AutosCode

Tyler Poelking, Joey Mack, Xiaoqi Wang

12/4/2017

```
#yolanda
#load("/Users/yolandalala/Desktop/STAT 4620/Project/Autos/train.dat")
#ty
load("~/Desktop/All Stuff/School Stuff/STATS/Project/Autos/test.dat")
load("~/Desktop/All Stuff/School Stuff/STATS/Project/Autos/train.dat")

attach(train)
#summary function
summary(train)
```

```
##              dateCrawled                                name
##  2016-03-23 17:46:12:   2   Audi_A4_Avant_1.9_TDI           :   5
##  2016-03-28 16:57:13:   2   BMW_320i                        :   5
##  2016-03-05 14:09:18:   1   Volkswagen_Passat_Variant_1.9_TDI:   4
##  2016-03-05 14:14:33:   1   BMW_116i                        :   3
##  2016-03-05 14:17:34:   1   BMW_318i                        :   3
##  2016-03-05 14:24:20:   1   Citro<eb>n_C1_1.0_Advance       :   3
##  (Other)            :1992   (Other)                         :1977
##         seller       offerType        price            abtest
##  gewerblich:   0   Angebot:2000   Min.   :     0   control: 968
##  privat    :2000   Gesuch :   0   1st Qu.:  1150   test   :1032
##                                   Median :  2950
##                                   Mean   :  6042
##                                   3rd Qu.:  7500
##                                   Max.   :123456
##
##      vehicleType  yearOfRegistration      gearbox        powerPS
##  limousine :490   Min.   :1934               : 128   Min.   :   0
##  kleinwagen:412   1st Qu.:1999       automatik: 425   1st Qu.:  69
##  kombi     :374   Median :2003       manuell  :1447   Median : 105
##            :220   Mean   :2013                        Mean   : 117
##  bus       :161   3rd Qu.:2009                        3rd Qu.: 150
##  cabrio    :129   Max.   :8888                        Max.   :2331
##  (Other)   :214
##      model         kilometer      monthOfRegistration    fuelType
##  golf    : 158   Min.   :  5000   Min.   : 0.000      benzin :1212
##  andere  : 150   1st Qu.:100000   1st Qu.: 3.000      diesel : 541
##          : 126   Median :150000   Median : 6.000             : 208
##  3er     : 125   Mean   :124370   Mean   : 5.671      lpg    :  30
##  polo    :  71   3rd Qu.:150000   3rd Qu.: 9.000      cng    :   4
##  a4      :  62   Max.   :150000   Max.   :12.000      hybrid :   3
##  (Other):1308                                        (Other):   2
##           brand     notRepairedDamage              dateCreated
##  volkswagen   :436       : 365         2016-04-02 00:00:00:  84
##  bmw          :235   ja  : 181         2016-03-20 00:00:00:  80
##  opel         :203   nein:1454         2016-03-14 00:00:00:  78
##  audi         :192                     2016-03-19 00:00:00:  78
```

```
##  mercedes_benz:188                   2016-03-07 00:00:00:  76
##  ford         :127                   2016-04-03 00:00:00:  76
##  (Other)      :619                   (Other)            :1528
##   nrOfPictures   postalCode                  lastSeen
##  Min.   :0     Min.   : 1069   2016-03-12 21:44:42:   2
##  1st Qu.:0     1st Qu.:28326   2016-03-17 14:45:58:   2
##  Median :0     Median :49514   2016-04-03 16:16:09:   2
##  Mean   :0     Mean   :50491   2016-04-05 13:17:05:   2
##  3rd Qu.:0     3rd Qu.:71588   2016-04-05 14:45:46:   2
##  Max.   :0     Max.   :99974   2016-04-05 15:16:27:   2
##                                (Other)            :1988
```

```
#get data types of each columns
sapply(train, class)
```

```
##         dateCrawled                name              seller
##            "factor"            "factor"            "factor"
##           offerType               price             abtest
##            "factor"           "integer"            "factor"
##         vehicleType  yearOfRegistration             gearbox
##            "factor"           "integer"            "factor"
##             powerPS               model           kilometer
##           "integer"            "factor"           "integer"
## monthOfRegistration             fuelType               brand
##           "integer"            "factor"            "factor"
##   notRepairedDamage         dateCreated        nrOfPictures
##            "factor"            "factor"           "integer"
##          postalCode             lastSeen
##           "integer"            "factor"
```

```
train$dateCrawled <- as.Date(dateCrawled)
```

```
## Warning in strptime(xx, f <- "%Y-%m-%d", tz = "GMT"): unknown timezone
## 'zone/tz/2018c.1.0/zoneinfo/America/New_York'
```

```
train$dateCreated <- as.Date(dateCreated)
train$lastSeen <- as.Date(lastSeen)

test$dateCrawled <- as.Date(dateCrawled)
test$dateCreated <- as.Date(dateCreated)
test$lastSeen <- as.Date(lastSeen)

sapply(train, class)
```

```
##         dateCrawled                name              seller
##              "Date"            "factor"            "factor"
##           offerType               price             abtest
##            "factor"           "integer"            "factor"
##         vehicleType  yearOfRegistration             gearbox
##            "factor"           "integer"            "factor"
##             powerPS               model           kilometer
##           "integer"            "factor"           "integer"
## monthOfRegistration             fuelType               brand
##           "integer"            "factor"            "factor"
##   notRepairedDamage         dateCreated        nrOfPictures
##            "factor"              "Date"           "integer"
```

```
##          postalCode            lastSeen
##           "integer"              "Date"
```

```
#get levels of each factor variable
#table(train$seller) #only have value "privat"
#table(train$offerType) #only have value "angebot"
#table(train$abtest)# "control":968  "test":1032
#table(train$vehicleType) #missing value:220
#table(train$yearOfRegistration)
#table(train$gearbox)#missing value: 128
#table(train$powerPS)
#table(train$model)#missing value:126
#table(train$kilometer) # transfer to categorical data
#table(train$monthOfRegistration)#missing value:205
#tale(train$fuelType)#missing value:208
#table(train$brand)
#table(train$notRepairedDamage)#missing value:365
#table(train$dateCreated)
#table(train$nrOfPictures)#only have value 0
#table(train$postalCode)
#table(train$lastSeen)
```

Covariates that we can safetly remove: The Seller variable since it is always 'privat' and never gewblich. The Type variable since is always 'Angebot' and never gewblich. The nrOfPictures variable since it only has v

Covariates with null values that might need to be handled: -vehicleType -model -monthOfRegistration -fuelType -notRepairedDamage

```
#remove seller, type, and nrOfPictures
train$seller = NULL
train$offerType = NULL
train$nrOfPictures = NULL

test$seller = NULL
test$offerType = NULL
test$nrOfPictures = NULL


#replace the missing value for null
train$gearbox = as.character(train$gearbox)
train$gearbox = replace(train$gearbox, train$gearbox=="", "null")
train$gearbox = as.factor(train$gearbox)

train$vehicleType = as.character(train$vehicleType)
train$vehicleType = replace(train$vehicleType, train$vehicleType=="", "null")
train$vehicleType = as.factor(train$vehicleType)

train$model = as.character(train$model)
train$model = replace(train$model, train$model=="", "null")
train$model = as.factor(train$model)

train$fuelType = as.character(train$fuelType)
train$fuelType = replace(train$fuelType, train$fuelType=="", "null")
train$fuelType = as.factor(train$fuelType)

train$notRepairedDamage = as.character(train$notRepairedDamage)
```

```
train$notRepairedDamage = replace(train$notRepairedDamage, train$notRepairedDamage=="", "maybe")
train$notRepairedDamage = as.factor(train$notRepairedDamage)

#For Test
test$gearbox = as.character(test$gearbox)
test$gearbox = replace(test$gearbox, test$gearbox=="", "null")
test$gearbox = as.factor(test$gearbox)
test$vehicleType = as.character(test$vehicleType)
test$vehicleType = replace(test$vehicleType, test$vehicleType=="", "null")
test$vehicleType = as.factor(test$vehicleType)
test$model = as.character(test$model)
test$model = replace(test$model, test$model=="", "null")
test$model = as.factor(test$model)
test$fuelType = as.character(test$fuelType)
test$fuelType = replace(test$fuelType, test$fuelType=="", "null")
test$fuelType = as.factor(test$fuelType)
test$notRepairedDamage = as.character(test$notRepairedDamage)
test$notRepairedDamage = replace(test$notRepairedDamage, test$notRepairedDamage=="", "maybe")
test$notRepairedDamage = as.factor(test$notRepairedDamage)
```

When is comes to nulls, we replaced blanks with 'null' for most of the factors so they were easily identified in plots. We treated 0's as nulls for monthOfRegistration since month 0 does not exist in real life. Also, we put 'maybe' instead of null for blank notReparedDamage values because we felt this represented reality.

In order to more effectively graph and make inferences from our data we looked into dealing with extreme values for our independent continuous variables as well as our dependent variable price. Typically we don't want to change the variable we are trying to predict, but in our case, since we believe that the values found don't actually reflect reality, we changed them, essentially treating them like nulls. This not-representative-of-reality characteristic is typical of data inputted by human. Discovery of outliers was done both by sorting and empirically observing the data as well as through visuals we'd make that had points far from the norm.

Our first step was removing rows in the training that contained prices that were obviously synthesized/inaccurate. Sorting the train data in descending order, we discovered a row that had the extremely high price of '123456'. The price was most likely entered by the user posting it as a filler and the year 8888 obviously doesn't exist.

In the test dataset there were a few instances extremely high prices (more than an order of two magnitudes greater than the rest of the dataset's prices). Including these values will cause evaluation metrics such as the MSE to be so inflated that it becomes nearly impossible to compare models. These prices are also most likely synthesized as a product of the person posting with the hopes of the viewer making the first move and initialize a bartering process. We therefore removed these instances where price was 1234566, 27322222, and 99999999 in the test data.

For these values where price is 1 or less we removed them from both the test and the training data. We did this because people who posted prices as low as this were more than likely trying to start a barter process online. So instead of actually suggesting a price themselves, they put a ludicrous value like 0 or 1 and have buyer come to them with their price offers. The prices therefore do not actually represent the price of the car, which is what we are trying to predict.

```
#Handle the outliler for price
train = train[train$price!=123456,]
train = train[train$price!=0,]
train = train[train$price!=1,]

#remove outlier prices for test so we can effectively compare models without inflated error metrics
test = test[test$price!=99999999,]
```

```
test = test[test$price!=27322222,]
test = test[test$price!=1234566,]
test = test[test$price!=0,]
test = test[test$price!=1,]
```

We did a similar process for year of registration, discovering values of 7777, 8888 in the training set and a value of 1500 in the testing. We left the value of 1500 in the testing because we did not want to change the test set too much. However, we removed the extreme valeus of 7777 and 8888 in the test set, since these values obviously weren't entered in with serious intent. We removed these years instead of filling them in (similar to how you fill in nulls) because there were only two rows to deal with and there was no obvious strong predictor for year.

```
#Remove outliers in year of registration

row = which(train$yearOfRegistration > 2018)
train = train[-row,]

row = which(test$yearOfRegistration < 1600)
test = test[-row,]
```

Next, we changed powerPS values that are above 850 because very few production cars have horsepower above that and we noticed our dataset had powerPS that was quit unreasonable on both ends (well above 800 and below 10). We changed the 0's because no car actually has a 0 horsepower and most people probably filled in 0 because they didn't know the specific horsepower of their vehicle. For these values we set the powerPS based on the mean powerPS of the given car's vehicle type (including nulls as a factor).

