```
my.boot.xy.conf<-
function(mat.train=auto.train.matq,mat.test=auto.test.matq,yind=1,xstring="lars",brep=10000,pred.int=T,alpha=.05){
 #specialized version of bootstrap
#note—xstring will be replaced with leaps and pred.int
   if(xstring=="lars"){
      library(lars) #loads lars library if xstring is initially set as “lars”.
      func0<-lars #function func0 created for lars
      reduce.function<-my.cp.extract.lars1 #calls my.cp.extract.lars1
   }
   if(xstring=="leaps"){
      library(leaps) #loads leaps library if xstring is initially set as “leaps”
      func0<-leaps #function func0 created for leaps
      reduce.function<-my.cp.extract.leaps1#calls my.cp.extract.leaps1
   }
   ypredmat<-NULL #creates a prediction list with zero length
   residmat<-NULL #creates a residual list with zero length
   betamat<-NULL #creates a beta list with zero length
   n1<-length(mat.train[,1]) #defines n1 as the length of training data
   out0<-reduce.function(func0(mat.train[,-yind],mat.train[,yind]),mat.train,mat.test,yind) #defines out1 as the output vector
   ypred0<-c(out0$ypredict0) #defines a yprediction function
   resid0<-c(out0$resid) #defines a residual function
   beta0<-c(out0$beta) #defines a beta function
   ypredmat<-rbind(ypredmat,ypred0) #joins the yprediction matrix to y prediction function outputs by row
   residmat<-rbind(residmat,resid0) #joins the residual matrix to residual function outputs defined above by row
   betamat<-rbind(betamat,beta0) #joins the beta matrix to beta function outputs by row

   for(i in 1:brep){ #run a for loop from 1 to the length of brep
      if((i/200)==floor(i/200)){ #print once every 200 iterations
         print(c(i,brep))
      }
```

```
        v1<-sort(sample(n1,n1,replace=T)) #Sort samples
        m1<-(mat.train[,-yind])[v1,] #Load mat.train minus yind
        y1<-(mat.train[,yind])[v1] #Load mat.train with yind only
        out1<-reduce.function(func0(m1,y1),mat.train,mat.test,yind) #Creates an interval
        ypred1<-c(out1$ypredict0) #Creates a yprediction function
        resid1<-c(out1$resid) #Creates a residual function
        beta1<-c(out1$beta) #Creates a beta function
        ypredmat<-rbind(ypredmat,ypred1) #Merge ypredmat and ypred1
        residmat<-rbind(residmat,resid1) #Merges residmat and resid1
        betamat<-rbind(betamat,beta1) #Merges betamat and beta1

    }
