ STORE         : num  1 1 1 1 0 0 0 0 0 0 ...
```

```
Purchase WeekofPurchase     StoreID        PriceCH         PriceMM         DiscCH            DiscMM
CH:653   Min.   :227.0   Min.   :1.00   Min.   :1.690   Min.   :1.690   Min.   :0.00000   Min.   :0.0000
MM:417   1st Qu.:240.0   1st Qu.:2.00   1st Qu.:1.790   1st Qu.:1.990   1st Qu.:0.00000   1st Qu.:0.0000
         Median :257.0   Median :3.00   Median :1.860   Median :2.090   Median :0.00000   Median :0.0000
         Mean   :254.4   Mean   :3.96   Mean   :1.867   Mean   :2.085   Mean   :0.05186   Mean   :0.1234
         3rd Qu.:268.0   3rd Qu.:7.00   3rd Qu.:1.990   3rd Qu.:2.180   3rd Qu.:0.00000   3rd Qu.:0.2300
         Max.   :278.0   Max.   :7.00   Max.   :2.090   Max.   :2.290   Max.   :0.50000   Max.   :0.8000
  SpecialCH        SpecialMM        LoyalCH          SalePriceMM     SalePriceCH      PriceDiff
Min.   :0.0000   Min.   :0.0000   Min.   :0.000011   Min.   :1.190   Min.   :1.390   Min.   :-0.6700
1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.325257   1st Qu.:1.690   1st Qu.:1.750   1st Qu.: 0.0000
Median :0.0000   Median :0.0000   Median :0.600000   Median :2.090   Median :1.860   Median : 0.2300
Mean   :0.1477   Mean   :0.1617   Mean   :0.565782   Mean   :1.962   Mean   :1.816   Mean   : 0.1465
3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.850873   3rd Qu.:2.130   3rd Qu.:1.890   3rd Qu.: 0.3200
Max.   :1.0000   Max.   :1.0000   Max.   :0.999947   Max.   :2.290   Max.   :2.090   Max.   : 0.6400
Store7      PctDiscMM        PctDiscCH       ListPriceDiff       STORE
No :714   Min.   :0.0000   Min.   :0.00000   Min.   :0.000   Min.   :0.000
Yes:356   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.140   1st Qu.:0.000
          Median :0.0000   Median :0.00000   Median :0.240   Median :2.000
          Mean   :0.0593   Mean   :0.02731   Mean   :0.218   Mean   :1.631
          3rd Qu.:0.1127   3rd Qu.:0.00000   3rd Qu.:0.300   3rd Qu.:3.000
          Max.   :0.4020   Max.   :0.25269   Max.   :0.440   Max.   :4.000
```

將資料集切成 training 和 testing，training set 共有 800 筆資料、其餘則為 testing set

訓練集：

```
'data.frame':   800 obs. of  18 variables:
 $ Purchase      : Factor w/ 2 levels "CH","MM": 1 1 1 2 1 2 2 1 1 1 ...
 $ WeekofPurchase: num  236 277 267 236 274 231 261 258 268 258 ...
 $ StoreID       : num  7 1 2 3 7 2 2 7 7 1 ...
 $ PriceCH       : num  1.75 1.99 1.86 1.79 1.86 1.69 1.86 1.86 1.86 1.76 ...
 $ PriceMM       : num  1.99 2.13 2.18 2.09 2.13 1.69 2.18 2.18 2.13 2.18 ...
 $ DiscCH        : num  0 0.24 0 0 0.47 0.3 0 0 0 0 ...
 $ DiscMM        : num  0.4 0 0.4 0 0.54 0 0 0 0 0 ...
 $ SpecialCH     : num  0 0 0 0 1 1 0 0 0 0 ...
 $ SpecialMM     : num  0 0 1 0 0 0 0 0 0 0 ...
 $ LoyalCH       : num  0.909 0.585 0.891 0.164 0.5 ...
 $ SalePriceMM   : num  1.59 2.13 1.78 2.09 1.59 1.69 2.18 2.18 2.13 2.18 ...
 $ SalePriceCH   : num  1.75 1.75 1.86 1.79 1.39 1.39 1.86 1.86 1.86 1.76 ...
 $ PriceDiff     : num  -0.16 0.38 -0.08 0.3 0.2 0.3 0.32 0.32 0.27 0.42 ...
 $ Store7        : Factor w/ 2 levels "No","Yes": 2 1 1 1 2 1 1 2 2 1 ...
 $ PctDiscMM     : num  0.201 0 0.183 0 0.254 ...
 $ PctDiscCH     : num  0 0.121 0 0 0.253 ...
 $ ListPriceDiff : num  0.24 0.14 0.32 0.3 0.27 0 0.32 0.32 0.27 0.42 ...
 $ STORE         : num  0 1 2 3 0 2 2 0 0 1 ...
```

測試集：

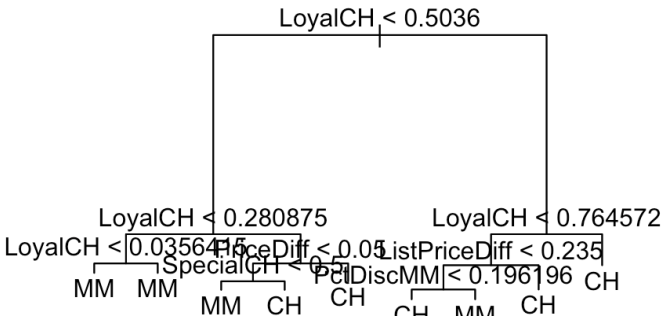
```
'data.frame':   270 obs. of  18 variables:
 $ Purchase      : Factor w/ 2 levels "CH","MM": 1 2 1 2 1 1 1 2 1 1 ...
 $ WeekofPurchase: num  239 227 238 268 259 269 265 232 277 240 ...
 $ StoreID       : num  1 1 7 2 7 7 7 2 1 4 ...
 $ PriceCH       : num  1.75 1.69 1.75 1.86 1.86 1.86 1.86 1.69 1.99 1.79 ...
 $ PriceMM       : num  1.99 1.69 1.99 2.18 2.18 2.13 2.13 1.69 2.13 2.23 ...
 $ DiscCH        : num  0 0 0 0 0 0.27 0.37 0 0.24 0 ...
 $ DiscMM        : num  0.3 0 0.4 0 0 0 0 0 0 0 ...
 $ SpecialCH     : num  0 0 0 0 0 1 1 1 0 0 ...
 $ SpecialMM     : num  1 0 0 1 0 0 0 0 0 0 ...
 $ LoyalCH       : num  0.6 0.4 0.986 0.4 0.744 ...
 $ SalePriceMM   : num  1.69 1.69 1.59 2.18 2.18 2.13 2.13 1.69 2.13 2.23 ...
 $ SalePriceCH   : num  1.75 1.69 1.75 1.86 1.86 1.59 1.49 1.69 1.75 1.79 ...
 $ PriceDiff     : num  -0.06 0 -0.16 0.32 0.32 0.54 0.64 0 0.38 0.44 ...
 $ Store7        : Factor w/ 2 levels "No","Yes": 1 1 2 1 2 2 2 1 1 1 ...
 $ PctDiscMM     : num  0.151 0 0.201 0 0 ...
 $ PctDiscCH     : num  0 0 0 0 0 ...
 $ ListPriceDiff : num  0.24 0 0.24 0.32 0.32 0.27 0.27 0 0.14 0.44 ...
 $ STORE         : num  1 1 0 2 0 0 0 2 1 4 ...
```

(b)

以下為用 training set 訓練所有的變數當 covariate、預測 Purchase 的模型，共有 9 個最終節點。

使用 testing set 的 Error rate 為 0.1703704