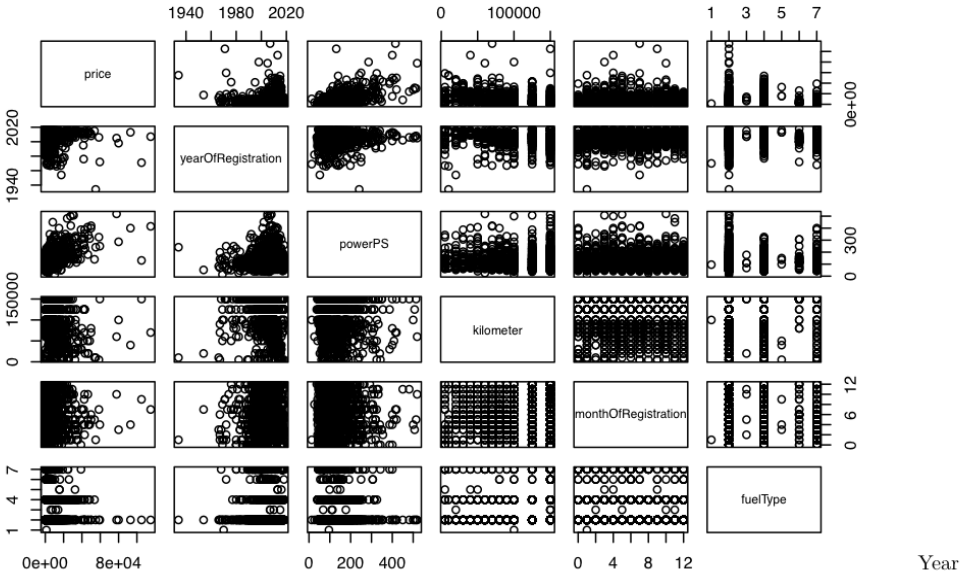
```
#Handle the outlier for powerPS
mean_vehicle = aggregate(train$powerPS, list(train$vehicleType), mean)
for(i in levels(train$vehicleType)){
  row = which(mean_vehicle$Group.1 == i)
  mean = mean_vehicle[row,2]
  train = within(train, powerPS[ powerPS<=10 & vehicleType==i] <- mean)
  train = within(train, powerPS[ powerPS>=850 & vehicleType==i] <- mean)
}


for(i in levels(train$vehicleType)){
  row = which(mean_vehicle$Group.1 == i)
  mean = mean_vehicle[row,2]
  test = within(test, powerPS[ powerPS<=10 & vehicleType==i] <- mean)
  test = within(test, powerPS[ powerPS>=850 & vehicleType==i] <- mean)
}
```

```
#obtain integer (continuous) variables
s.fact <- sapply(train, is.integer)
s.fact
```

```
##         dateCrawled                name               price
##               FALSE               FALSE                TRUE
##              abtest         vehicleType  yearOfRegistration
##               FALSE               FALSE                TRUE
##             gearbox             powerPS               model
##               FALSE               FALSE               FALSE
##           kilometer monthOfRegistration            fuelType
##                TRUE                TRUE               FALSE
##               brand   notRepairedDamage         dateCreated
```

```
##              FALSE              FALSE              FALSE
##         postalCode            lastSeen
##               TRUE              FALSE
```

```
cont = c('price', 'yearOfRegistration', 'powerPS', 'kilometer','monthOfRegistration', 'fuelType')
plot(train[cont])
```



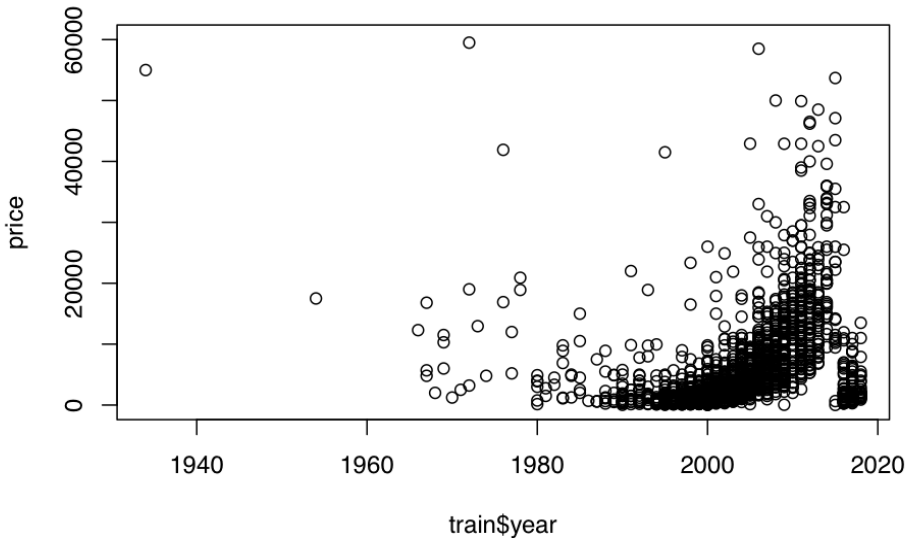
seemed to play a very significant roll in predicting price. The relationship seems non-linear, and as year increases, the effect it has on price increases as well (similar to a quadratic relationship). We will explore this relationship in more detail below.

PowerPS also seems like a decent predictor of price. This relationship is more linear in nature. Also, the collinearity between Year of registraion and powerPS seems minutely present, but it is nothing prominent.

The rest of the variables in the scatterplot above act more categorical in nature, since their values aren't uniformily distributed and instead jump in equal increments. We will visualize/explore these variables and their effects via other methods.

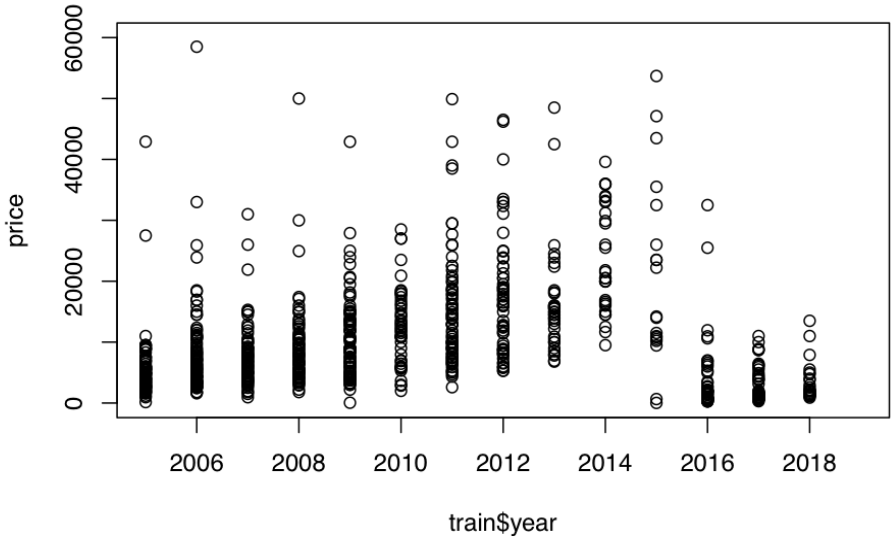
Lets further explore YearOfRegistration to get a better understanding of the form of its relationship price. Especially want to look at years post 2016 since this is when the data actually was released.

```
#all cars
plot(train$price~train$year,ylab="price",ylim=c(0,60000))
```

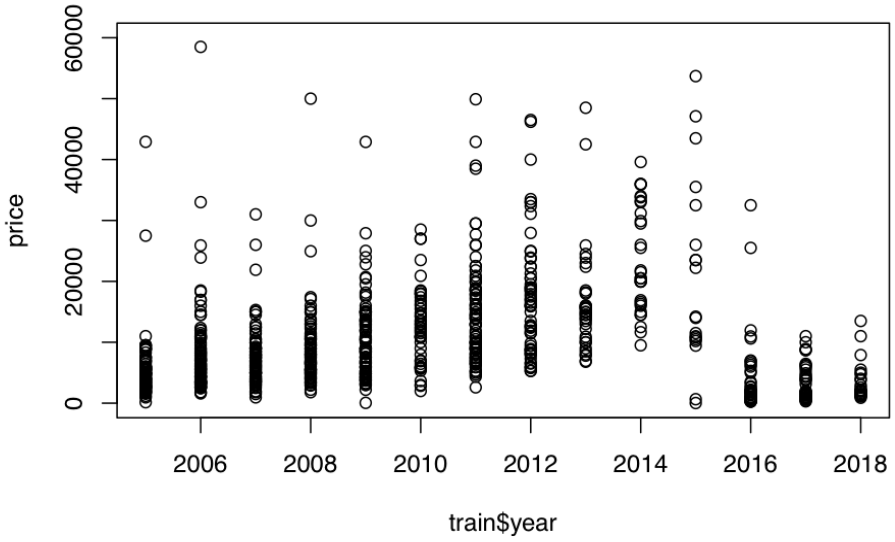



```
#younger cars
plot(train$price~train$year,ylab="price",ylim=c(0,60000), xlim=c(1950, 2010))
```


```
#prior 2015
plot(train$price~train$year,ylab="price",ylim=c(0,60000), xlim=c(2005, 2019))
```

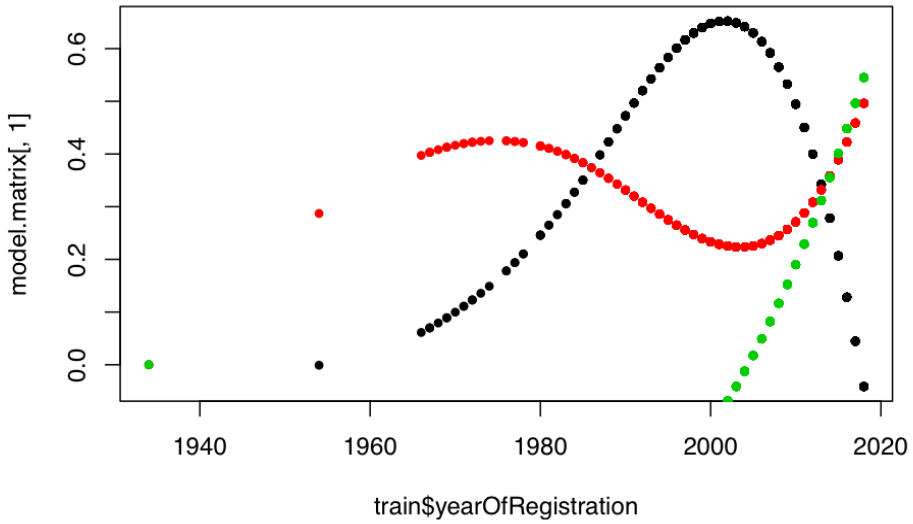



```
#post 2016
plot(train$price~train$year,ylab="price",ylim=c(0,60000), xlim=c(2005, 2018))
```



There seems to be a polynomial relationship between years and price for year prior to 2015 There is definitely a peak around 2013/2014. We may want to

```
model.matrix=ns(train$yearOfRegistration,knots=c(1985,2015)) # create model matrix for cubic splines us
plot(train$yearOfRegistration,model.matrix[,1],pch=20,col=1)
points(train$yearOfRegistration,model.matrix[,2],pch=20,col=2)
points(train$yearOfRegistration,model.matrix[,3],pch=20,col=3)
```




```
fit.ns = lm(price~ns(yearOfRegistration,knots=c(1985,2015)),data=train)
summary(fit.ns)
```

```
##
## Call:
## lm(formula = price ~ ns(yearOfRegistration, knots = c(1985, 2015)),
##     data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -27314  -3164  -1683    933 107113
##
## Coefficients:
##                                             Estimate Std. Error t value
## (Intercept)                                    82314       5971   13.79
## ns(yearOfRegistration, knots = c(1985, 2015))1   -66096       5192  -12.73
## ns(yearOfRegistration, knots = c(1985, 2015))2  -150708      11972  -12.59
## ns(yearOfRegistration, knots = c(1985, 2015))3     3767       1350    2.79
##                                             Pr(>|t|)
## (Intercept)                                  < 2e-16 ***
## ns(yearOfRegistration, knots = c(1985, 2015))1  < 2e-16 ***
## ns(yearOfRegistration, knots = c(1985, 2015))2  < 2e-16 ***
## ns(yearOfRegistration, knots = c(1985, 2015))3  0.00532 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7989 on 1926 degrees of freedom
## Multiple R-squared:  0.1869, Adjusted R-squared:  0.1856
## F-statistic: 147.6 on 3 and 1926 DF,  p-value: < 2.2e-16
```

```
attach(train)
```