    #Apply function (margin=1: rows, margin=2: columns)
    bagged.pred<-apply(ypredmat,2,mean)
    bagged.beta<-apply(betamat,2,mean)
    quant.boot<-function(x){quantile(x,c(alpha/2,1-alpha/2))}

    if(pred.int){ #if pred.int = T
        main1<-paste("Prediction interval",xstring,"alpha=",alpha) #Concatenate strings and values of xstring and alpha)
        qboot<-apply(ypredmat+residmat,2,quant.boot)
    }
    else{ #if pred.int = F
        main1=paste("Confidence interval,",xstring,"alpha=",alpha) #Concatenate strings and values of xstring and alpha)
        qboot<-apply(ypredmat,2,quant.boot)
    }
    #
    y0<-mat.test[,yind] #Defines y0 as the test data
    plot(rep(bagged.pred,5),c(y0,ypred0,bagged.pred,qboot[1,],qboot[2,]),xlab="Bagged prediction",ylab="Data and
intervals",type="n",main=main1) #Plot bagged prediction
    points(bagged.pred,y0) #Add points to the plot, bagged.pred on x axis and y0 on y axis
    lines(bagged.pred,bagged.pred) #Draws the predicted line in black
    o1<-order(bagged.pred) #Order the predicted line
    lines(bagged.pred[o1],ypred0[o1],col=2) #Draws the trained data in red
    lines(bagged.pred[o1],smooth(qboot[1,o1]),col=3) #Draws the lower bound in green
    lines(bagged.pred[o1],smooth(qboot[2,o1]),col=3) #Draws the upper bound in green

    #plotting all of the bagging prediction lines calculated above
    list(bpred=bagged.pred,ypred0=ypred0,type=xstring,bagged.beta=bagged.beta,orig.beta=beta0,pred.int=pred.int)
}
```

Output:

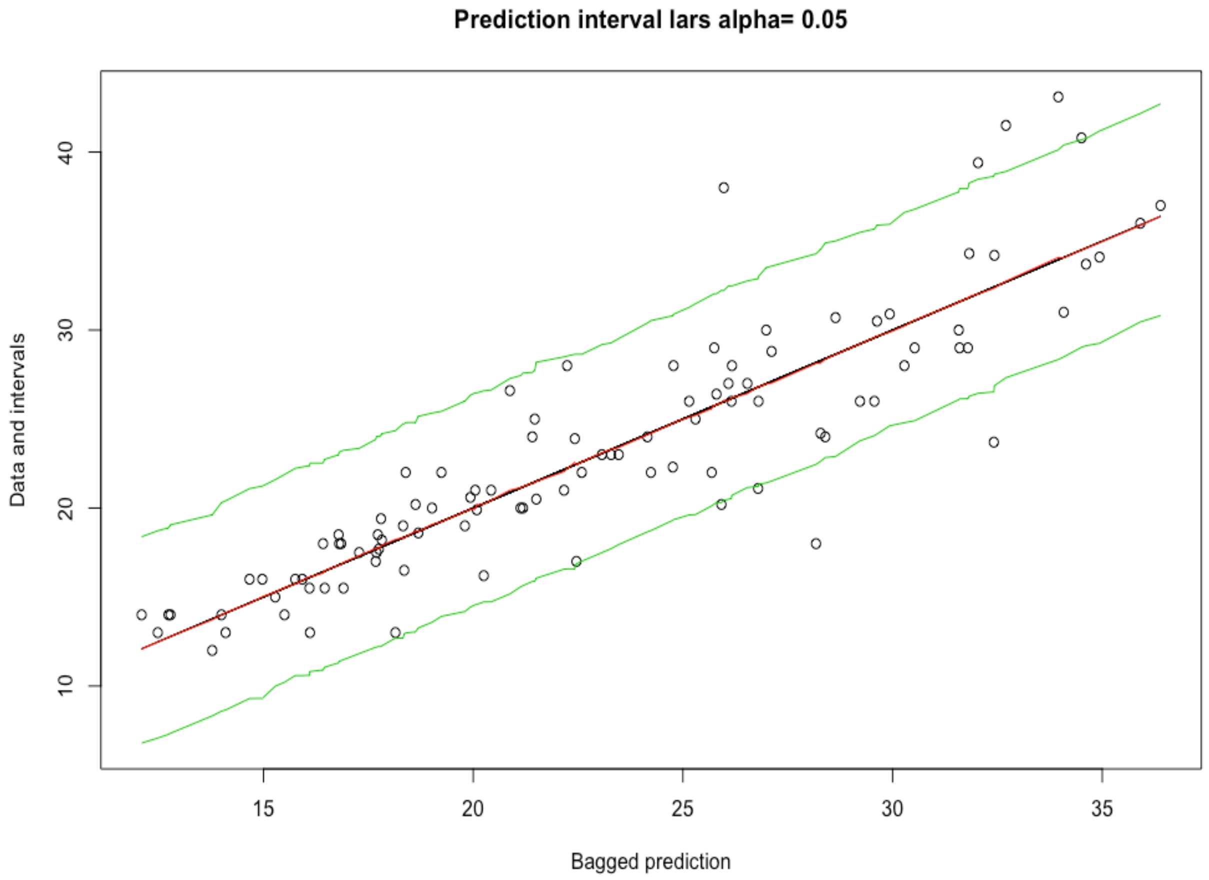
```
> dum.lars<-my.boot.xy.conf(xtring="lars")
> dum.lars

$bpred
      21       25       30       31       38       41       46 
21.46768 20.04791 26.53714 22.23931 16.41992 12.09383 16.85118 
      47       51       52       59       61       66       75 
22.58873 24.77677 26.98629 25.29998 21.12871 12.73430 12.47834 
      79       80       83       85       87       94       96 
20.42771 26.15746 23.46921 26.08652 15.50028 12.78240 13.77750 
      97       99      111      112      114      118      121 
14.09709 15.92483 24.23752 28.17415 22.16633 31.59728 19.80193 
     123      128      139      142      144      145      150 
21.40965 18.32807 13.99661 25.74727 26.80261 34.08113 24.15400 
     151      154      157      162      163      166      167 
25.14930 16.80321 14.66677 15.75639 15.27911 19.01873 18.14501 
```

