Session 04

Model selection

An example: Cars 93 data

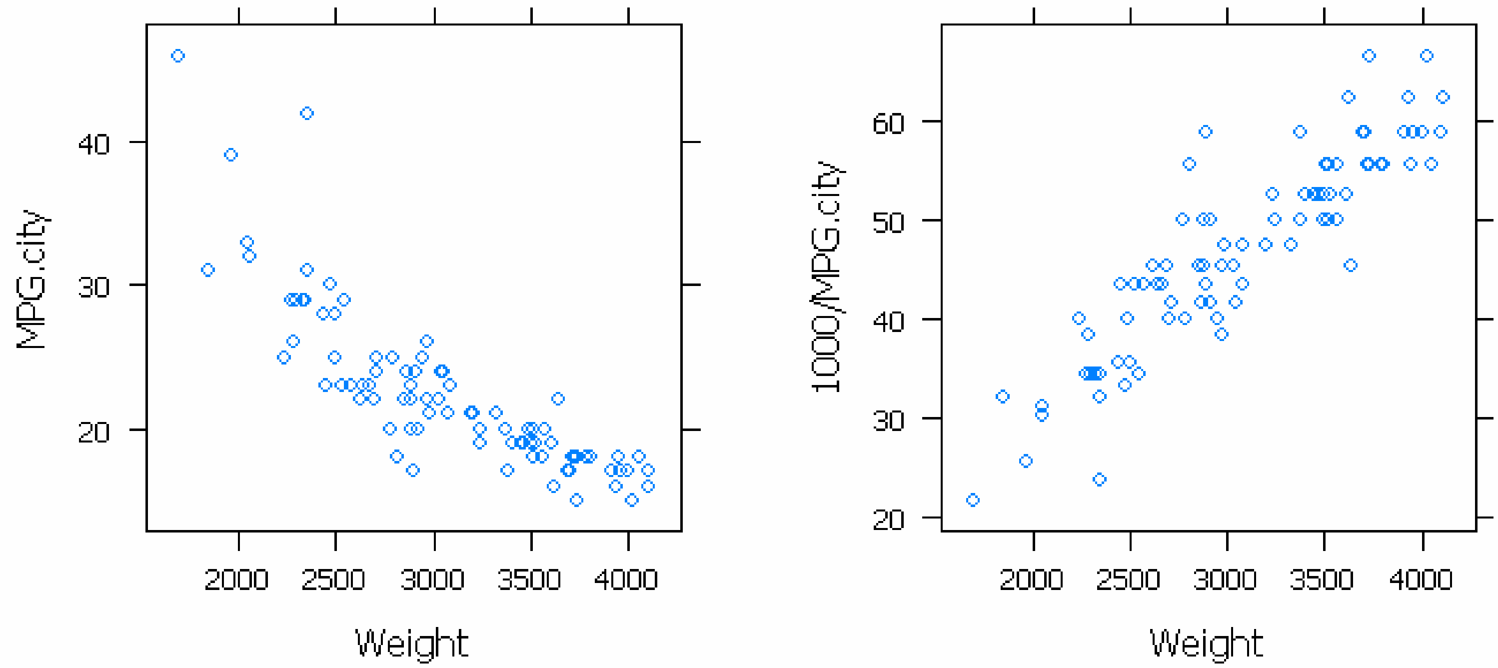
- Details (~data) from 93 makes of car released in the USA in 1993. Variable names largely self-explanatory.
- Problem: build a prediction equation for the fuel economy from the other variables available.

```
print(names(Cars93), quote = F)

 [1] Manufacturer       Type               Min.Price
 [4] Price              Max.Price          MPG.city
 [7] MPG.highway        AirBags            DriveTrain
[10] Cylinders          EngineSize         Horsepower
[13] RPM                Rev.per.mile       Man.trans.avail
[16] Fuel.tank.capacity Passengers         Length
[19] Wheelbase          Width              Turn.circle
[22] Rear.seat.room     Luggage.room       Weight
[25] Origin             Make
```

Scale of the response

- As the response we choose `MPG.city`
- The dominant predictor will (presumably) be the weight of the vehicle
- Consider some exploratory plots:
  - **`MPG.city`** vs **`Weight`**
  - **`1000/MPG.city`** vs **`Weight`**

```
require(lattice)
p1 <- xyplot(MPG.city ~ Weight, Cars93)
p2 <- xyplot(1000/MPG.city ~ Weight, Cars93)
print(p1, c(0, 0.5, 0.5, 1), more = T)
print(p2, c(0.5, 0.5, 1, 1))
```