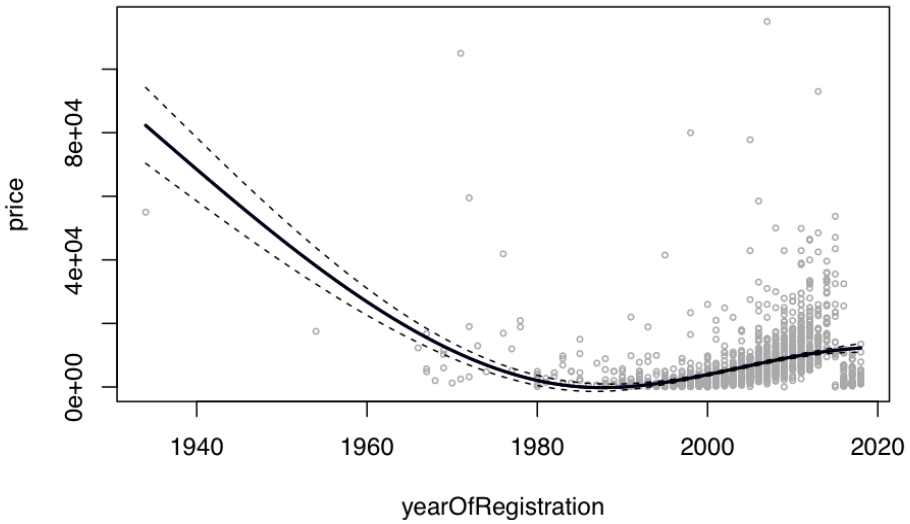
```
## The following objects are masked from train (pos = 3):
```

```
##
##     abtest, brand, dateCrawled, dateCreated, fuelType, gearbox,
##     kilometer, lastSeen, model, monthOfRegistration, name,
##     notRepairedDamage, postalCode, powerPS, price, vehicleType,
##     yearOfRegistration
```

```
yearlims =range(yearOfRegistration)
year.grid=seq(from=yearlims [1], to=yearlims [2])
print(length(year.grid))
```

```
## [1] 85
```

```
preds=predict(fit.ns,data.frame(yearOfRegistration=year.grid), se=T)
plot(yearOfRegistration,price,xlim=yearlims ,cex =.5, col =" darkgrey ")
#print(length(preds$fit))
lines(year.grid,preds$fit ,lwd =2, col =" blue")
lines(year.grid,preds$fit,lwd=2)
lines(year.grid,preds$fit-2*preds$se,lty=2)
lines(year.grid,preds$fit+2*preds$se,lty=2)
```




```
fit.lr=loess(price~yearOfRegistration,span=0.1,data=train)
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 2003
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 1
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 1
```

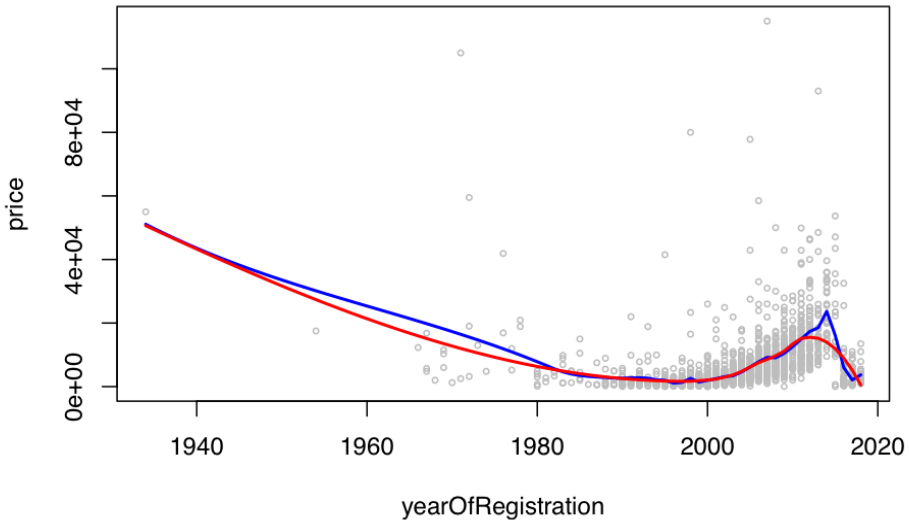
```
summary(fit.lr)
```

```
## Call:
## loess(formula = price ~ yearOfRegistration, data = train, span = 0.1)
##
## Number of Observations: 1930
## Equivalent Number of Parameters: 24.38
## Residual Standard Error: 7037
## Trace of smoother matrix: 26.96  (exact)
##
## Control settings:
##   span     :  0.1
##   degree   :  2
##   family   :  gaussian
##   surface  :  interpolate      cell = 0.2
##   normalize:  TRUE
##  parametric:  FALSE
## drop.square:  FALSE
```

```
fit.lr2=loess(price~yearOfRegistration,span=0.5,data=train)
summary(fit.lr2)
```

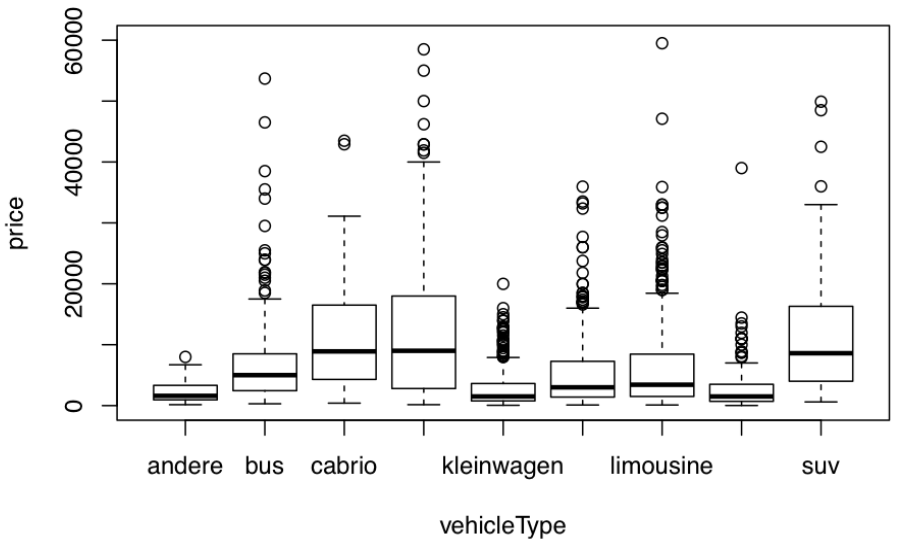
```
## Call:
## loess(formula = price ~ yearOfRegistration, data = train, span = 0.5)
##
## Number of Observations: 1930
## Equivalent Number of Parameters: 7.7
## Residual Standard Error: 7249
## Trace of smoother matrix: 8.48  (exact)
##
## Control settings:
##   span     :  0.5
##   degree   :  2
##   family   :  gaussian
##   surface  :  interpolate      cell = 0.2
##   normalize:  TRUE
##  parametric:  FALSE
## drop.square:  FALSE
```

```
plot(yearOfRegistration,price,xlim=range(yearOfRegistration),cex=0.5,col="grey")
pred.lr=predict(fit.lr,data.frame(yearOfRegistration=year.grid))
lines(year.grid,pred.lr,col="blue",lwd=2)
pred.lr2=predict(fit.lr2,data.frame(yearOfRegistration=year.grid))
lines(year.grid,pred.lr2,col="red",lwd=2)
```

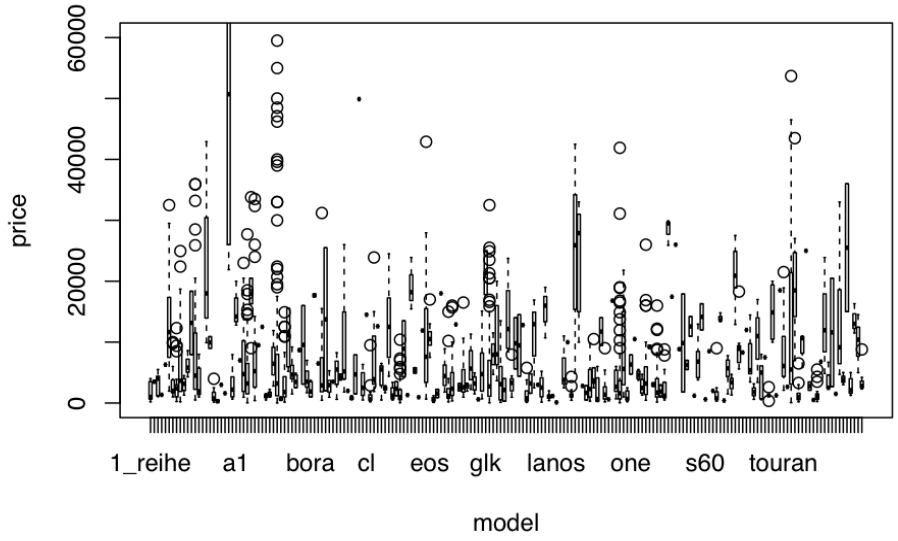


A spline with span = 0.1 fit to year nicely, since it caputured the sudden change of slope experienced after 2015. Though its line might have been a little higher for earlier years than we preffered, it still fits to price well for the majority of the data (at and around 2000).

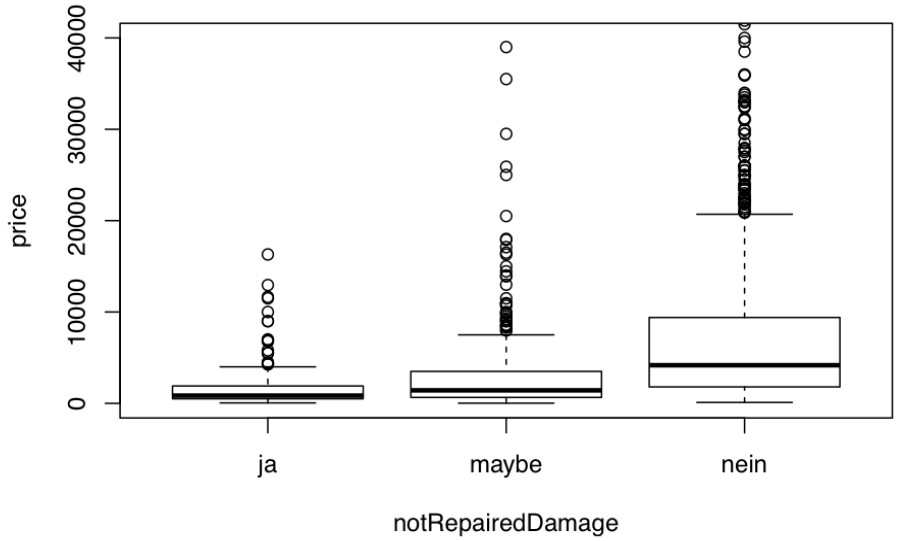
```
#Make the boxplot for variable with missing value
plot(train$price~train$vehicleType,xlab="vehicleType",ylab="price",ylim=c(0,60000))#null is not signifi
```