```
1) root 800 1073.00 CH ( 0.60625 0.39375 )
  2) LoyalCH < 0.5036 365  441.60 MM ( 0.29315 0.70685 )
    4) LoyalCH < 0.280875 177  140.50 MM ( 0.13559 0.86441 )
      8) LoyalCH < 0.0356415 59   10.14 MM ( 0.01695 0.98305 ) *
      9) LoyalCH > 0.0356415 118  116.40 MM ( 0.19492 0.80508 ) *
    5) LoyalCH > 0.280875 188  258.00 MM ( 0.44149 0.55851 )
     10) PriceDiff < 0.05 79   84.79 MM ( 0.22785 0.77215 )
       20) SpecialCH < 0.5 64   51.98 MM ( 0.14062 0.85938 ) *
       21) SpecialCH > 0.5 15   20.19 CH ( 0.60000 0.40000 ) *
     11) PriceDiff > 0.05 109  147.00 CH ( 0.59633 0.40367 ) *
  3) LoyalCH > 0.5036 435  337.90 CH ( 0.86897 0.13103 )
    6) LoyalCH < 0.764572 174  201.00 CH ( 0.73563 0.26437 )
     12) ListPriceDiff < 0.235 72   99.81 MM ( 0.50000 0.50000 )
       24) PctDiscMM < 0.196196 55   73.14 CH ( 0.61818 0.38182 ) *
       25) PctDiscMM > 0.196196 17   12.32 MM ( 0.11765 0.88235 ) *
     13) ListPriceDiff > 0.235 102   65.43 CH ( 0.90196 0.09804 ) *
    7) LoyalCH > 0.764572 261   91.20 CH ( 0.95785 0.04215 ) *
```



因為 Purchase 預測出來是 CH 和 EE 分別的機率，所以有寫一個 for loop 把所有的 testing data 都看過一次，如果該筆資料 CH 的機率大於 EE，則將之設為 CH，然後再判斷實際上是否為 CH，若不是，則將 Error 的 counter+1（EE 也是相同概念）。最後，Error rate=Error 除以 270。（270 是 testing set 的大小）。

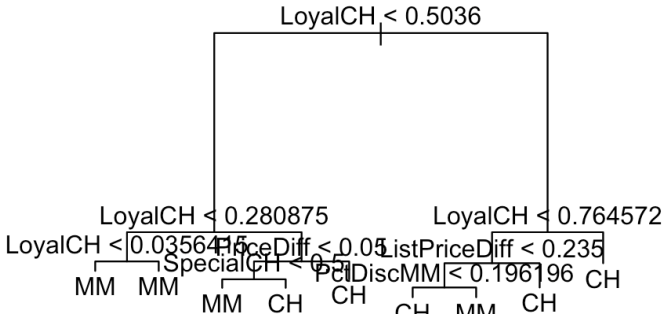
(c)

21 號 node 是最終節點(在樹的結構中最底層的 node)，分割準則(split criterion)是 **SpecialCH > 0.5**，觀察到 15 筆數據，偏差值為 20.19，預測為 CH，在此群組中，60%的實際觀察值為 CH、其他則為 MM。

```
1) root 800 1073.00 CH ( 0.60625 0.39375 )
  2) LoyalCH < 0.5036 365  441.60 MM ( 0.29315 0.70685 )
    4) LoyalCH < 0.280875 177  140.50 MM ( 0.13559 0.86441 )
      8) LoyalCH < 0.0356415 59   10.14 MM ( 0.01695 0.98305 ) *
      9) LoyalCH > 0.0356415 118  116.40 MM ( 0.19492 0.80508 ) *
    5) LoyalCH > 0.280875 188  258.00 MM ( 0.44149 0.55851 )
     10) PriceDiff < 0.05 79   84.79 MM ( 0.22785 0.77215 )
       20) SpecialCH < 0.5 64   51.98 MM ( 0.14062 0.85938 ) *
       21) SpecialCH > 0.5 15   20.19 CH ( 0.60000 0.40000 ) *
     11) PriceDiff > 0.05 109  147.00 CH ( 0.59633 0.40367 ) *
  3) LoyalCH > 0.5036 435  337.90 CH ( 0.86897 0.13103 )
    6) LoyalCH < 0.764572 174  201.00 CH ( 0.73563 0.26437 )
     12) ListPriceDiff < 0.235 72   99.81 MM ( 0.50000 0.50000 )
       24) PctDiscMM < 0.196196 55   73.14 CH ( 0.61818 0.38182 ) *
       25) PctDiscMM > 0.196196 17   12.32 MM ( 0.11765 0.88235 ) *
     13) ListPriceDiff > 0.235 102   65.43 CH ( 0.90196 0.09804 ) *
    7) LoyalCH > 0.764572 261   91.20 CH ( 0.95785 0.04215 ) *
```

(d)

從下面的 tree 可觀察到，預測 Purchases 最重要的影響因素為 LoyalCH，因為在樹的最上層，即針對此因素做區分，且最上層的三個 nodes 皆以 LoyalCH 作為區分標準。



(e)

預測出來是 CH 和 MM 分別的機率，於是使用一個自己寫的 for 迴圈（同前面小題），判斷該情況會預測為 CH 或是 MM，並存進一個新的 list 中，再拿這個 list 和實際值算出混淆矩陣。

Testing set 中預測錯誤的有 46 筆，Error Rate = 0.17037037。

混淆矩陣：