- The first scale asymptotes to zero and the variance contracts for large weight.
- The second scale is open-ended for large vehicles and shows much more variance stability
- Either scale is a convenient one for fuel economy

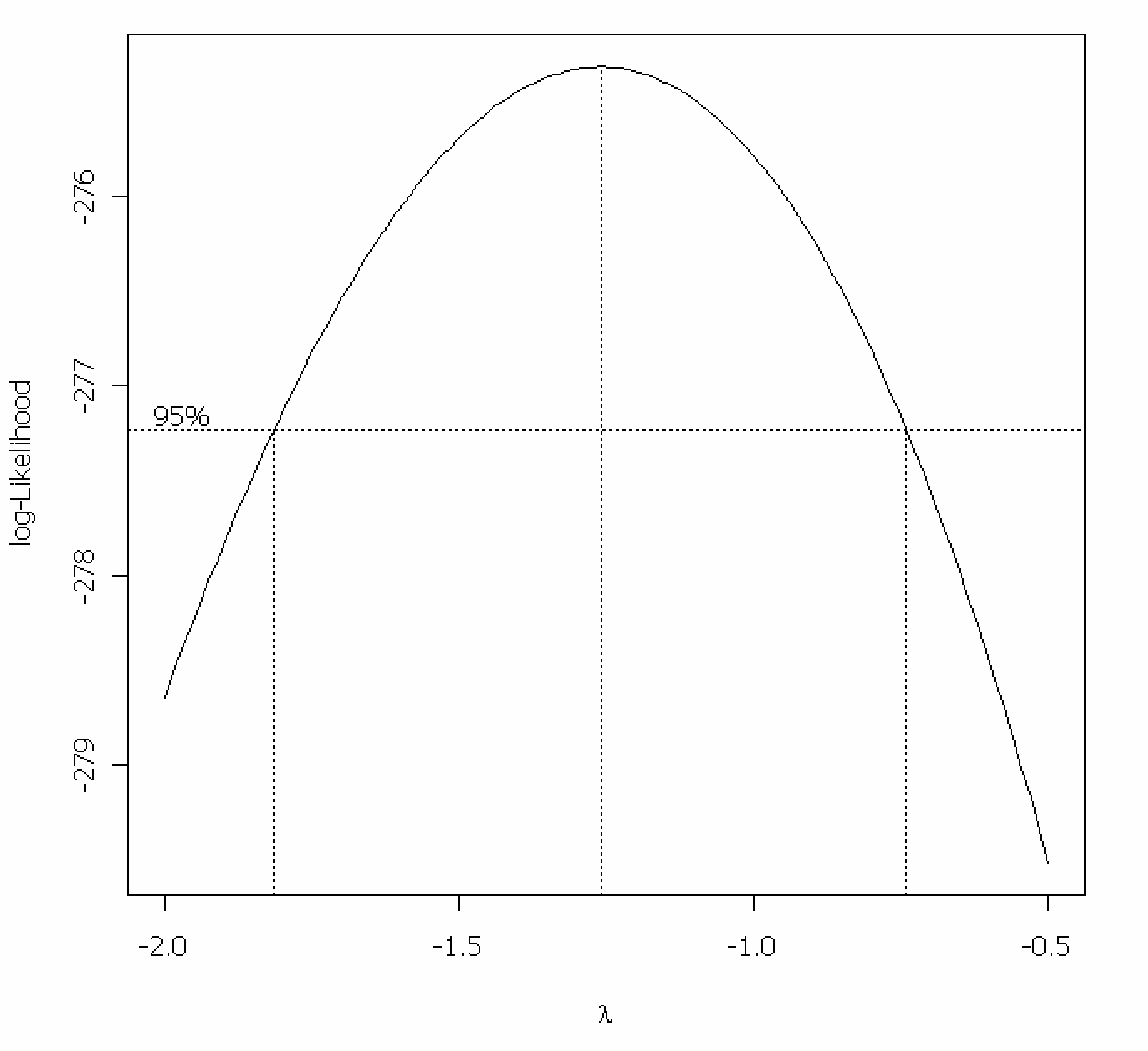
Box-cox transformations

- Device for choosing a scale which is a power transform of the original. (See introductory session.)

```
Cars93.lm <- lm(MPG.city ~ Weight, Cars93)
boxcox(Cars93.lm, lambda = seq(-2, -0.5, len=15))
```

- Since λ = -1 is well within the acceptable range (next slide), this is the scale we confirm.
- Now look for other variables that might improve the prediction.

```
Cars93.lm <- update(Cars93.lm, 1000/MPG.city ~ .)
```



© C

Automated selection of variables

- It is never a good idea to entrust the selection of variables in a regression entirely to some automated procedure.
- It is, nevertheless, often quite a good idea to take into account which variables such procedures suggest as important, along with other things.
- We fit an intermediate regression and consider an automated procedure that steps “up and down”
- Rather than minimize AIC, we choose BIC, which penalizes redundant variables much harder.