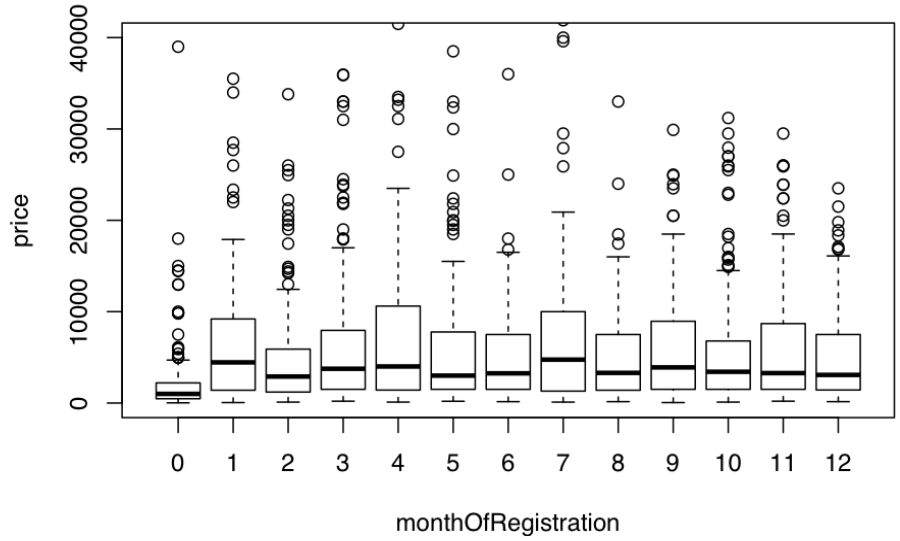

```
plot(train$price~train$model,xlab="model",ylab="price",ylim=c(0,60000))
```



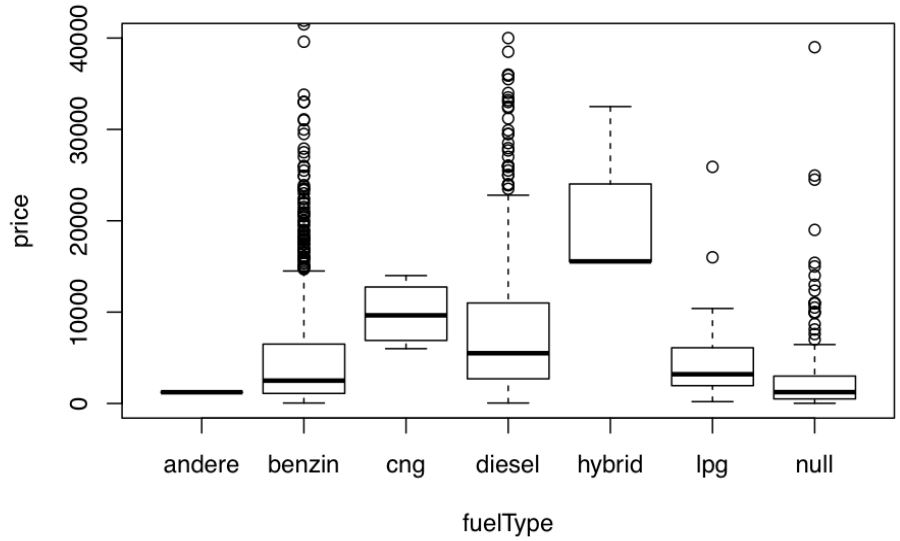

```
plot(train$price~train$notRepairedDamage,xlab="notRepairedDamage",ylab="price",ylim=c(0,40000))#the mea
```



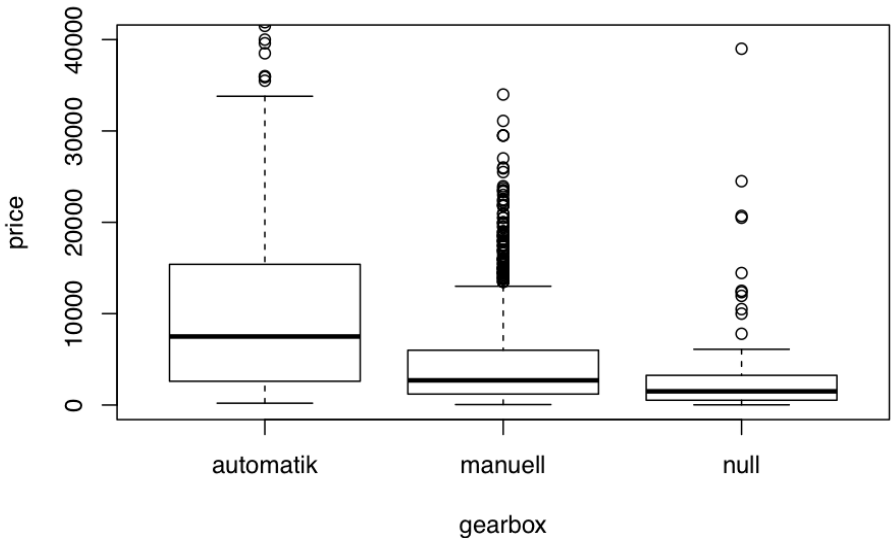

```
plot(train$price~as.factor(train$monthOfRegistration),xlab="monthOfRegistration",ylab="price",ylim=c(0,
```




```
boxplot(train$price~train$fuelType,xlab="fuelType",ylab="price",ylim=c(0,40000)) #Only have two observa
```




```
plot(train$price~train$gearbox,xlab="gearbox",ylab="price",ylim=c(0,40000))# "null" looks different fro
```



Model

Dealing with model more difficult than normal because there were 231 different levels that resulted in data that was spread thin across the levels. Plotting the data formed in visuals that are too granular to interpret and no obvious clusters arose. Therefore, in order to find which levels were most important in determining price we utilized lasso regression. We created a singular lasso model using model as the predictor for price with lambda cross validated. The cross validated lambda resulted in around 13 model factors not being set to zero. This however was too detailed since some of those models were not present in the test dataset. We increased lambda until only 7 model factors were not zero in the model. Printing the coefficients allows us to extract the resulting model factor levels that had the highest significance.

```
set.seed(42)
#exploring which model levels are significant utilizing lasso regression

modelsAll = intersect(levels(test$model),levels(train$model))
train_temp= subset(train, model %in% modelsAll)

xfactors <- model.matrix(train_temp$price ~ train_temp$model)[, -1]
x        <- as.matrix(data.frame(xfactors))
lasso.cv = cv.glmnet(y=train_temp$price,x=x,alpha=1)
min_lam = lasso.cv$lambda.min
min_lam = min_lam + 700

lasso.tight=glmnet(y=train_temp$price,x=x,alpha=1,lambda=min_lam)

coef(lasso.tight)
```

```
## 192 x 1 sparse Matrix of class "dgCMatrix"
##                                      s0
## (Intercept)                5748.4074
## train_temp.model100           .
## train_temp.model147           .
## train_temp.model156           .
## train_temp.model159           .
## train_temp.model1er        1321.8385
```

```
## train_temp.model2_reihe         .
## train_temp.model3_reihe         .
## train_temp.model3er             .
## train_temp.model4_reihe         .
## train_temp.model5_reihe         .
## train_temp.model500             .
## train_temp.model5er             .
## train_temp.model6_reihe         .
## train_temp.model601             .
## train_temp.model6er             .
## train_temp.model7er             .
## train_temp.model80              .
## train_temp.model850             .
## train_temp.model90              .
## train_temp.model900             .
## train_temp.model911         44778.1992
## train_temp.modela_klasse        .
## train_temp.modela1              .
## train_temp.modela2              .
## train_temp.modela3              .
## train_temp.modela4              .
## train_temp.modela5              .
## train_temp.modela6              .
## train_temp.modela8              .
## train_temp.modelaccord          .
## train_temp.modelagila           .
## train_temp.modelalmera          .
## train_temp.modelaltea           .
## train_temp.modelandere          .
## train_temp.modelarosa           .
## train_temp.modelastra           .
## train_temp.modelauris           .
## train_temp.modelavensis         .
## train_temp.modelaygo            .
## train_temp.modelb_klasse        .
## train_temp.modelbeetle          .
## train_temp.modelberlingo        .
## train_temp.modelbora            .
## train_temp.modelboxster         .
## train_temp.modelbravo           .
## train_temp.modelc_klasse        .
## train_temp.modelc_max           .
## train_temp.modelc1              .
## train_temp.modelc2              .
## train_temp.modelc3              .
## train_temp.modelc5              .
## train_temp.modelcaddy           .
## train_temp.modelcalibra         .
## train_temp.modelcarisma         .
## train_temp.modelcarnival        .
## train_temp.modelcayenne         .
## train_temp.modelcivic           .
## train_temp.modelcl              .
## train_temp.modelclio            .
```

```
## train_temp.modelclk            .
## train_temp.modelclubman        .
## train_temp.modelcolt           .
## train_temp.modelcombo          .
## train_temp.modelcooper         .
## train_temp.modelcordoba        .
## train_temp.modelcorolla        .
## train_temp.modelcorsa          .
## train_temp.modelcr_reihe       .
## train_temp.modelcuore          .
## train_temp.modelcx_reihe       .
## train_temp.modeldoblo          .
## train_temp.modelducato         .
## train_temp.modelduster         .
## train_temp.modele_klasse       .
## train_temp.modeleos            .
## train_temp.modelescort         .
## train_temp.modelespace         .
## train_temp.modelexeo           .
## train_temp.modelfabia          .
## train_temp.modelfiesta         .
## train_temp.modelfocus          .
## train_temp.modelforester       .
## train_temp.modelforfour        .
## train_temp.modelfortwo         .
## train_temp.modelfox            .
## train_temp.modelfreelander     .
## train_temp.modelfusion         .
## train_temp.modelgalant         .
## train_temp.modelgalaxy         .
## train_temp.modelglk            .
## train_temp.modelgolf           .
## train_temp.modelgrand          .
## train_temp.modeli_reihe        .
## train_temp.modelibiza          .
## train_temp.modelimpreza        .
## train_temp.modelinsignia       .
## train_temp.modeljazz           .
## train_temp.modeljetta          .
## train_temp.modeljimny          .
## train_temp.modeljuke           .
## train_temp.modelka             .
## train_temp.modelkadett         .
## train_temp.modelkaefer         .
## train_temp.modelkalina         .
## train_temp.modelkangoo         .
## train_temp.modelkuga           .
## train_temp.modellaguna         .
## train_temp.modellancer         .
## train_temp.modellanos          .
## train_temp.modellegacy         .
## train_temp.modelleon           .
## train_temp.modellogan          .
## train_temp.modellupo           .
```

```
## train_temp.modelm_klasse       32.4689
## train_temp.modelm_reihe          .
## train_temp.modelmateria          .
## train_temp.modelmatiz            .
## train_temp.modelmegane           .
## train_temp.modelmeriva           .
## train_temp.modelmicra            .
## train_temp.modelmii              .
## train_temp.modelmondeo           .
## train_temp.modelmove             .
## train_temp.modelmustang          .
## train_temp.modelmx_reihe         .
## train_temp.modelnull             .
## train_temp.modeloctavia          .
## train_temp.modelomega            .
## train_temp.modelone              .
## train_temp.modeloutlander        .
## train_temp.modelpajero           .
## train_temp.modelpanda            .
## train_temp.modelpassat           .
## train_temp.modelphaeton          .
## train_temp.modelpicanto          .
## train_temp.modelpolo             .
## train_temp.modelprimera          .
## train_temp.modelpunto            .
## train_temp.modelq3               .
## train_temp.modelq5               .
## train_temp.modelq7               .
## train_temp.modelqashqai          .
## train_temp.modelrav              .
## train_temp.modelroadster         .
## train_temp.modelroomster         .
## train_temp.modelrx_reihe         .
## train_temp.models_klasse         .
## train_temp.models_max            .
## train_temp.models60              .
## train_temp.modelsandero          .
## train_temp.modelsanta            .
## train_temp.modelscenic           .
## train_temp.modelscirocco         .
## train_temp.modelseicento         .
## train_temp.modelsharan           .
## train_temp.modelsignum           .
## train_temp.modelsl             844.0845
## train_temp.modelslk              .
## train_temp.modelsorento          .
## train_temp.modelspider           .
## train_temp.modelsprinter         .
## train_temp.modelstilo            .
## train_temp.modelsuperb           .
## train_temp.modelswift            .
## train_temp.modelterios           .
## train_temp.modeltigra            .
## train_temp.modeltiguan           .
```

```
## train_temp.modeltoledo          .
## train_temp.modeltouareg         .
## train_temp.modeltouran          .
## train_temp.modeltransit         .
## train_temp.modeltransporter  1327.7113
## train_temp.modeltt           3031.6461
## train_temp.modeltwingo          .
## train_temp.modelup              .
## train_temp.modelv_klasse        .
## train_temp.modelv40             .
## train_temp.modelv70             .
## train_temp.modelvectra          .
## train_temp.modelverso           .
## train_temp.modelviano           .
## train_temp.modelvito            .
## train_temp.modelvivaro          .
## train_temp.modelvoyager         .
## train_temp.modelx_reihe         .
## train_temp.modelx_trail         .
## train_temp.modelxc_reihe        .
## train_temp.modelyaris           .
## train_temp.modelyeti            .
## train_temp.modelz_reihe         .
## train_temp.modelzafira          .
```

```
train$newModel = ifelse(train$model == "1er", "1er",ifelse(train$model == "911", "911",ifelse(train$mod

test$newModel = ifelse(test$model == "1er", "1er",ifelse(test$model == "911", "911",ifelse(test$model =
```