```
pred_list  CH  MM
       CH 160  38
       MM   8  64
```

Error Rate = 46/270 = 0.17037037

(f)

使用 cv.tree()，做交叉驗證以選擇樹最佳的複雜度

```
$size
[1] 9 8 7 4 2 1

$dev
[1] 152 152 149 153 173 315

$k
[1]       -Inf   0.000000   3.000000   4.333333  10.500000
[6] 151.000000

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

(g)

cross-validated classification error rate：

```
> cv.oj$dev/ nrow(train)
[1] 0.19000 0.19000 0.18625 0.19125 0.21625 0.39375
```

以 size 為 x 軸、cross-validated classification error rate 為 y 軸做圖，可觀察到在不同大小時的 Error rate，隨著 Tree size 的增加，Error rate 也逐漸下降。

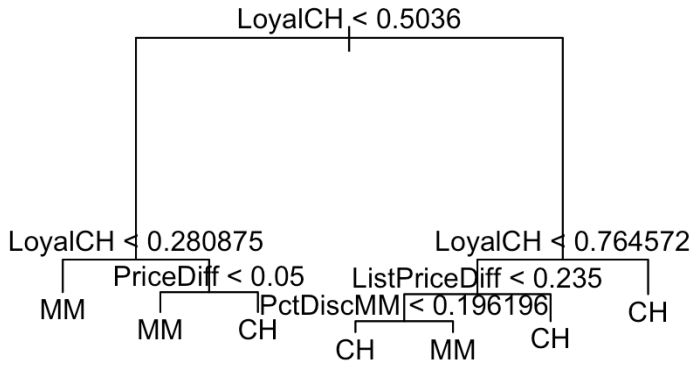
(h)

當 tree size = 7 時，有最小的 error rate

```
> min_idx = which.min(cv.oj$dev)
> min_idx
[1] 3
> cv.oj$size[min_idx]
[1] 7
```

(i)

由上小題可知，最佳的 size 為 7，於是將樹做修剪，畫出較小的樹。




```
Classification tree:
snip.tree(tree = tree2, nodes = c(10L, 4L))
Variables actually used in tree construction:
[1] "LoyalCH"       "PriceDiff"     "ListPriceDiff"
[4] "PctDiscMM"
Number of terminal nodes:  7
Residual mean deviance:  0.7748 = 614.4 / 793
Misclassification error rate: 0.1625 = 130 / 800
```

(j)

下面是 unpruned

training set 的混淆矩陣：

predicted \ actual	CH	MM
CH	450	92
MM	35	223

Accuracy = 673/800 = 0.84125

Error Rate = 127/800 = 0.15875

下面是 pruned

training set 的混淆矩陣：

predicted \ actual	CH	MM
CH	441	86
MM	44	229

Accuracy = 670/800 = 0.8375

Error Rate = 130/800 = 0.1625

使用 training set 時，Unpruned 的 Accuracy 比較高、Error rate 比較低。

(k)

下面是 unpruned

testing set 的混淆矩陣：

predicted \ actual	CH	MM
CH	160	38
MM	8	64

Accuracy = 224/270 = 0.82962963

Error Rate = 46/270 = 0.17037037

下面是 pruned

testing set 的混淆矩陣：

predicted \ actual	CH	MM
CH	160	36
MM	8	66

Accuracy = 226/270 = 0.83703704

Error Rate = 44/270 = 0.16296296

使用 testing set 時，Pruned 的 Accuracy 比較高、Error rate 比較低。

總結以上，修剪過的樹在 testing set 表現較未修剪的好，但仍然要考慮 overfitting 的問題。

10.

(a)

將 Salary 為 NA 的刪除，並將 Salary 取 log

```
     AtBat            Hits           HmRun            Runs             RBI       
 Min.   : 19.0   Min.   :  1.0   Min.   : 0.00   Min.   :  0.00   Min.   :  0.00  
 1st Qu.:282.5   1st Qu.: 71.5   1st Qu.: 5.00   1st Qu.: 33.50   1st Qu.: 30.00  
 Median :413.0   Median :103.0   Median : 9.00   Median : 52.00   Median : 47.00  
 Mean   :403.6   Mean   :107.8   Mean   :11.62   Mean   : 54.75   Mean   : 51.49  
 3rd Qu.:526.0   3rd Qu.:141.5   3rd Qu.:18.00   3rd Qu.: 73.00   3rd Qu.: 71.00  
 Max.   :687.0   Max.   :238.0   Max.   :40.00   Max.   :130.00   Max.   :121.00  
     Walks            Years            CAtBat           CHits       
 Min.   :  0.00   Min.   : 1.000   Min.   :   19.0   Min.   :   4.0  
 1st Qu.: 23.00   1st Qu.: 4.000   1st Qu.:  842.5   1st Qu.: 212.0  
 Median : 37.00   Median : 6.000   Median : 1931.0   Median : 516.0  
 Mean   : 41.11   Mean   : 7.312   Mean   : 2657.5   Mean   : 722.2  
 3rd Qu.: 57.00   3rd Qu.:10.000   3rd Qu.: 3890.5   3rd Qu.:1054.0  
 Max.   :105.00   Max.   :24.000   Max.   :14053.0   Max.   :4256.0  
     CHmRun           CRuns             CRBI            CWalks       League  Division
 Min.   :  0.00   Min.   :   2.0   Min.   :   3.0   Min.   :   1.0   A:139   E:129   
 1st Qu.: 15.00   1st Qu.: 105.5   1st Qu.:  95.0   1st Qu.:  71.0   N:124   W:134   
 Median : 40.00   Median : 250.0   Median : 230.0   Median : 174.0                   
 Mean   : 69.24   Mean   : 361.2   Mean   : 330.4   Mean   : 260.3                   
 3rd Qu.: 92.50   3rd Qu.: 497.5   3rd Qu.: 424.5   3rd Qu.: 328.5                   
 Max.   :548.00   Max.   :2165.0   Max.   :1659.0   Max.   :1566.0                   
    PutOuts          Assists          Errors           Salary       NewLeague
 Min.   :   0.0   Min.   :  0.0   Min.   : 0.000   Min.   :  67.5   A:141    
 1st Qu.: 113.5   1st Qu.:  8.0   1st Qu.: 3.000   1st Qu.: 190.0   N:122    
 Median : 224.0   Median : 45.0   Median : 7.000   Median : 425.0            
 Mean   : 290.7   Mean   :118.8   Mean   : 8.593   Mean   : 535.9            
 3rd Qu.: 322.5   3rd Qu.:192.0   3rd Qu.:13.000   3rd Qu.: 750.0            
 Max.   :1377.0   Max.   :492.0   Max.   :32.000   Max.   :2460.0            
```