Initial model

```
Cars93.lm1 <- lm(1000/MPG.city ~ Type * (Weight +
  Horsepower + Length), Cars93)

dropterm(Cars93.lm1, test = "F", k = log(93))
```

```
Single term deletions

Model:
1000/MPG.city ~ Type * (Weight + Horsepower + Length)
                Df Sum of Sq      RSS      AIC  F Value     Pr(F)
         <none>               1059.993 335.0903
    Type:Weight  5  163.6090 1223.602 325.7762 2.130018 0.0720422
Type:Horsepower  5   92.1563 1152.150 320.1804 1.199779 0.3185266
    Type:Length  5  149.1609 1209.154 324.6715 1.941918 0.0984718
```

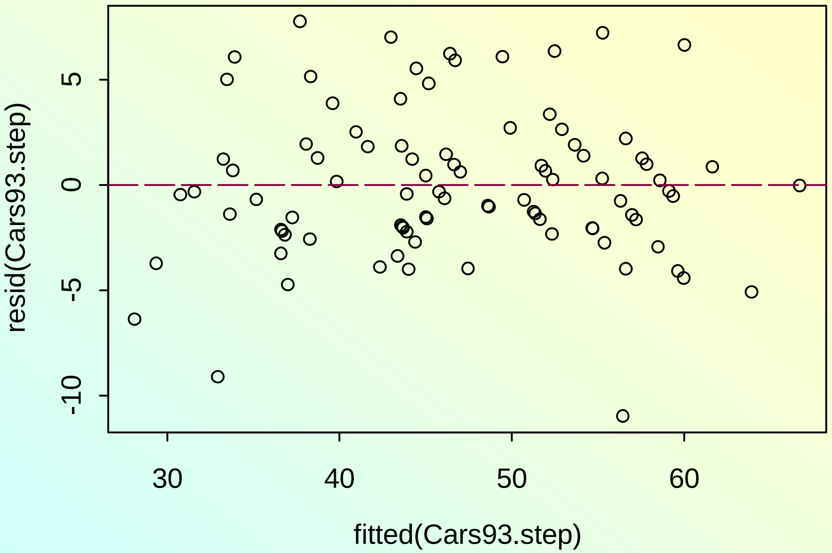
- Notice that only the marginal terms are dropped and none are significant.

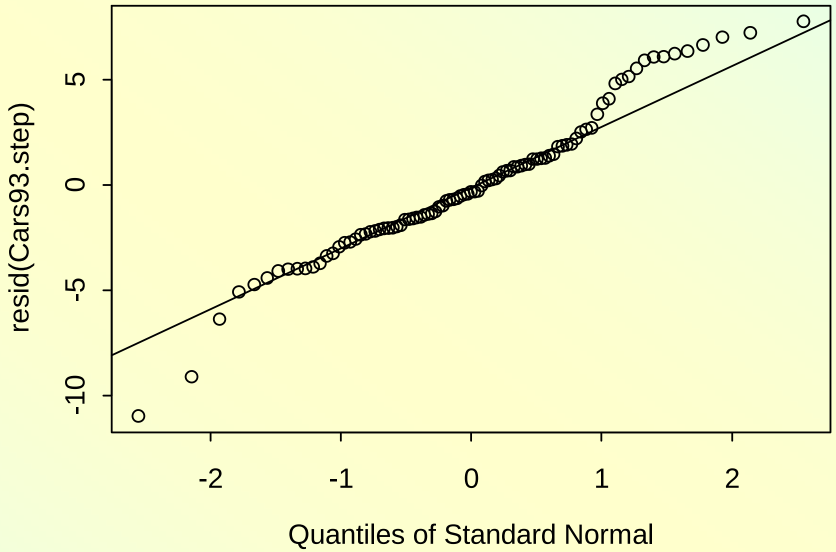
Stepwise refinement

```
Cars93.step <- stepAIC(Cars93.lm1, scope = list(lower = ~
   Weight, upper = ~ Type*(Min.Price + Price + Max.Price +
   AirBags + DriveTrain + Cylinders + EngineSize + Horsepower
   + RPM + Rev.per.mile + Fuel.tank.capacity + Passengers +
   Length + Wheelbase + Width + Turn.circle + Weight +
   Origin)), k = log(93))

dropterm(Cars93.step, test = "F", k = log(93), sorted = T)
```

```
Single term deletions

Model:
1000/MPG.city ~ Weight + Length + Fuel.tank.capacity + Origin +
    Min.Price
                   Df Sum of Sq     RSS    AIC F Value     Pr(F)
<none>                          1126.91 259.20
Weight              1    362.04 1488.95 280.57   27.95 9.137e-07
Length              1    122.42 1249.33 264.25    9.45 0.0028192
Fuel.tank.capacity  1    223.10 1350.01 271.46   17.22 7.718e-05
Origin              1    188.66 1315.57 269.06   14.57 0.0002529
Min.Price           1    153.14 1280.05 266.51   11.82 0.0009001
```

```
par(mfrow=c(2,2))
plot(fitted(Cars93.step), resid(Cars93.step))
abline(h = 0, lty = 4, col = 3)
qqnorm(resid(Cars93.step))
qqline(resid(Cars93.step))
```