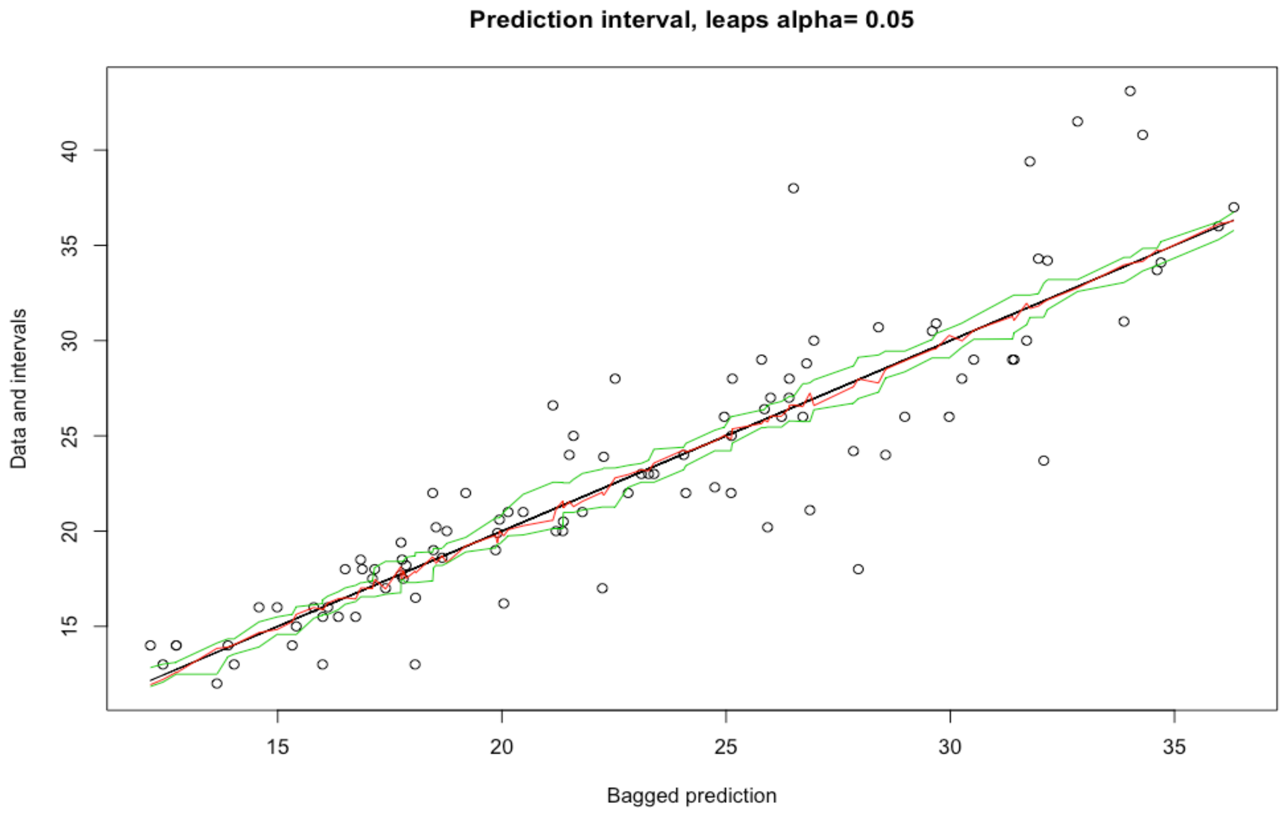
```
     171      176      178      190      192      193      196 
23.06923 30.52602 23.28873 16.90633 19.24202 18.39108 31.79573 
     202      208      212      214      222      223      224 
16.79046 21.18685 18.35472 16.10843 17.28186 17.67540 16.09706 
     226      231      233      236      241      242      245 
17.69286 16.45926 14.97145 29.22281 29.62881 25.68780 33.95156 
     248      250      251      252      254      259      261 
32.03900 20.08626 17.80226 18.62503 21.50448 19.93361 18.69166 
     264      270      271      276      278      283      289 
17.75095 29.93151 26.79345 22.46064 20.25168 24.76073 17.82105 
     293      295      301      302      307      310      314 
17.72592 34.93411 22.42331 32.42842 27.11513 32.70291 26.17054 
     315      318      325      335      343      348      356 
25.80052 31.82844 34.50512 32.41456 31.57744 36.39068 34.61290 
     361      363      365      366      368      374      381 
28.63960 28.28369 20.87537 25.91840 30.28145 28.39566 35.90814 
     387      388 
25.97577 29.56555 

$ypred0
      21       25       30       31       38       41       46 
21.42318 19.96140 26.42933 22.20857 16.34332 12.07277 16.81678 
      47       51       52       59       61       66       75 
22.58682 24.79000 26.96410 25.20895 21.13958 12.72771 12.46560 
      79       80       83       85       87       94       96 
20.43573 26.18006 23.40016 26.02254 15.51526 12.78802 13.72291 
      97       99      111      112      114      118      121 
14.12823 15.93276 24.18793 28.12721 22.04774 31.61594 19.79497 
     123      128      139      142      144      145      150 
21.40744 18.34391 13.98201 25.77724 26.80141 34.05830 24.12104 
     151      154      157      162      163      166      167 
```

```
25.10086 16.79457 14.63281 15.72876 15.28141 19.08168 18.23063
     171      176      178      190      192      193      196
23.09610 30.55349 23.30426 16.94354 19.22396 18.38371 31.78877
     202      208      212      214      222      223      224
16.77206 21.20770 18.29400 16.13706 17.32243 17.72229 16.13456
     226      231      233      236      241      242      245
17.69375 16.49183 15.01154 29.21498 29.65698 25.58915 34.07937
     248      250      251      252      254      259      261
32.02554 20.16961 17.87274 18.69751 21.48967 19.91929 18.67242