計算各變數的最大、最小值、四分位數、中位數、平均

```
'data.frame':   263 obs. of  20 variables:
 $ AtBat    : int  315 479 496 321 594 185 298 323 401 574 ...
 $ Hits     : int  81 130 141 87 169 37 73 81 92 159 ...
 $ HmRun    : int  7 18 20 10 4 1 0 6 17 21 ...
 $ Runs     : int  24 66 65 39 74 23 24 26 49 107 ...
 $ RBI      : int  38 72 78 42 51 8 24 32 66 75 ...
 $ Walks    : int  39 76 37 30 35 21 7 8 65 59 ...
 $ Years    : int  14 3 11 2 11 2 3 2 13 10 ...
 $ CAtBat   : int  3449 1624 5628 396 4408 214 509 341 5206 4631 ...
 $ CHits    : int  835 457 1575 101 1133 42 108 86 1332 1300 ...
 $ CHmRun   : int  69 63 225 12 19 1 0 6 253 90 ...
 $ CRuns    : int  321 224 828 48 501 30 41 32 784 702 ...
 $ CRBI     : int  414 266 838 46 336 9 37 34 890 504 ...
 $ CWalks   : int  375 263 354 33 194 24 12 8 866 488 ...
 $ League   : Factor w/ 2 levels "A","N": 2 1 2 2 1 2 1 2 1 1 ...
 $ Division : Factor w/ 2 levels "E","W": 2 2 1 1 2 1 2 2 1 1 ...
 $ PutOuts  : int  632 880 200 805 282 76 121 143 0 238 ...
 $ Assists  : int  43 82 11 40 421 127 283 290 0 445 ...
 $ Errors   : int  10 14 3 4 25 7 9 19 0 22 ...
 $ Salary   : num  475 480 500 91.5 750 ...
 $ NewLeague: Factor w/ 2 levels "A","N": 2 1 2 2 1 1 1 2 1 1 ...
 - attr(*, "na.action")= 'omit' Named int [1:59] 1 16 19 23 31 33 37 39 40 42 ...
  ..- attr(*, "names")= chr [1:59] "-Andy Allanson" "-Billy Beane" "-Bruce Bochte" "-Bob Bo
one" ...
```

(b)

將資料集切成 training 和 testing，training set 共有 200 筆資料、其餘則為 testing set

訓練集：

```
'data.frame':   200 obs. of  20 variables:
 $ AtBat    : int  599 616 265 211 580 370 677 490 633 572 ...
 $ Hits     : int  183 163 68 43 207 96 238 150 210 152 ...
 $ HmRun    : int  10 27 8 10 8 21 31 21 6 18 ...
 $ Runs     : int  80 83 26 26 107 49 117 69 91 105 ...
 $ RBI      : int  74 107 30 35 71 46 113 58 56 49 ...
 $ Walks    : int  32 32 29 39 105 60 53 35 59 65 ...
 $ Years    : int  5 3 7 3 5 15 5 14 6 2 ...
 $ CAtBat   : int  2482 1437 1337 498 2778 6986 2223 6126 3070 978 ...
 $ CHits    : int  715 377 339 116 978 1972 737 1839 872 249 ...
 $ CHmRun   : int  27 65 32 14 32 231 93 121 19 36 ...
 $ CRuns    : int  330 181 135 59 474 1070 349 983 420 168 ...
 $ CRBI     : int  326 227 163 55 322 955 401 707 230 91 ...
 $ CWalks   : int  158 82 128 78 417 921 171 600 274 101 ...
 $ League   : Factor w/ 2 levels "A","N": 1 1 2 1 1 2 1 1 2 1 ...
 $ Division : Factor w/ 2 levels "E","W": 1 2 2 2 1 1 1 1 2 2 ...
 $ PutOuts  : int  231 110 92 463 121 137 1377 96 367 325 ...
 $ Assists  : int  374 308 5 32 267 5 100 5 432 13 ...
 $ Errors   : int  18 15 3 8 19 9 6 3 16 3 ...
 $ Salary   : num  6.65 5.3 6.05 4.79 7.38 ...
 $ NewLeague: Factor w/ 2 levels "A","N": 1 1 1 1 1 2 1 2 2 1 ...
 - attr(*, "na.action")= 'omit' Named int [1:59] 1 16 19 23 31 33 37 39 40 42 ...
  ..- attr(*, "names")= chr [1:59] "-Andy Allanson" "-Billy Beane" "-Bruce Bochte" "-Bob Bo
```

測試集：

```
'data.frame':   63 obs. of  20 variables:
 $ AtBat    : int  323 587 550 513 268 217 526 214 236 199 ...
 $ Hits     : int  81 163 152 137 60 46 146 53 56 53 ...
 $ HmRun    : int  6 4 6 20 5 7 13 2 0 5 ...
 $ Runs     : int  26 92 92 90 24 32 71 30 27 29 ...
 $ RBI      : int  32 51 37 95 25 19 70 29 15 22 ...
 $ Walks    : int  8 70 81 90 15 9 84 23 11 21 ...
 $ Years    : int  2 6 5 14 2 4 6 2 4 3 ...
 $ CAtBat   : int  341 2695 2308 5201 350 694 2648 226 1115 514 ...
 $ CHits    : int  86 747 633 1382 78 160 715 59 270 120 ...
 $ CHmRun   : int  6 17 32 166 5 32 77 2 1 8 ...
 $ CRuns    : int  32 442 349 763 34 86 352 32 116 57 ...
 $ CRBI     : int  34 198 182 734 29 76 342 32 64 40 ...
 $ CWalks   : int  8 317 308 784 18 32 289 27 57 39 ...
 $ League   : Factor w/ 2 levels "A","N": 2 1 2 1 2 1 2 2 1 1 ...
 $ Division : Factor w/ 2 levels "E","W": 2 1 2 2 2 1 2 1 2 2 ...
 $ PutOuts  : int  143 434 262 267 442 307 303 109 125 152 ...
 $ Assists  : int  290 9 329 5 59 25 9 7 199 3 ...
 $ Errors   : int  19 3 16 3 6 1 9 3 13 5 ...
 $ Salary   : num  4.32 6.64 6.44 6.8 4.5 ...
 $ NewLeague: Factor w/ 2 levels "A","N": 2 1 2 1 2 1 2 2 1 1 ...
 - attr(*, "na.action")= 'omit' Named int [1:59] 1 16 19 23 31 33 37 39 40 42 ...
  ..- attr(*, "names")= chr [1:59] "-Andy Allanson" "-Billy Beane" "-Bruce Bochte" "-Bob Bo
```