What happens if we use AIC?

```
Cars93.AIC <- stepAIC(Cars93.lm,
   scope = list(lower = ~ Weight, upper = ~ Type +
   Min.Price +
   Price + Max.Price + AirBags + DriveTrain + Cylinders
   + EngineSize + Horsepower + RPM + Rev.per.mile +
   Fuel.tank.capacity + Passengers + Length + Wheelbase
   + Width + Turn.circle + Weight + Origin), k = 2)

dropterm(Cars93.AIC, test = "F", sorted = T)
```

```
Single term deletions

Model:
1000./MPG.city ~ Weight + Cylinders + Fuel.tank.capacity + Length +
  Origin + Min.Price + Horsepower + Wheelbase
                   Df Sum of Sq      RSS      AIC  F Value     Pr(F)
            <none>                938.132 240.9501
         Wheelbase  1   24.2090  962.341 241.3195  2.06444 0.1546699
        Horsepower  1   45.2546  983.387 243.3315  3.85913 0.0529484
            Length  1   64.4150 1002.547 245.1260  5.49305 0.0215748
         Cylinders  5  157.5719 1095.704 245.3894  2.68742 0.0268981
         Min.Price  1   82.2667 1020.399 246.7675  7.01536 0.0097334
            Origin  1  105.3495 1043.481 248.8478  8.98377 0.0036272
Fuel.tank.capacity  1  156.0240 1094.156 253.2579 13.30508 0.0004697
            Weight  1  239.4604 1177.592 260.0924 20.42019 0.0000212
```

- Much less stringent choice of variables. Perhaps we should remove some starting with the least significant. The ‘backwards elimination’ sequence is as follows:

```
Cars93.AIC <- update(Cars93.AIC, .~.-Wheelbase)
dropterm(Cars93.AIC, test = "F", sorted = T)
Cars93.AIC <- update(Cars93.AIC, .~.-Horsepower)
dropterm(Cars93.AIC, test = "F", sorted = T)
Cars93.AIC <- update(Cars93.AIC, .~.-Cylinders)
dropterm(Cars93.AIC, test = "F", sorted = T)
```