notReparedDamage notReparedDamage may be significant in determing price, though there is fair overlap between each of the factor level's boxes. 'Maybe' seemed closly aligned with ja and might be a result of people who do have damage on their car being hesistant to mention it.

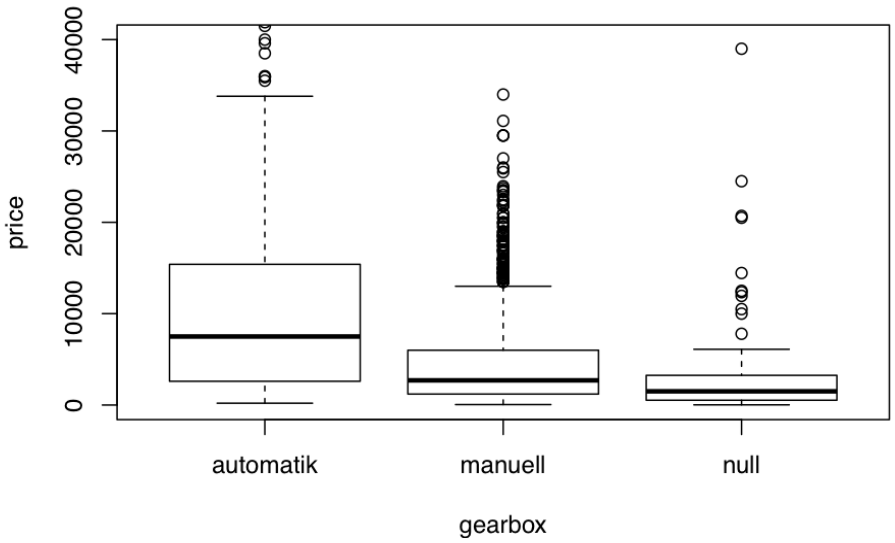
MonthOfRegistration There did not seem to be much difference between that various month of registrations, except for the face that monthOfRegistration = 0 was lower than the rest. Month 0 obviously does not exist, so these people probably did not provide a value when filling out the form. Perhaps we can say that these sort of 'lazy' people tended to sell less expensive cars? Though that may be a stretch.

FuelType For fuel type benzin(standard gas) and diesel represent almost 88 percent of the cars. Null represent about 10 percent and the rest make up less than 2 percent. Based off of the size of these groups and their box plots for price, we decided to make three buckets benzin, diesel, and other.

```
#Combine the level of fuelType to "diesel" "benzin" and "others"
train$fuelType = ifelse(train$fuelType == "benzin", "benzin", ifelse(train$fuelType=="diesel","diesel",

#for test
test$fuelType = ifelse(test$fuelType == "benzin", "benzin", ifelse(test$fuelType=="diesel","diesel","otl
```

Gearbox To deal with the null values under the gearbox categorical variable, we filled in all nulls to be 'manual'. We did this because the majority of instances in the data are manual and the null boxplot looks very similar to the manual box plot (see figure in between price and gearbox).

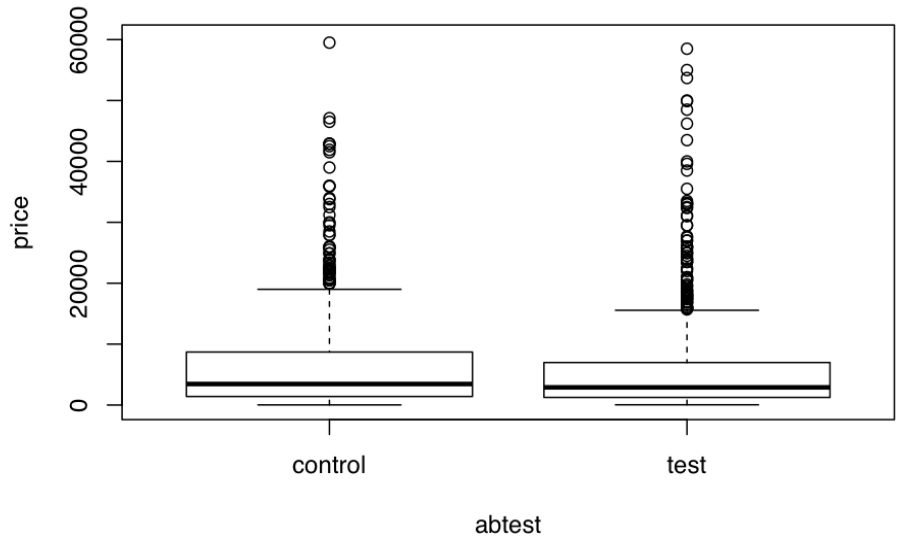
```
#Replace the gearbox's missing value to manuel
plot(train$price~train$gearbox,xlab="gearbox",ylab="price",ylim=c(0,40000))# "null" looks different fro
```



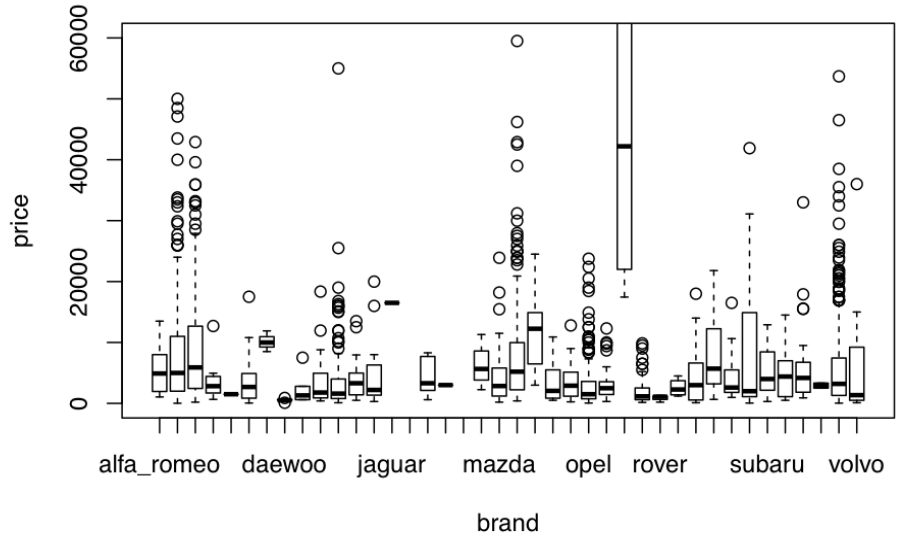

```
train$gearbox = as.character(train$gearbox)
train$gearbox = replace(train$gearbox, train$gearbox=="null", "manuell")
train$gearbox = as.factor(train$gearbox)

#for test
test$gearbox = as.character(test$gearbox)
test$gearbox = replace(test$gearbox, test$gearbox=="null", "manuell")
test$gearbox = as.factor(test$gearbox)
```

We continued to create boxplots for the other factor variables.

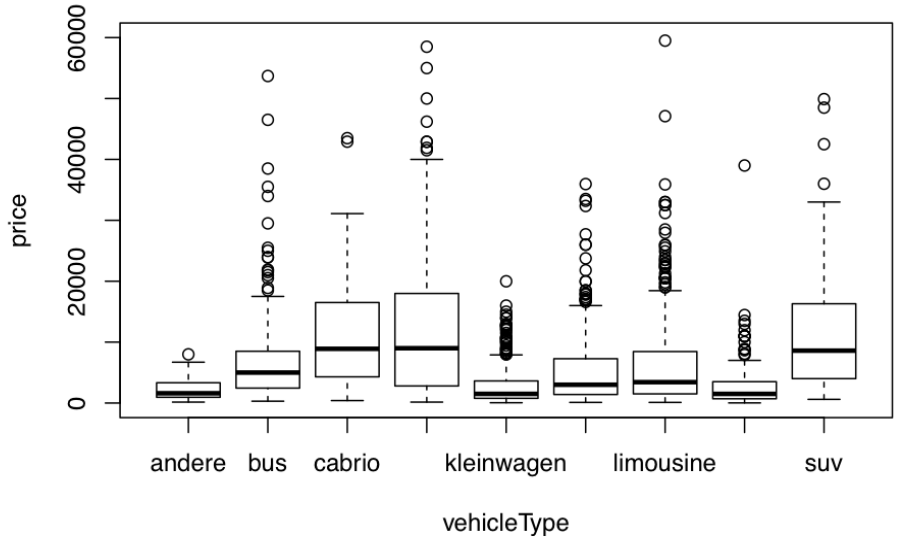
```
#Make the boxplot for variable with missing value

plot(train$price~train$abtest,xlab="abtest",ylab="price",ylim=c(0,60000))
```



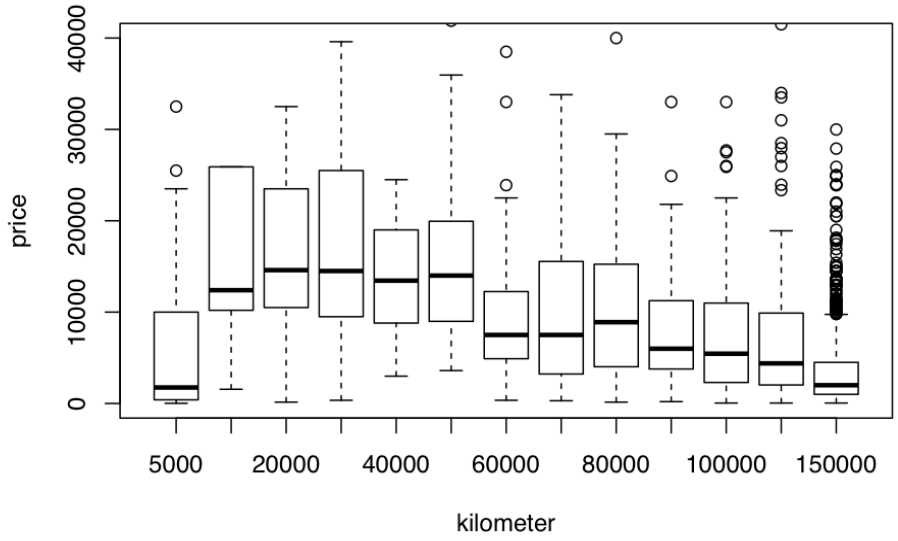

```
plot(train$price~train$brand,xlab="brand",ylab="price",ylim=c(0,60000))
```