(c)

用 GBM 套件做 boosting，調整 Shrinkage(學習步伐)，建立模型數量為 1000 的 GBM

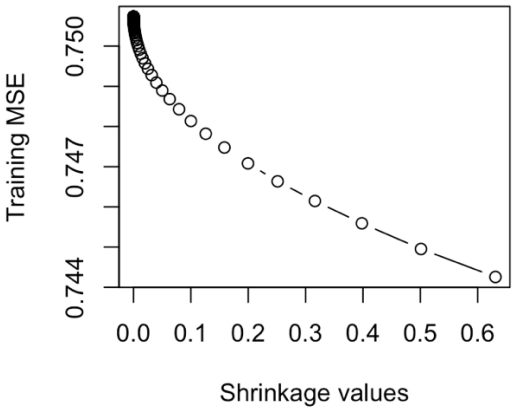
參數 Shrinkage 使用 seq()生成一串次方項，接著產生 lambda，如下圖：

```
> exp_seq
  [1] -10.00  -9.95  -9.90  -9.85  -9.80  -9.75  -9.70  -9.65  -9.60  -9.55
 [11]  -9.50  -9.45  -9.40  -9.35  -9.30  -9.25  -9.20  -9.15  -9.10  -9.05
 [21]  -9.00  -8.95  -8.90  -8.85  -8.80  -8.75  -8.70  -8.65  -8.60  -8.55
 [31]  -8.50  -8.45  -8.40  -8.35  -8.30  -8.25  -8.20  -8.15  -8.10  -8.05
 [41]  -8.00  -7.95  -7.90  -7.85  -7.80  -7.75  -7.70  -7.65  -7.60  -7.55
 [51]  -7.50  -7.45  -7.40  -7.35  -7.30  -7.25  -7.20  -7.15  -7.10  -7.05
 [61]  -7.00  -6.95  -6.90  -6.85  -6.80  -6.75  -6.70  -6.65  -6.60  -6.55
 [71]  -6.50  -6.45  -6.40  -6.35  -6.30  -6.25  -6.20  -6.15  -6.10  -6.05
 [81]  -6.00  -5.95  -5.90  -5.85  -5.80  -5.75  -5.70  -5.65  -5.60  -5.55
 [91]  -5.50  -5.45  -5.40  -5.35  -5.30  -5.25  -5.20  -5.15  -5.10  -5.05
[101]  -5.00  -4.95  -4.90  -4.85  -4.80  -4.75  -4.70  -4.65  -4.60  -4.55
[111]  -4.50  -4.45  -4.40  -4.35  -4.30  -4.25  -4.20  -4.15  -4.10  -4.05
[121]  -4.00  -3.95  -3.90  -3.85  -3.80  -3.75  -3.70  -3.65  -3.60  -3.55
[131]  -3.50  -3.45  -3.40  -3.35  -3.30  -3.25  -3.20  -3.15  -3.10  -3.05
[141]  -3.00  -2.95  -2.90  -2.85  -2.80  -2.75  -2.70  -2.65  -2.60  -2.55
[151]  -2.50  -2.45  -2.40  -2.35  -2.30  -2.25  -2.20  -2.15  -2.10  -2.05
[161]  -2.00  -1.95  -1.90  -1.85  -1.80  -1.75  -1.70  -1.65  -1.60  -1.55
[171]  -1.50  -1.45  -1.40  -1.35  -1.30  -1.25  -1.20  -1.15  -1.10  -1.05
[181]  -1.00  -0.95  -0.90  -0.85  -0.80  -0.75  -0.70  -0.65  -0.60  -0.55
[191]  -0.50  -0.45  -0.40  -0.35  -0.30  -0.25  -0.20
```

```
> lambda
  [1] 1.000000e-10 1.122018e-10 1.258925e-10 1.412538e-10 1.584893e-10
  [6] 1.778279e-10 1.995262e-10 2.238721e-10 2.511886e-10 2.818383e-10
 [11] 3.162278e-10 3.548134e-10 3.981072e-10 4.466836e-10 5.011872e-10
 [16] 5.623413e-10 6.309573e-10 7.079458e-10 7.943282e-10 8.912509e-10
 [21] 1.000000e-09 1.122018e-09 1.258925e-09 1.412538e-09 1.584893e-09
 [26] 1.778279e-09 1.995262e-09 2.238721e-09 2.511886e-09 2.818383e-09
 [31] 3.162278e-09 3.548134e-09 3.981072e-09 4.466836e-09 5.011872e-09
 [36] 5.623413e-09 6.309573e-09 7.079458e-09 7.943282e-09 8.912509e-09
 [41] 1.000000e-08 1.122018e-08 1.258925e-08 1.412538e-08 1.584893e-08
 [46] 1.778279e-08 1.995262e-08 2.238721e-08 2.511886e-08 2.818383e-08
 [51] 3.162278e-08 3.548134e-08 3.981072e-08 4.466836e-08 5.011872e-08
 [56] 5.623413e-08 6.309573e-08 7.079458e-08 7.943282e-08 8.912509e-08
 [61] 1.000000e-07 1.122018e-07 1.258925e-07 1.412538e-07 1.584893e-07
 [66] 1.778279e-07 1.995262e-07 2.238721e-07 2.511886e-07 2.818383e-07
 [71] 3.162278e-07 3.548134e-07 3.981072e-07 4.466836e-07 5.011872e-07
 [76] 5.623413e-07 6.309573e-07 7.079458e-07 7.943282e-07 8.912509e-07
 [81] 1.000000e-06 1.122018e-06 1.258925e-06 1.412538e-06 1.584893e-06
 [86] 1.778279e-06 1.995262e-06 2.238721e-06 2.511886e-06 2.818383e-06
 [91] 3.162278e-06 3.548134e-06 3.981072e-06 4.466836e-06 5.011872e-06
 [96] 5.623413e-06 6.309573e-06 7.079458e-06 7.943282e-06 8.912509e-06
[101] 1.000000e-05 1.122018e-05 1.258925e-05 1.412538e-05 1.584893e-05
[106] 1.778279e-05 1.995262e-05 2.238721e-05 2.511886e-05 2.818383e-05
[111] 3.162278e-05 3.548134e-05 3.981072e-05 4.466836e-05 5.011872e-05
[116] 5.623413e-05 6.309573e-05 7.079458e-05 7.943282e-05 8.912509e-05
[121] 1.000000e-04 1.122018e-04 1.258925e-04 1.412538e-04 1.584893e-04
[126] 1.778279e-04 1.995262e-04 2.238721e-04 2.511886e-04 2.818383e-04
[131] 3.162278e-04 3.548134e-04 3.981072e-04 4.466836e-04 5.011872e-04

[136] 5.623413e-04 6.309573e-04 7.079458e-04 7.943282e-04 8.912509e-04
[141] 1.000000e-03 1.122018e-03 1.258925e-03 1.412538e-03 1.584893e-03
[146] 1.778279e-03 1.995262e-03 2.238721e-03 2.511886e-03 2.818383e-03
[151] 3.162278e-03 3.548134e-03 3.981072e-03 4.466836e-03 5.011872e-03
[156] 5.623413e-03 6.309573e-03 7.079458e-03 7.943282e-03 8.912509e-03
[161] 1.000000e-02 1.122018e-02 1.258925e-02 1.412538e-02 1.584893e-02
[166] 1.778279e-02 1.995262e-02 2.238721e-02 2.511886e-02 2.818383e-02
[171] 3.162278e-02 3.548134e-02 3.981072e-02 4.466836e-02 5.011872e-02
[176] 5.623413e-02 6.309573e-02 7.079458e-02 7.943282e-02 8.912509e-02
[181] 1.000000e-01 1.122018e-01 1.258925e-01 1.412538e-01 1.584893e-01
[186] 1.778279e-01 1.995262e-01 2.238721e-01 2.511886e-01 2.818383e-01
[191] 3.162278e-01 3.548134e-01 3.981072e-01 4.466836e-01 5.011872e-01
[196] 5.623413e-01 6.309573e-01
```