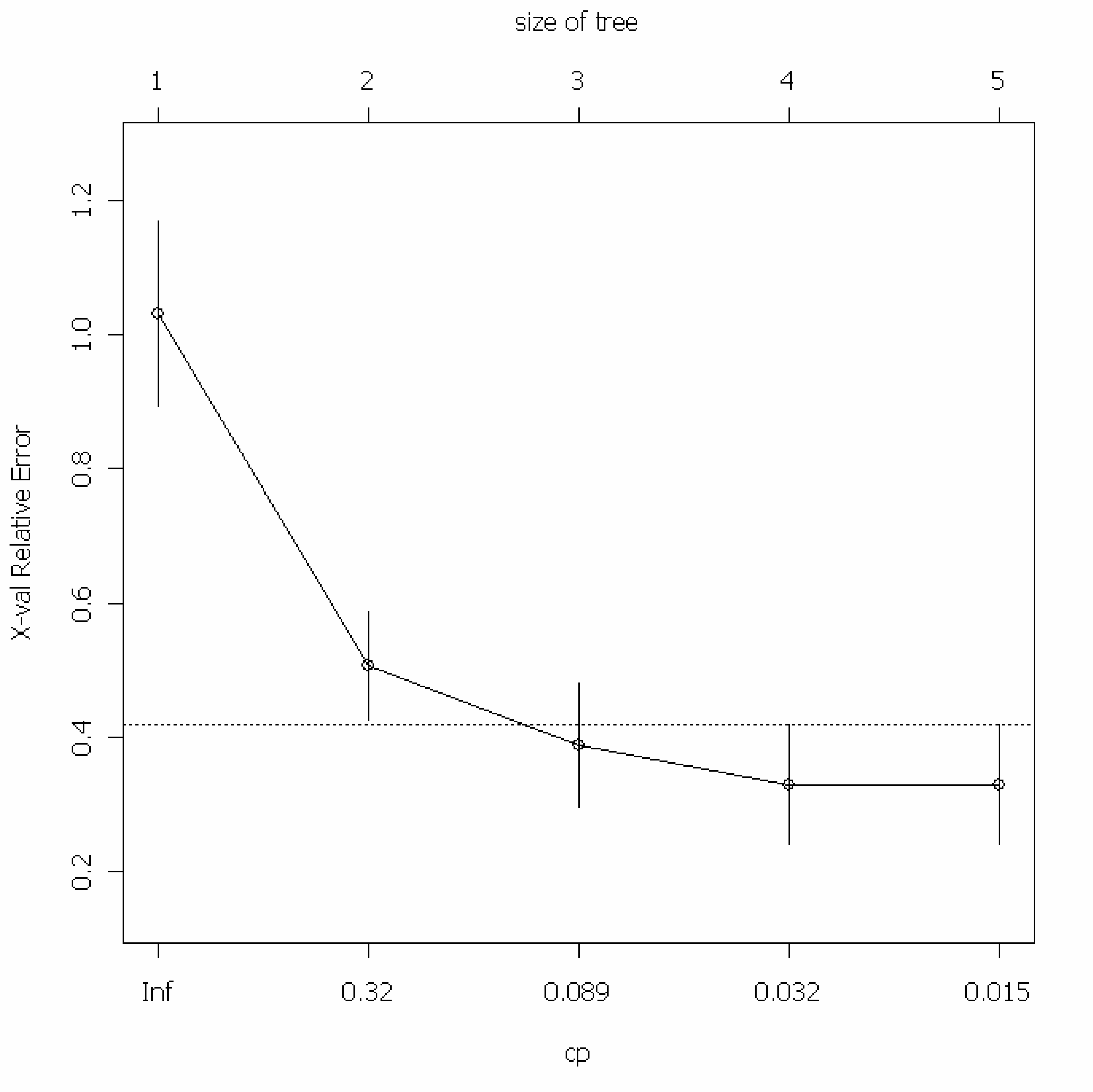
```
Single term deletions

Model:
1000./MPG.city ~ Weight + Fuel.tank.capacity + Length + Origin +
  Min.Price
                   Df Sum of Sq      RSS      AIC F Value  Pr(F)
            <none>              1126.909 244.0010
            Length  1  122.4164 1249.326 251.5917  9.4508 0.0028
         Min.Price  1  153.1380 1280.047 253.8509 11.8226 0.0009
            Origin  1  188.6606 1315.570 256.3966 14.5650 0.0003
Fuel.tank.capacity  1  223.0965 1350.006 258.7996 17.2236 0.0001
            Weight  1  362.0418 1488.951 267.9102 27.9505 0.0000
```