     264      270      271      276      278      283      289
17.72310 29.87887 26.81635 22.42998 20.22092 24.75243 17.90319
     293      295      301      302      307      310      314
17.78790 34.90277 22.56390 32.36578 27.02313 32.74473 26.19845
     315      318      325      335      343      348      356
25.79837 31.87364 34.49186 32.35165 31.55256 36.38771 34.59080
     361      363      365      366      368      374      381
28.62752 28.15346 21.00224 25.89457 30.23200 28.36010 35.87864
     387      388
26.03725 29.57650

$type
[1] "lars"

$bagged.beta
    cylinders  displacement    horsepower        weight
 8.712580e-01 -5.499130e-02 -1.586733e-01 -9.587537e-03
 acceleration          year        origin    cylinders2
-2.301177e+00 -7.176849e+00 -1.264356e+00 -2.198890e-02
displacement2   horsepower2       weight2 acceleration2
 5.995468e-05  3.787715e-04  1.155602e-06  6.008333e-02
        year2       origin2
 5.218488e-02  4.734263e-01

$orig.beta
    cylinders  displacement    horsepower        weight 
 5.948313e-01 -5.097367e-02 -1.605702e-01 -9.712061e-03 
 acceleration          year        origin    cylinders2 
-2.279628e+00 -6.961978e+00 -8.843445e-01  0.000000e+00 
displacement2   horsepower2       weight2 acceleration2 
 5.437943e-05  3.871014e-04  1.158954e-06  5.974574e-02 
        year2       origin2 
 5.081065e-02  3.827672e-01 

$pred.int
[1] TRUE
```

```
> dum.leaps<-my.boot.xy.conf(xtring="leaps")
> dum.leaps

$bpred
  [1] 21.59559 20.13557 26.40157 22.51555 16.50069 12.15612
  [7] 17.15265 22.80914 25.13647 26.95912 25.11304 21.34734
 [13] 12.73534 12.44270 20.47746 26.23757 23.39074 25.98601
 [19] 15.33252 12.73693 13.64087 14.02893 16.10626 24.09816
 [25] 27.95067 21.79628 31.43018 19.85926 21.50445 18.46537
 [31] 13.89663 25.79005 26.71755 33.87565 24.05193 24.95697
 [37] 16.88008 14.57775 15.79927 15.40497 18.78037 18.06661
 [43] 23.10591 30.52885 23.27501 16.74295 19.19135 18.45242
 [49] 31.39423 16.84407 21.21170 18.08437 16.00044 17.12029
 [55] 17.41585 16.00769 17.79053 16.35637 14.98241 28.98454
 [61] 29.60200 25.11999 34.02100 31.77441 19.90668 17.75215
 [67] 18.53442 21.36876 19.94249 18.66267 17.74614 29.68583
 [73] 26.86704 22.24964 20.04792 24.74596 17.85934 17.76721
 [79] 34.70234 22.27811 32.17458 26.80236 32.84128 26.40286
 [85] 25.85212 31.96427 34.29076 32.09061 31.70369 36.32141