Training set 的 MSE 如下：



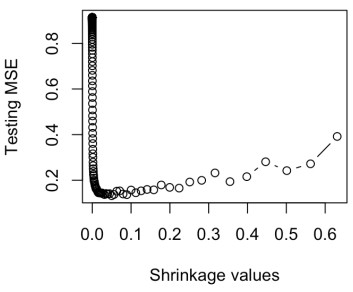

```
> train.mse
  [1] 0.750728855 0.750728845 0.750728834 0.750728821 0.750728807 0.750728791
  [7] 0.750728773 0.750728753 0.750728731 0.750728706 0.750728678 0.750728645
 [13] 0.750728610 0.750728571 0.750728527 0.750728477 0.750728419 0.750728357
 [19] 0.750728285 0.750728206 0.750728120 0.750728018 0.750727905 0.750727780
 [25] 0.750727636 0.750727485 0.750727298 0.750727104 0.750726867 0.750726620
 [31] 0.750726352 0.750726027 0.750725675 0.750725280 0.750724827 0.750724332
 [37] 0.750723778 0.750723152 0.750722424 0.750721619 0.750720759 0.750719748
 [43] 0.750718613 0.750717368 0.750715922 0.750714386 0.750712589 0.750710590
 [49] 0.750708357 0.750705836 0.750703066 0.750699829 0.750696329 0.750692266
 [55] 0.750687872 0.750682904 0.750677227 0.750670757 0.750663772 0.750655793
 [61] 0.750647234 0.750636778 0.750625885 0.750613031 0.750598507 0.750583379
 [67] 0.750564815 0.750545826 0.750523006 0.750498508 0.750469144 0.750437482
 [73] 0.750401607 0.750363565 0.750317729 0.750266162 0.750212812 0.750149963
 [79] 0.750077899 0.750000297 0.749908872 0.749815235 0.749700585 0.749568104
 [85] 0.749428725 0.749275310 0.749099597 0.748897846 0.748683355 0.748420243
```

可觀察到當 Shrinkage 越大，MSE 越小，

(d)

同樣使用(c)小題產生的 lambda，

Testing set 的 MSE 如下：




```
> test.mse
  [1] 0.9137040 0.9137040 0.9137039 0.9137039 0.9137039 0.9137039 0.9137039
  [8] 0.9137038 0.9137038 0.9137038 0.9137037 0.9137037 0.9137037 0.9137036
 [15] 0.9137036 0.9137035 0.9137034 0.9137033 0.9137032 0.9137031 0.9137030
 [22] 0.9137029 0.9137028 0.9137026 0.9137024 0.9137022 0.9137020 0.9137017
 [29] 0.9137014 0.9137011 0.9137008 0.9137004 0.9136999 0.9136994 0.9136988
 [36] 0.9136982 0.9136975 0.9136966 0.9136958 0.9136948 0.9136936 0.9136923
 [43] 0.9136909 0.9136893 0.9136875 0.9136855 0.9136832 0.9136807 0.9136779
 [50] 0.9136745 0.9136710 0.9136670 0.9136624 0.9136576 0.9136516 0.9136453
```

可觀察到當 Shrinkage 靠近 0.1，MSE 快速下降，但當 Shrinkage 大於 0.1，MSE 逐漸增加。

(e)