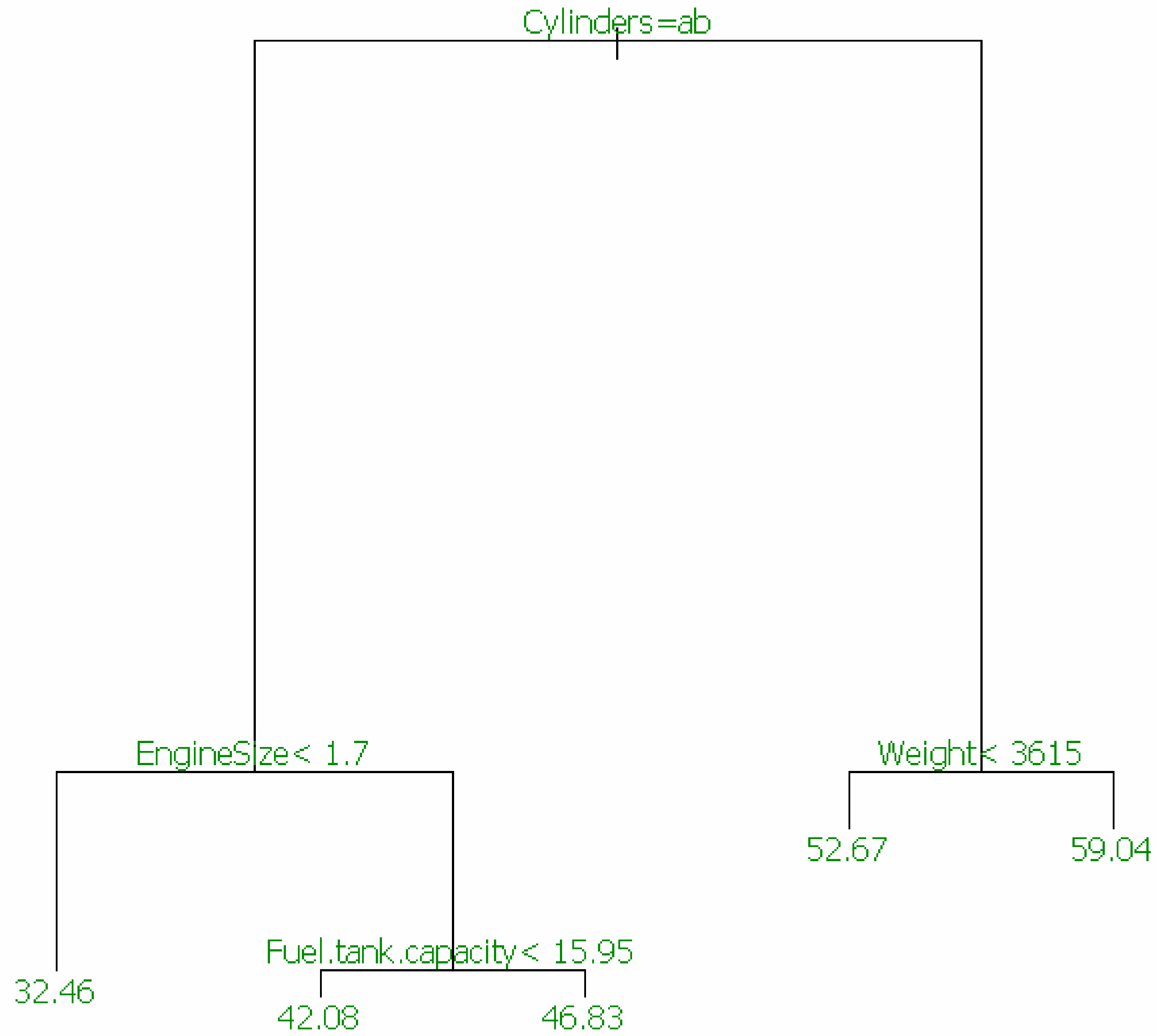
Notes

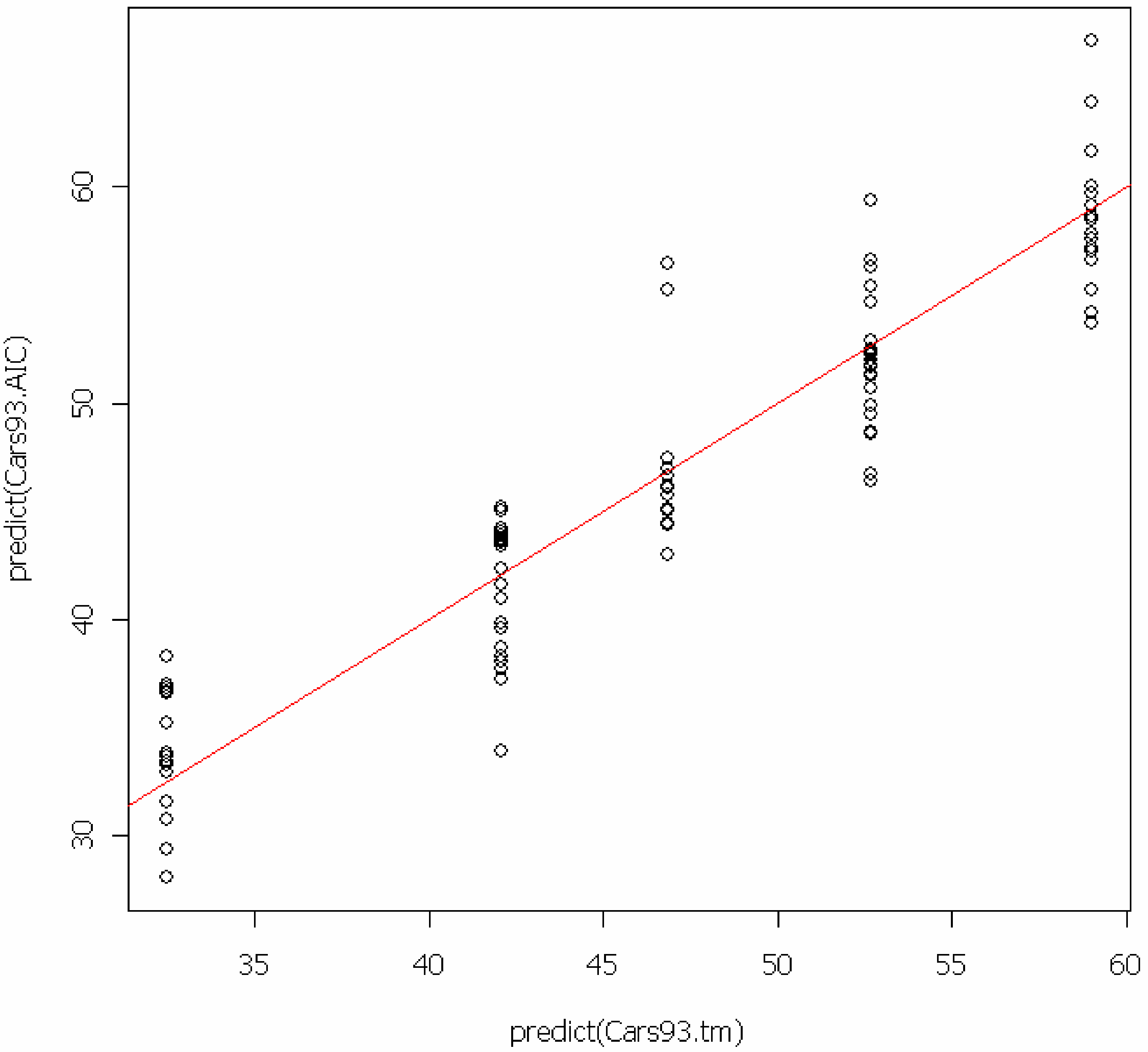
- All interaction terms have been removed
- With the BIC model
  - “`Type`” is not present, but “`Origin`” is.
  - “`Min.price`” is presumably a surrogate variable for engineering refinements
- AIC model is very different, but has a slightly lower multiple $R^{2.}$ (Probably a very biased equation)
- Consider the standard diagnostic plots for the BIC model:
  - residuals vs fitted values,
  - normal scores plot of the residuals

Tree modelling strategy

```
#### now for something completely different

require(rpart)
Cars93.tm <- rpart(I(1000/MPG.city) ~ Type + Min.Price
   + Price + Max.Price + AirBags + DriveTrain +
   Cylinders + EngineSize + Horsepower + RPM +
   Rev.per.mile + Fuel.tank.capacity + Passengers +
   Length + Wheelbase + Width + Turn.circle + Weight +
   Origin, Cars93)

plotcp(Cars93.tm)
plot(Cars93.tm); text(Cars93.tm, col = "green4")


plot(predict(Cars93.tm), predict(Cars93.AIC))
abline(0, 1, lty = "solid", col = "red")
```


size of tree
1
2
3
4
5
X-val Relative Error
1.2
1.0
0.8
0.6
0.4
0.2
Inf
0.32
0.089
0.032
0.015
cp