 [91] 34.61202 28.41198 27.84697 21.12331 25.92462 30.25431
 [97] 28.55306 35.98216 26.49051 29.96752

$ypred0
  [1] 21.28347 20.07285 26.45919 22.79503 16.50010 11.93691
  [7] 17.45892 22.96651 25.36357 26.59104 24.88765 21.58556
 [13] 12.55951 12.21240 20.28224 26.02857 23.57273 26.04000
 [19] 15.24288 12.58627 13.84189 14.03874 16.22467 24.14759
 [25] 27.99119 21.56148 31.06221 19.77133 21.55433 18.59970
 [31] 13.91068 25.64223 26.53412 33.97272 24.25540 25.02507
 [37] 17.01885 14.67327 15.98758 15.62819 18.34586 17.89296
 [43] 23.21468 30.53909 23.16067 16.43354 19.22502 18.64543
 [49] 31.26369 17.00344 21.15152 17.80152 15.88359 16.98189
 [55] 16.96861 15.86166 17.94358 16.45212 14.83379 28.93729
 [61] 29.52902 24.83002 34.05114 31.68191 19.36524 17.51395
 [67] 18.30745 21.21523 19.99560 18.70618 18.13888 29.57392
 [73] 27.25118 22.03934 19.74863 24.72051 17.51389 17.52275
 [79] 34.69412 21.87173 32.13384 26.92150 32.76429 26.59034
 [85] 25.83086 31.81097 34.15499 32.01623 31.96625 36.28956
 [91] 34.75639 27.77470 27.58005 20.57298 25.72529 29.98669
 [97] 28.49876 36.10507 26.62590 30.27473

$type
```

```
[1] "leaps"

$bagged.beta
    Intercept  displacement    horsepower        weight
 351.81191962    0.25092279   -0.10074886   -0.37983216
 acceleration          year   horsepower2       weight2
  -2.27698445   -4.87135531   -2.73534211    0.02239958
acceleration2         year2       origin2
   0.09544773    6.75218566   31.14588181

$orig.beta
    Intercept  displacement    horsepower        weight
 3.438112e+02 -1.466917e-02 -1.562728e-01 -1.280351e-02
 acceleration          year   horsepower2       weight2
-2.453913e+00 -7.674975e+00  3.905395e-04  1.580605e-06
acceleration2         year2       origin2
 6.514618e-02  5.553159e-02  1.969168e-01

$pred.int
[1] TRUE
```


Prediction interval lars alpha= 0.05
Data and intervals
Bagged prediction
10
20
30
40
15
20
25
30
35



Most of the data lies within the prediction interval. As you can see, the interval is not a straight line, it fluctuates between your data points. The lars function is more accurate, because most of the data points lie between the predicator interval. On the other hand, the leaps interval does not consist of most of the points. They lie outside the interval which makes it not accurate. Regardless, the intervals of both graphs are going to fluctuate as they will not be straight lines.