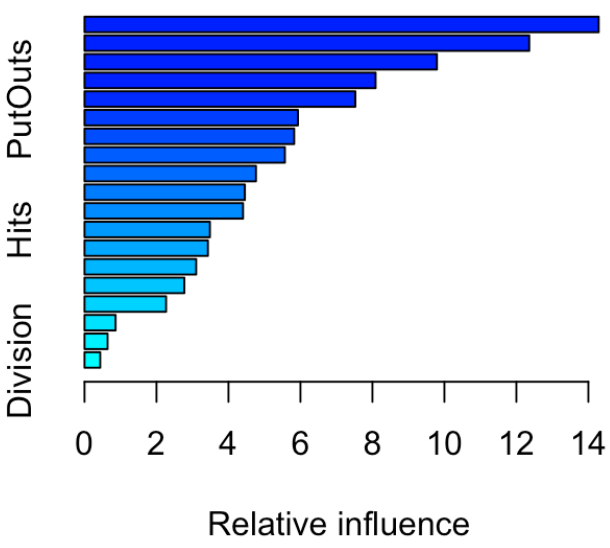
- Linear regression 的 MSE：0.3580125
- Ridge regression 的 MSE：0.3507145
- (d)小題中 Boosting 的 MSE：0.1320665

由上可知，Boosting 的 MSE 比 Linear 和 Ridge 都還小

(f)

取 Testing set MSE 最小的 Shrinkage = 0.04466836，可觀察到 CATBat 是對 Salary 影響最大的變數。

	var	rel.inf
CAtBat	CAtBat	14.2872847
CRuns	CRuns	12.3580075
CWalks	CWalks	9.7919765
CRBI	CRBI	8.0890003
PutOuts	PutOuts	7.5249610
CHmRun	CHmRun	5.9300538
CHits	CHits	5.8262239
Walks	Walks	5.5684288
AtBat	AtBat	4.7645655
Assists	Assists	4.4552929
HmRun	HmRun	4.4048774
Hits	Hits	3.4850658
Runs	Runs	3.4292788
Years	Years	3.1028409
RBI	RBI	2.7731904
Errors	Errors	2.2670147
League	League	0.8631278
NewLeague	NewLeague	0.6418964
Division	Division	0.4369131



(g)

bagging 是將樣本重複抽樣(取後放回)，產生多個子資料集後，依序建立多個模型，最後再將所有模型的結果彙整在一起，有助於降低 variance。

使用 bagging、抽 100 次 boostrap sample 建模，取 100 次平均計算 MSE = 0.1883786

下面是各個變數的重要性，可發現 CATBat 是最重要的變數

```
> B_tree$variable.importance
    CAtBat      CRuns      CHits       CRBI     CWalks      Years      AtBat 
93.5012686 83.9577862 81.1396058 74.0321125 68.4984021 48.1788829 27.4539434 
      Hits       Runs        RBI      Walks    PutOuts      HmRun     CHmRun 
24.2197726 23.8742722 18.2741671 17.9193454  6.1863769  1.0437815  0.8540031 
```

總結，以此題為例，四種方法的 MSE 依序為 Boosting < Bagging < Ridge regression < Linear regression。