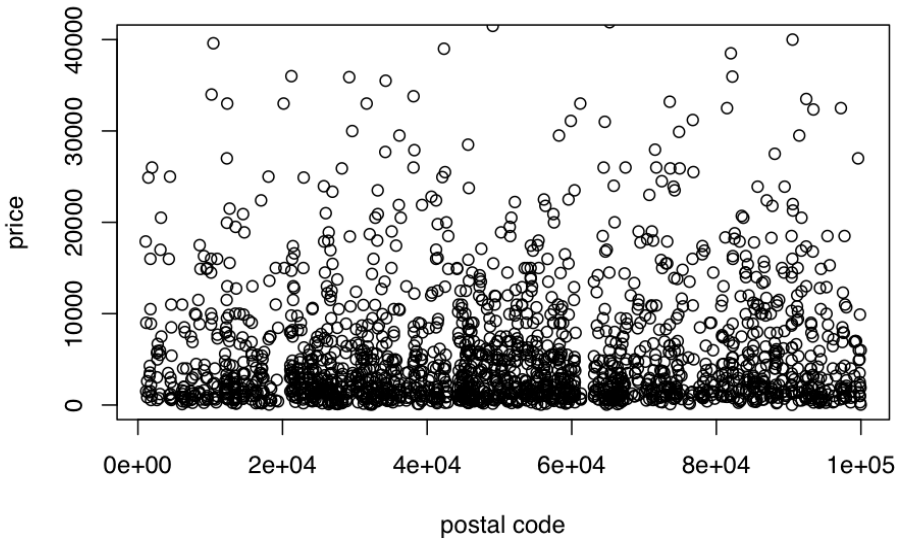

```
plot(train$price~train$vehicleType,xlab="vehicleType",ylab="price",ylim=c(0,60000))
```




```
plot(train$price~as.factor(train$kilometer),xlab="kilometer",ylab="price",ylim=c(0,40000))
```




```
plot(train$price~train$postalCode,xlab="postal code",ylab="price",ylim=c(0,40000))
```

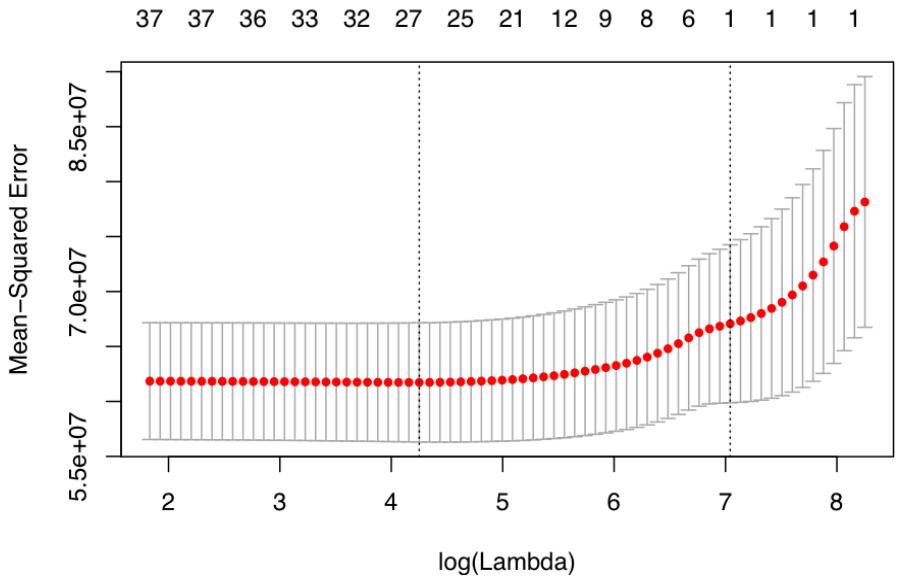


Abtest

This feature did not seem to help in determining price since the two boxplots overlapped so much.

Brand Like model, brand was more difficult because there were around 40 different levels that resulted in data that was spread thin across the levels. To avoid introducing too many predictors we want to generilize the information contained in brand. Again, to find which levels were most important in determining price we utilized lasso regression. We created a singular lasso model using brand as the predictor for price with lambda cross validated.

```
#exploring which model levels are significant utilizing lasso regression
xfactors <- model.matrix(train$price ~ train$brand)[, -1]
x        <- as.matrix(data.frame(xfactors))
lasso.cv = cv.glmnet(y=train$price,x=x,alpha=1)
plot(lasso.cv)
```




```
coef(lasso.cv)
```

```
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                      1
## (Intercept)                5962.887
## train.brandaudi               .
## train.brandbmw                .
## train.brandchevrolet          .
## train.brandchrysler           .
## train.brandcitroen            .
## train.branddacia              .
## train.branddaewoo             .
## train.branddaihatsu           .
## train.brandfiat               .
## train.brandford               .
## train.brandhonda              .
## train.brandhyundai            .
## train.brandjaguar             .
## train.brandjeep               .
## train.brandkia                .
## train.brandlada               .
## train.brandlancia             .
## train.brandland_rover         .
## train.brandmazda              .
## train.brandmercedes_benz      .
## train.brandmini               .
## train.brandmitsubishi         .
## train.brandnissan             .
## train.brandopel               .
## train.brandpeugeot            .
## train.brandporsche        31688.156
```

```
## train.brandrenault            .
## train.brandrover              .
## train.brandsaab               .
## train.brandseat               .
## train.brandskoda              .
## train.brandsmart              .
## train.brandsonstige_autos     .
## train.brandsubaru             .
## train.brandsuzuki             .
## train.brandtoyota             .
## train.brandtrabant            .
## train.brandvolkswagen         .
## train.brandvolvo              .
```

This resulted in brandaudi, brandbmw, brandopel, and brandporsche rising to the top as the levels that did not get set to zero.

```
train$newBrand = ifelse(train$brand == "audi", "audi",ifelse(train$brand == "bmw", "bmw", ifelse(train$

#for test
test$newBrand = ifelse(test$brand == "audi", "audi",ifelse(test$brand == "bmw", "bmw", ifelse(test$bran
```

VehicleType This feature had a fair amount of spread between the different categories. After empirically observing the data, making some google translate searches, and brainstorming, we decided to bucket these into three groups with three vehicleTypes encompassed in each, defined as Compact, Luxury, and Family. See the r code below for mappings.

```
train$carType = ifelse(train$vehicleType == "andere", "Compact", ifelse(train$vehicleType == "bus", "Fa

#for test
test$carType = ifelse(test$vehicleType == "andere", "Compact", ifelse(test$vehicleType == "bus", "Famil
```

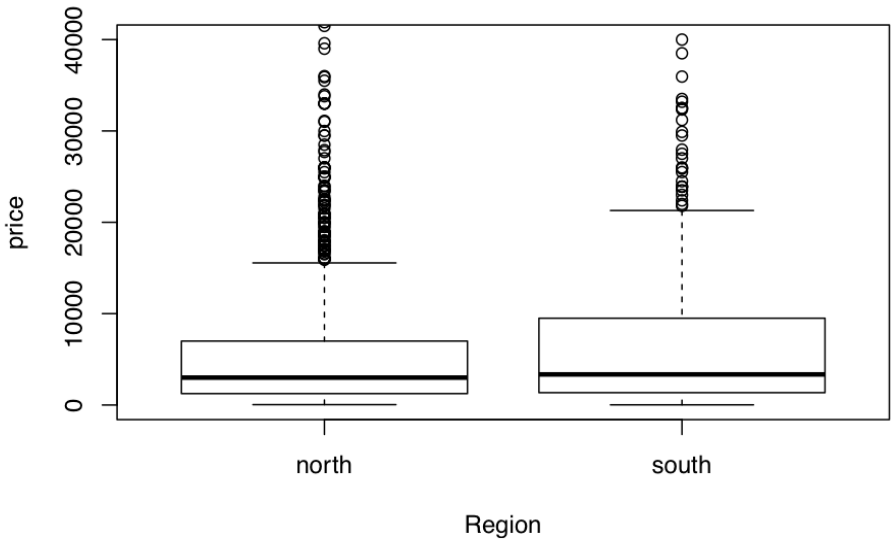
Kilometer We anticipated kilometer being a continuous variable but its field must have been a dropdown selector because it only has 13 unique values The smallest level at 5000 seemed to on averge have a low price. After that, however, the average price decreased and kilometer increased. Peak mean price was experienced at kilometer = 20000

```
#treat the kilometer to categorical data
train$kilometer = as.factor(train$kilometer)
test$kilometer = as.factor(test$kilometer)
```

PostalCode The postalCode variable was quite granular. After some research online, we knew that the first digit in the postal code signifies one of nine regions of germany. The second digit indicates a subregion of the higher level region. This pattern continues with each consecutive digit. Therefore, in order to generalize the data into larger buckets and decrease granularity, we extracted the first digit of postalCode and plotted that against price. For the most part, the spread of price between region levels was consistent and the mean the same, though '8' was slightly larger on average. The region corresponding to 8 was in the southernmost part of Germany, which we found is characterized by mountains. Perhaps cars made to deal with hilly and mountainous landscapes are on average more expensive. To implement this idea in the data, we created a new column that is 'South' if the region is 7,8 or 9, and 'North' otherwise. Note, 7,8,9 correspond to the southern regions of Germany. We thought this may add significant value, though it is also quite probable the that higher prices associated with cars sold in south germany will be explained by other feature values.

```
train$region = substr(train$postalCode,1,1)
train$region = ifelse(train$region == "7", "south", ifelse(train$region == "8", "south", ifelse(train$r

test$region = substr(test$postalCode,1,1)
```

```
test$region = ifelse(test$region == "7", "south", ifelse(test$region == "8", "south", ifelse(test$regio
#train$region = ifelse(train$region == "8", "south", "north")
boxplot(train$price~train$region,xlab="Region",ylab="price",ylim=c(0,40000))
```

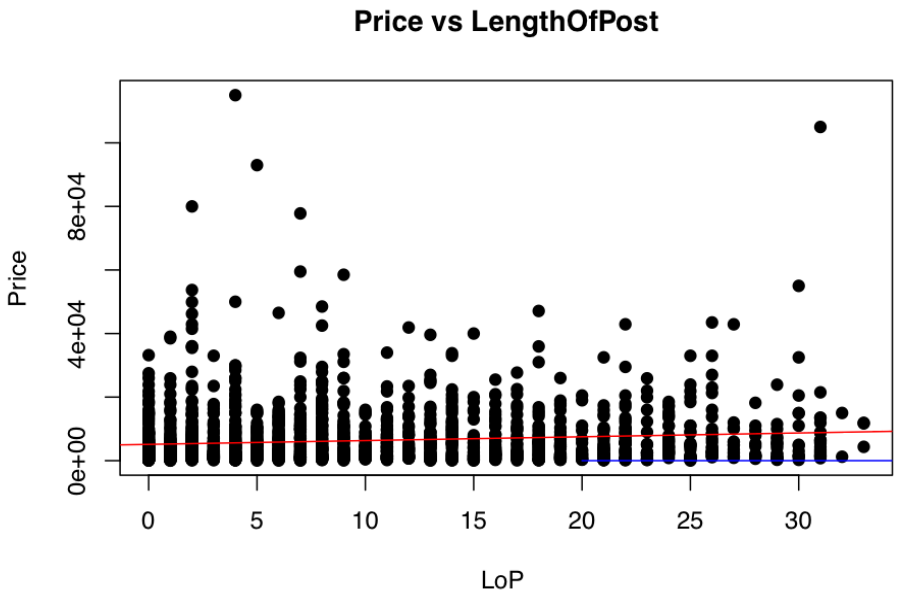


There is not much difference here so it probably won't be too useful for our models.