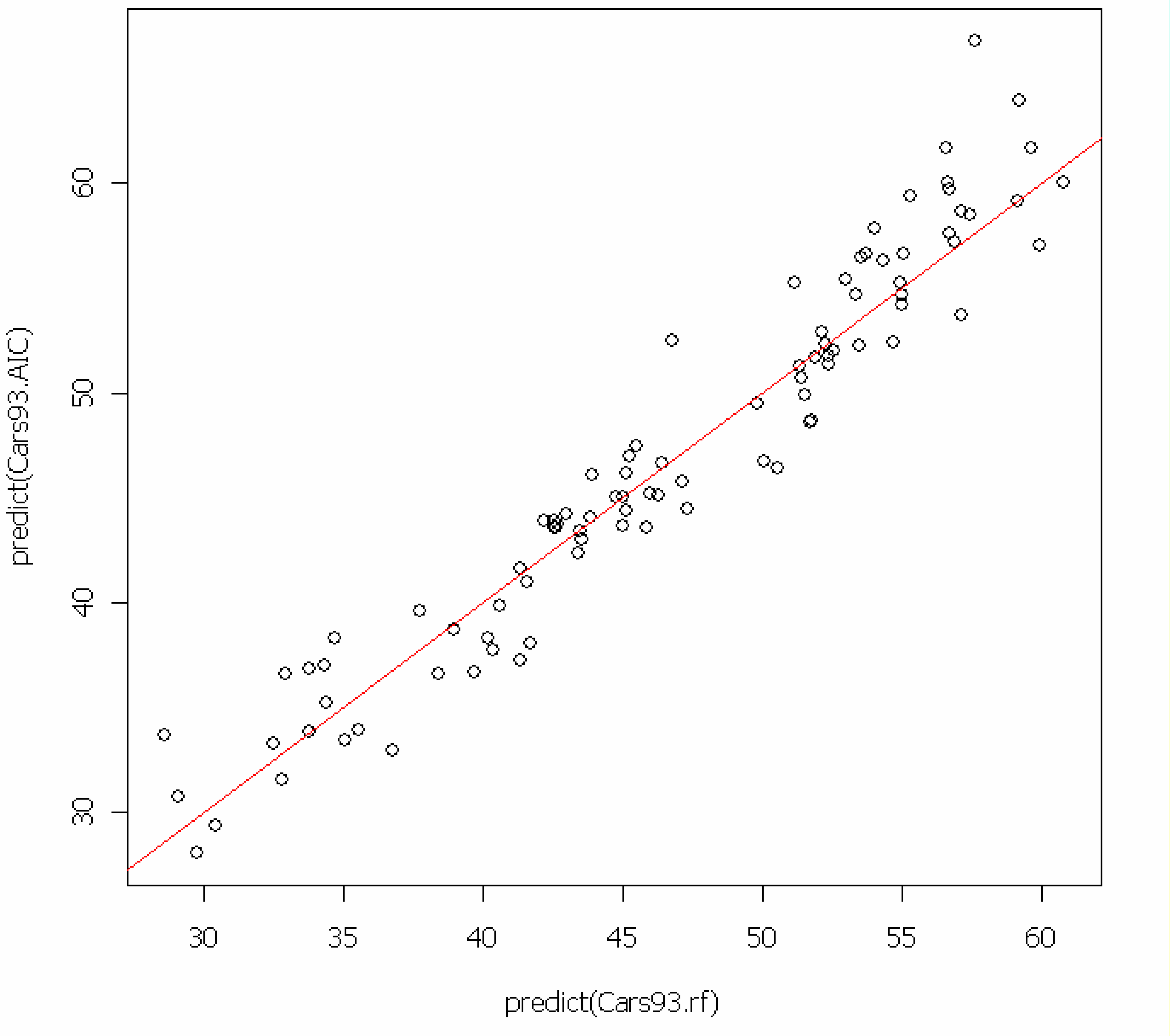

Cylinders=ab
EngineSize< 1.7
Weight< 3615
52.67
59.04
Fuel.tank.capacity< 15.95
32.46
42.08
46.83



predict(Cars93.AIC)
30
40
50
60
35
40
45
50
55
60
predict(Cars93.tm)

Random forests of trees

```
require(randomForest)
Cars93.rf <- randomForest(1000/MPG.city ~ Type +
   Min.Price + Price + Max.Price + AirBags +
   DriveTrain + Cylinders + EngineSize +
   Horsepower + RPM + Rev.per.mile +
   Fuel.tank.capacity + Passengers + Length +
   Wheelbase + Width + Turn.circle + Weight +
   Origin, Cars93)

plot(predict(Cars93.rf), predict(Cars93.AIC))
abline(0, 1, lty = "solid", col = "red")
```



predict(Cars93.AIC)
30
40
50
60
30
35
40
45
50
55
60
predict(Cars93.rf)

Notes

- Tree models can be unstable, but the tree structure is often enlightening and predictions from them can be fairly stable
- Random forests can substantially improve the predictive capacity of tree models, but at the expense of interpretability: a 'black box' predictor
- Really tools from machine learning and data mining, but useful in conjunction with classical models
- More later in the course…