附錄：R 程式碼

```
## 4.
# (b)
plot( xlim = c(-2, 2), ylim = c(-3, 3), xlab = "X1", ylab = "X2")
lines(x = c(-2, 2), y = c(1, 1))
lines(x = c(1, 1), y = c(-3, 1))
text(x = (-2 + 1)/2, y = -1, labels = c(-1.8))
text(x = 1.5, y = -1, labels = c(0.63))
lines(x = c(-2, 2), y = c(2, 2))
text(x = 0, y = 2.5, labels = c(2.49))
lines(x = c(0, 0), y = c(1, 2))
```

```
text(x = -1, y = 1.5, labels = c(-1.06))
text(x = 1, y = 1.5, labels = c(0.21))

## 8. (a)
data(Carseats, package="ISLR")
str(Carseats)
head(Carseats)
summary(Carseats)
set.seed(5)
index = sample(1:nrow(Carseats),ceiling(0.8*nrow(Carseats)))
train = Carseats[index,]
test = Carseats[-index,]

## (b)
library(rpart)
tree = rpart(Sales~.,data=train)
summary(tree)
rpart.plot(tree)
pred = predict(tree,newdata = test)
mean((pred - test$Sales)^2)  # 計算 MSE

# cp
tree = rpart(Sales~., data=train, cp=0.03)
pred = predict(tree,newdata = test)
rpart.plot(tree)
mean((pred - test$Sales)^2)

tree = rpart(Sales~., data=train, cp=0.05)
pred = predict(tree,newdata = test)
rpart.plot(tree)
mean((pred - test$Sales)^2)

tree = rpart(Sales~., data=train, cp=0.01)
pred = predict(tree,newdata = test)
rpart.plot(tree)
mean((pred - test$Sales)^2)
# minbucket
tree = rpart(Sales~., data=train, minbucket=1)
pred = predict(tree,newdata = test)
rpart.plot(tree)
mean((pred - test$Sales)^2)
```

```
tree = rpart(Sales~., data=train, minbucket=5)
pred = predict(tree,newdata = test)
rpart.plot(tree)
mean((pred - test$Sales)^2)

tree = rpart(Sales~., data=train, minbucket=10)
pred = predict(tree,newdata = test)
rpart.plot(tree)
mean((pred - test$Sales)^2)

tree = rpart(Sales~., data=train, minbucket=15)
pred = predict(tree,newdata = test)
rpart.plot(tree)
mean((pred - test$Sales)^2)

tree = rpart(Sales~., data=train, minbucket=20)
pred = predict(tree,newdata = test)
rpart.plot(tree)
mean((pred - test$Sales)^2)

## (c)
library(tree)
tree2 = tree(Sales~.,data=train)
plot(tree2) ;    text(tree2,pretty=0)
# 利用未修剪的 tree 對 test data 做預測
pred1 = predict(tree2,newdata = test)
mean((pred1 - test$Sales)^2)   # 計算 MSE
# 利用 cross-validation 找最佳樹的大小
set.seed(10)
cv_tree = cv.tree(tree2,K=5)    # K-folds CV
plot(cv_tree$size ,cv_tree$dev ,type="b")
tree.min <- which.min(cv_tree$dev)
points(tree.min, cv_tree$dev[tree.min],cex = 1.5, pch = 16)

prune_tree = prune.tree(tree2,best = 6)
plot(prune_tree)
text(prune_tree,pretty=0)

pred2 = predict(prune_tree,newdata = test)
mean((pred2 - test$Sales)^2)

## (d) bagging
```

```
B = 100
B_pred = matrix(ncol=B,nrow=nrow(test))
for (i in 1:B){
   B_sample = train[sample(1:nrow(train),replace = T),]
   B_tree = rpart(Sales~.,data=B_sample)
   B_pred[,i] = predict(B_tree,newdata=test)
}
mean((apply(B_pred,1,mean) - test$Sales)^2)
B_tree$variable.importance

## (e) RF
set.seed(10)
rf = randomForest(Sales~.,data=train)
plot(rf)
pred3 = predict(rf,newdata = test)
mean((pred3 - test$Sales)^2) #  計算 MSE
importance(rf)
varImpPlot(rf,main="variable importance plot")

rf = randomForest(Sales~., mtry = 4 ,data=train)
plot(rf)
pred3 = predict(rf,newdata = test)
mean((pred3 - test$Sales)^2)

rf = randomForest(Sales~., mtry = 6 ,data=train)
plot(rf)
pred3 = predict(rf,newdata = test)
mean((pred3 - test$Sales)^2)

rf = randomForest(Sales~., mtry = 10 ,data=train)
plot(rf)
pred3 = predict(rf,newdata = test)
mean((pred3 - test$Sales)^2)

## 9. (a)
set.seed(1)
data(OJ, package="ISLR")
str(OJ)
head(OJ)
summary(OJ)
index = sample(1:nrow(OJ), 800)
train = OJ[index,]
```

```
test = OJ[-index,]
str(train)
str(test)

## (b)
# library(rpart)
# tree = rpart(Purchase~.,data=train)
tree2 = tree(Purchase~.,data=train)
tree2
plot(tree2) ;    text(tree2,pretty=0)
pred = predict(tree2,newdata = test)    #預測 test data
error = 0
for (i in c(1:270)) {
   brand = NULL
   if (pred[i,1]>pred[i,2])
      brand = 'CH'
   else
      brand = 'MM'
   if (brand!=test[i,]$Purchase) {
      error=error+1
   }
}
error_rate = error/270

## (c)
tree2

## (d)
plot(tree2) ;    text(tree2,pretty=0)

## (e)
pred = predict(tree2,newdata = test)    #預測 test data
pred_list=NULL
for (i in c(1:270)) {
   brand = NULL
   if (pred[i,1]>pred[i,2])
      brand = 'CH'
   else
      brand = 'MM'
   pred_list[i]=brand
}
table(pred_list, test$Purchase)
```

```
## (f)
set.seed(3)
cv.oj <- cv.tree(tree2, FUN = prune.misclass)
cv.oj

## (g)
plot(cv.oj$size, cv.oj$dev / nrow(train), type = "b",
     xlab = "Tree Size", ylab = "CV classification error rate")
cv.oj$dev/ nrow(train)

## (h)
min_idx = which.min(cv.oj$dev)
min_idx
cv.oj$size[min_idx]

## (i)
prune_tree = prune.tree(tree2,best = 7)   #修剪
plot(prune_tree)
text(prune_tree,pretty=0)
summary(prune_tree)

## (j)
### unpruned
unprune_trn_pred = predict(tree2, train, type = "class")
table(predicted = unprune_trn_pred, actual = train$Purchase)
### pruned
prune_trn_pred = predict(prune_tree, train, type = "class")
table(predicted = prune_trn_pred, actual = train$Purchase)

## (k)
### unpruned
unprune_tst_pred = predict(tree2, test, type = "class")
table(predicted = unprune_tst_pred, actual = test$Purchase)
### pruned
prune_tst_pred = predict(prune_tree, test, type = "class")
table(predicted = prune_tst_pred, actual = test$Purchase)

## 10. (a)
data(Hitters, package="ISLR")
str(Hitters)
head(Hitters)
```

```
summary(Hitters)
Hitters <- na.omit(Hitters)
Hitters$Salary <- log(Hitters$Salary)

## (b)
index = sample(1:nrow(Hitters), 200)
train = Hitters[index,]
test = Hitters[-index,]
str(train)
str(test)

## (c) boosting
#install.packages("gbm")
library(gbm)
set.seed(1)
exp_seq <- seq(-10, -0.2, by = 0.05)
lambda <- 10^exp_seq
train.mse <- rep(NA, length(lambda))
for (i in 1:length(lambda)) {
   boost.hitters <- gbm(Salary ~ ., data = train, distribution = "gaussian", n.trees = 1000, shrinkage =
lambda[i])
   pred.train <- predict(boost.hitters, train, n.trees = 1000)
   train.mse[i] <- mean((pred.train - train$Salary)^2)
}
plot(lambdas, train.mse, type = "b", xlab = "Shrinkage values", ylab = "Training MSE")
train.mse
## (d)
test.mse <- rep(NA, length(lambda))
for (i in 1:length(lambda)) {
   boost.hitters <- gbm(Salary ~ ., data = train, distribution = "gaussian", n.trees = 1000, shrinkage =
lambda[i])
   yhat <- predict(boost.hitters, test, n.trees = 1000)
   test.mse[i] <- mean((yhat - test$Salary)^2)
}
plot(lambda, test.mse, type = "b", xlab = "Shrinkage values", ylab = "Testing MSE")
test.mse
min(test.mse)
lambda[which.min(test.mse)]
## (e)
### linear regression
library(glmnet)
fit <- lm(Salary ~ ., data = train)
```

```
pred_lm <- predict(fit, test)
mean((pred_lm - test$Salary)^2)
### ridge regression
x <- model.matrix(Salary ~ ., data = train)
x.test <- model.matrix(Salary ~ ., data = test)
y <- train$Salary
fit2 <- glmnet(x, y, alpha = 0)
pred_rid <- predict(fit2, s = 0.01, newx = x.test)
mean((pred_rid - test$Salary)^2)
## (f)
boost.Salary = gbm(Salary~., data = train, distribution = "gaussian",
                                n.trees = 1000, interaction.depth = 4, shrinkage = lambda[which.min(test.mse)])
summary(boost.Salary)
## (g)
B = 100
B_pred = matrix(ncol=B,nrow=nrow(test))
for (i in 1:B){
   B_sample = train[sample(1:nrow(train),replace = T),]
   B_tree = rpart(Salary~.,data=B_sample)
   B_pred[,i] = predict(B_tree,newdata=test)
}
mean((apply(B_pred,1,mean) - test$Salary)^2)
B_tree$variable.importance
```