We then moved on to exploring the three date features in the set: dateCrawled - when this ad was first crawled, all field-values are taken from this date dateCreated - the date for which the ad at ebay was created lastSeenOnline - which is when the crawler first saw the add

One feature we thought might be useful to extract was the difference between date created and date last seen online. This would give an approximation of how long the car has been posted for sale. Perhaps cars that have been up for longer are less or more expensive?

```
#Take the difference for date created and date of last seen
train$dateCreated = as.Date(train$dateCreated)
train$lastSeen = as.Date(train$lastSeen)
train$lengthOfPost = difftime(train$lastSeen,train$dateCreated,  units = "days")

#sort(train$lengthOfPost) #a few extreme values can be observed

row = which(train$lengthOfPost >40 )
temp = train[-row,]

#plot against price
plot(temp$lengthOfPost, temp$price, main="Price vs LengthOfPost",
    xlab="LoP ", ylab="Price", pch=19)

abline(lm(train$price~train$lengthOfPost), col="red") # regression line (y~x)
lines(lowess(train$price,train$lengthOfPost), col="blue") # lowess line (x,y)
```



There appears to be very little interaction going on between the length of the post and the price. The fited linear line has a slight upward slope but it is hardly noticeable. This lack of significance might be explained by the fact that this data only spans around 4 months. If it were longer, the true affect of the length of a post could be determined. We therefore will most likely not need to include this feature in our model. We won't create this feature for the test set.

```
train$lengthOfPost = NULL
```

Hypothesized interactions

We hypothesized there might an interaction between the horsepower and the car type. This is because increasing the horsepower of a luxury car implies a higher class car that will be faster and more expensive whereas increasing the horsepower of a compact car, for example, will make it more usefull - still raising its price but by a lesser amount. We tested the interaction using ggplot below and our hypothesis held true. The price of luxury cars is more strongly correlated with increases in price than the price of compact cars. The effect increasing horsepoweron had on the price of family type cars was inbetween the two.

```
#Visualize the potential interaction
library(GGally)
```

```
##
## Attaching package: 'GGally'
```

```
## The following object is masked from 'package:dplyr':
##
##     nasa
```

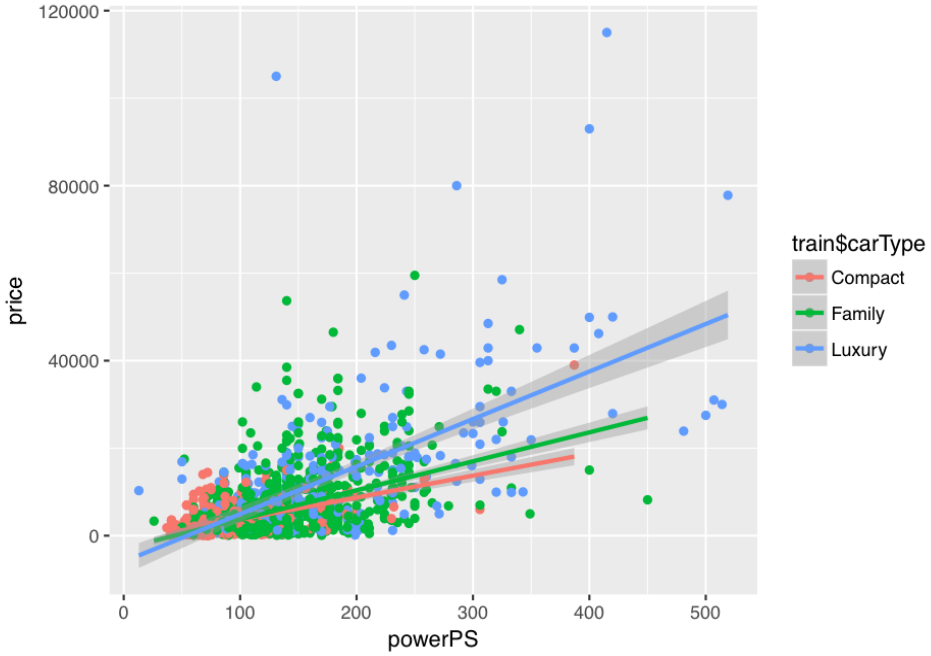
```
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
```

```
##     margin
```

```
qplot(x = powerPS,y = price,data = train, color = train$carType) +geom_smooth(method = "lm")
```



Constructing a Regression Tree

```
train= train %>% mutate_if(is.character, as.factor)
test = test %>% mutate_if(is.character, as.factor)

train$kilometer = as.integer(train$kilometer)
test$kilometer = as.integer(test$kilometer)

Y_test = test[,c("price")]
drops <- c('name','model','brand', 'postalCode', 'vehicleType')
X_test = test[ , !(names(test) %in% drops)]

Y_train = train[,c("price")]
drops <- c('name','model' ,'brand', 'postalCode', 'vehicleType')
X_train = train[ , !(names(train) %in% drops)]

set.seed(50)
rf.fit=randomForest(price~.,data=X_train,importance=TRUE , mtry=6, ntrees=500)


#mse train
rf.pred_train=predict(rf.fit,X_train)
print('train mse:')
```

```
## [1] "train mse:"
```

```
mean((rf.pred_train-Y_train)^2)
```

```
## [1] 3858878
```

```
#mse test
rf.pred=predict(rf.fit,newdata=X_test)
print('test mse:')
```

```
## [1] "test mse:"
```

```
mean((rf.pred-Y_test)^2)
```

```
## [1] 12319508
```

Typically mtry is set to p/3 for regression. We chose 6 because it produced the best MSE value. ntrees was selected to be 500 because this value has been quite common throughout the course and it also produced an optmially low MSE value.

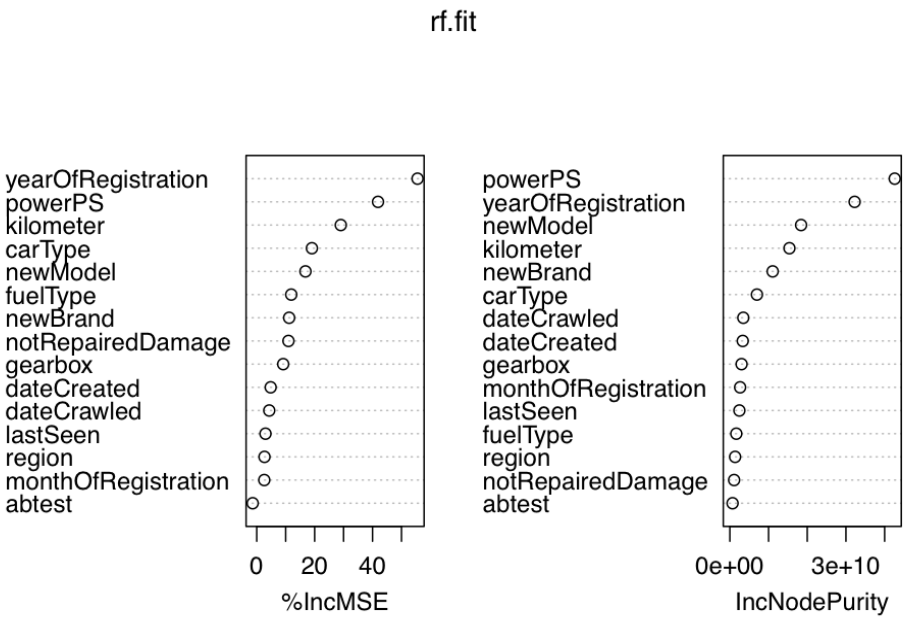
```
rf.pred=predict(rf.fit,newdata=X_test)
mean((rf.pred-Y_test)^2)
```

```
## [1] 12319508
```

```
importance(rf.fit)
```

```
##                      %IncMSE IncNodePurity
## dateCrawled          4.318701    3470130564
## abtest              -1.400930     690919398
## yearOfRegistration  55.529672   32251278265
## gearbox              9.138162    3021267092
## powerPS             41.892471   42604856892
## kilometer           29.040894   15392203203
## monthOfRegistration  2.593466    2647946281
## fuelType            11.924614    1660688752
## notRepairedDamage   10.957207    1091495952
## dateCreated          4.809604    3327870680
## lastSeen             3.020396    2457368813
## newModel            16.827034   18411314625
## newBrand            11.195484   11059534633
## carType             19.052464    6995796683
## region               2.675686    1350371878
```

```
varImpPlot(rf.fit)
```



YearOfRegistration was most useful in terms of reducing the MSE, with powerPS coming in fairly close second. However, the reverse was true when it came to Incoming Node Purity, and powerPS was first with yearOfRegistration second. Kilometer came in third for both. After that the results are fairly jumbled.

Note that we set kilometer to integer (continuous) here to allow the tree to capture the arc like pattern found between price in kilometer (see our boxplot). This significantly reduced the MSE from the random forest built with kilometer as a factor.

We discluded any interaction discovered in EDA because we assume the algorithm unveils interactions on its own.

Next we implemented a Lasso regression model. We chose this model because it does a great job at automatic feature selection and our data had many factor variables. We knew that it would push insignificant factor levels to zero, allowing for simpler more interpretabile model. The linear nature of this model also increases interpretability.

```
set.seed(11)
train$price_log = log(train$price)
test$price_log = log(test$price)

#make lasso predict non logged price
xfactors <- model.matrix(train$price ~ train$abtest + train$gearbox + train$newModel + train$kilometer +
x <- as.matrix(data.frame(xfactors))
y<- as.integer(train$price)
y.test <- as.integer(test$price)
lasso.cv = cv.glmnet(x,y,alpha=0)
pred.lasso = predict(lasso.cv,newx=x)
#training MSE
mean((y-pred.lasso)^2)
```

```
## [1] 31156469
```

```r
#testing MSE
xtestfactors <-model.matrix(test$price ~ test$abtest + test$gearbox + test$newModel + test$kilometer + 
x_test <- as.matrix(data.frame(xtestfactors))
pred.lasso = predict(lasso.cv, newx=x_test)
mean((as.integer(test$price)-pred.lasso)^2)
```

```
## [1] 23854367
```

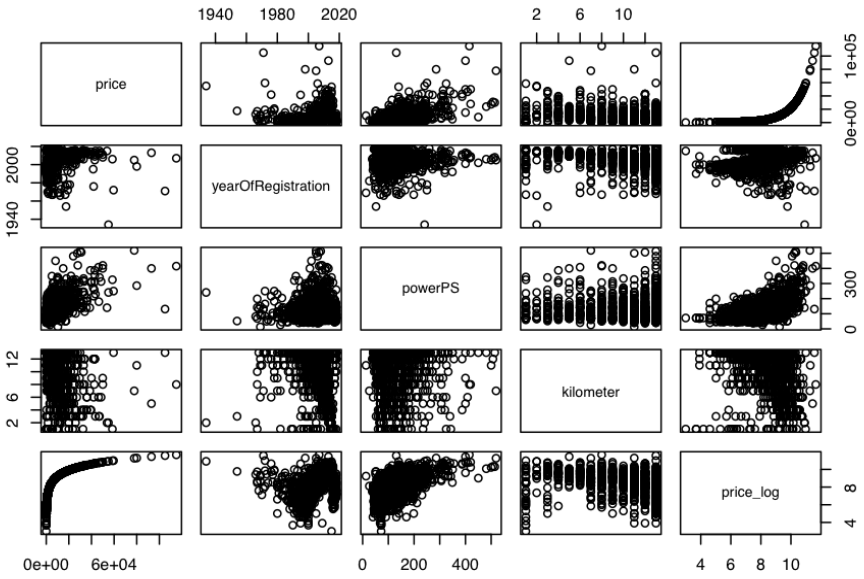
```r
#make lasso predicted on logged price
xlogfactors <-model.matrix(train$price_log ~ train$abtest + train$gearbox + train$newModel + train$kilo
x_train_log <- as.matrix(data.frame(xlogfactors))
y_log<- as.integer(train$price_log)
lasso.log.cv = cv.glmnet(x_train_log,y_log,alpha=0)
pred.log.lasso = predict(lasso.log.cv, newx=x_train_log)
#training MSE
mean((exp(y_log)-exp(pred.log.lasso))^2)
```

```
## [1] 14720231
```

```r
xtestlogfactors <-model.matrix(test$price_log ~ test$abtest + test$gearbox + test$newModel + test$kilom
x_test_log <- as.matrix(data.frame(xtestlogfactors))
y_test_log<- as.integer(test$price_log)
pred.log.lasso = predict(lasso.log.cv, newx=x_test_log)
#test MSE
mean((exp(y_test_log)-exp(pred.log.lasso))^2)
```

```
## [1] 13841453
```

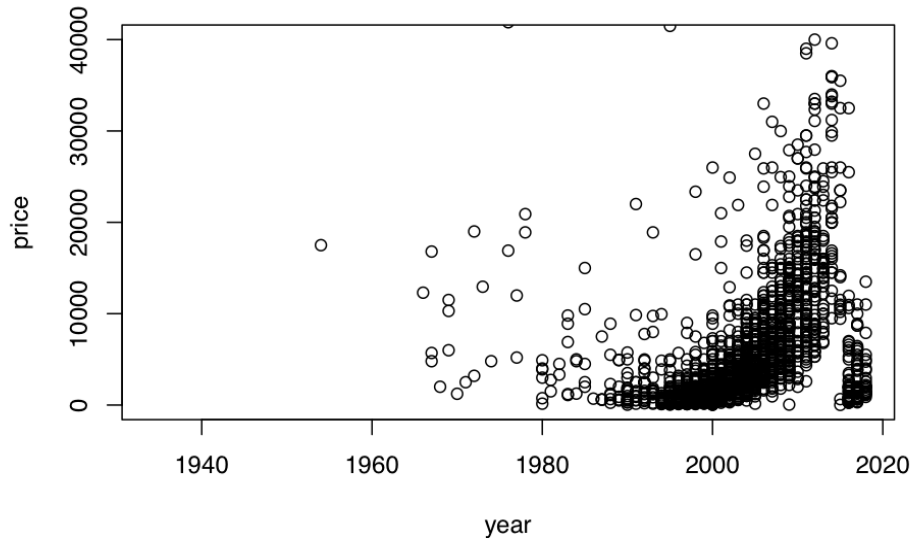
```r
#testing assumptions
cont = c('price', 'yearOfRegistration', 'powerPS', 'kilometer')
temp_train =train[cont]
temp_train$price_log = log(temp_train$price)
plot(temp_train)
```



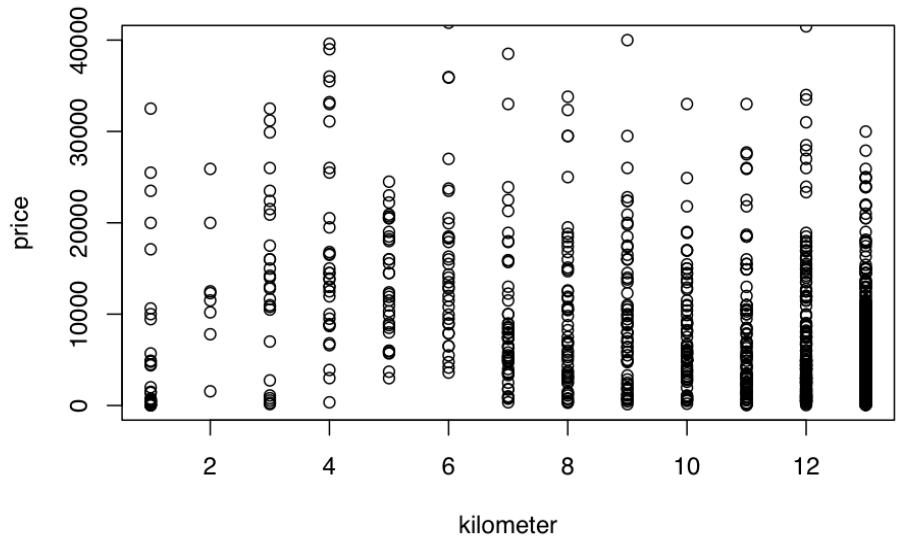
We chose to model both the log and the non-log price for the lasso because we felt the continuous data might be more linearly related to log(price) and maybe even powerPS. This fact was indeed supported by our findings and visual provided above. One can see that the relationship between the price_log and yearOfRegistraion/powerPS is definitely more linear. With kilometers, however, the impact appears negligible. And the model predicting log(price) performed significantly better when price was transformed this way.

Interactions were included because the LASSO does not discover them on its own (unlike regression forests)

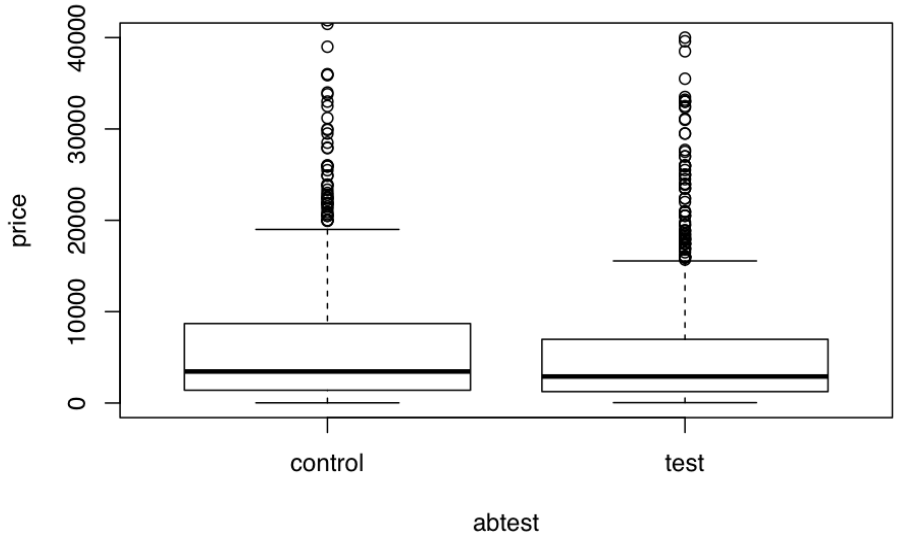
```
#change kilometer back to factor
train$kilometer = as.numeric(train$kilometer)
test$kilometer = as.numeric(test$kilometer)


#these functions are used to further confirm what we would like to use in predicting price
plot(train$price~train$yearOfRegistration,xlab="year",ylab="price",ylim=c(0,40000)) #local regression
```



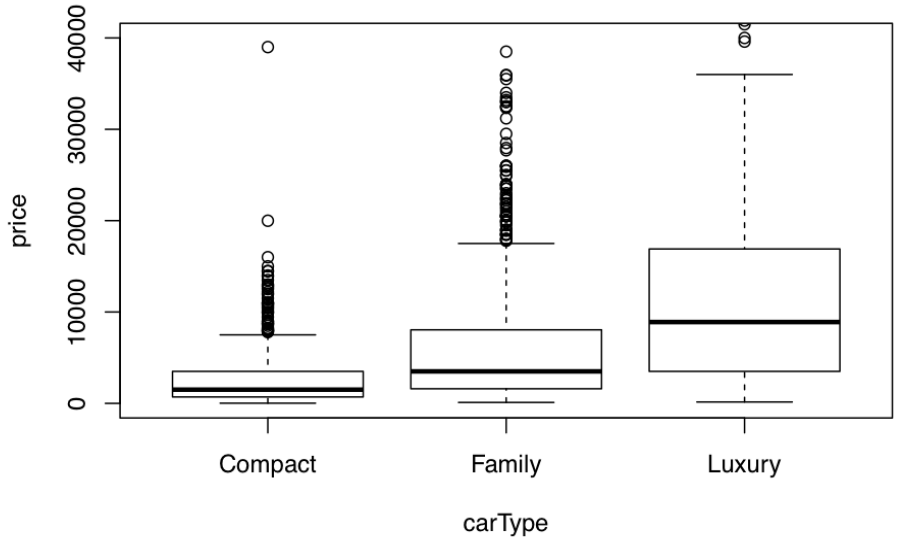

```
plot(train$price~train$kilometer,xlab="kilometer",ylab="price",ylim=c(0,40000)) #significantly differen
```



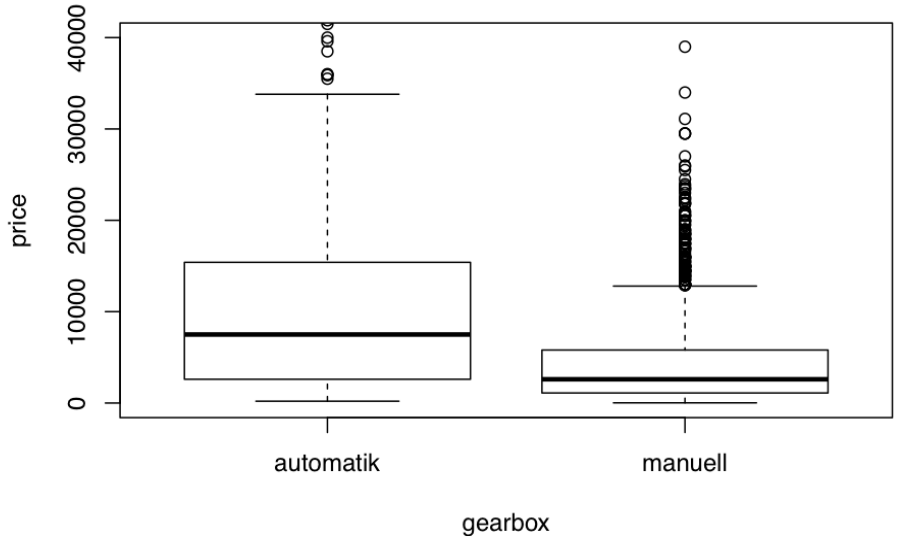

```
plot(train$price~train$abtest,xlab="abtest",ylab="price",ylim=c(0,40000)) #looks similar
```



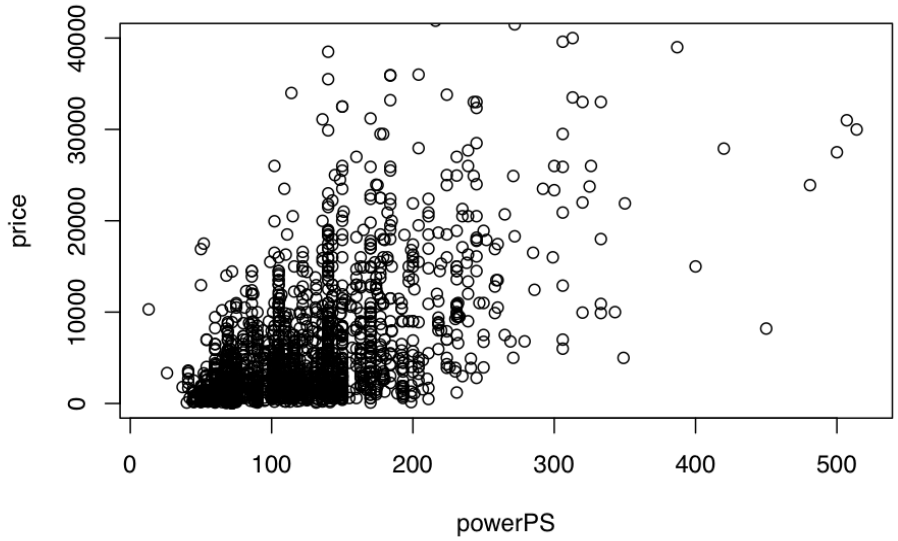

```
plot(train$price~train$carType,xlab="carType",ylab="price",ylim=c(0,40000))#luxury car have significant
```



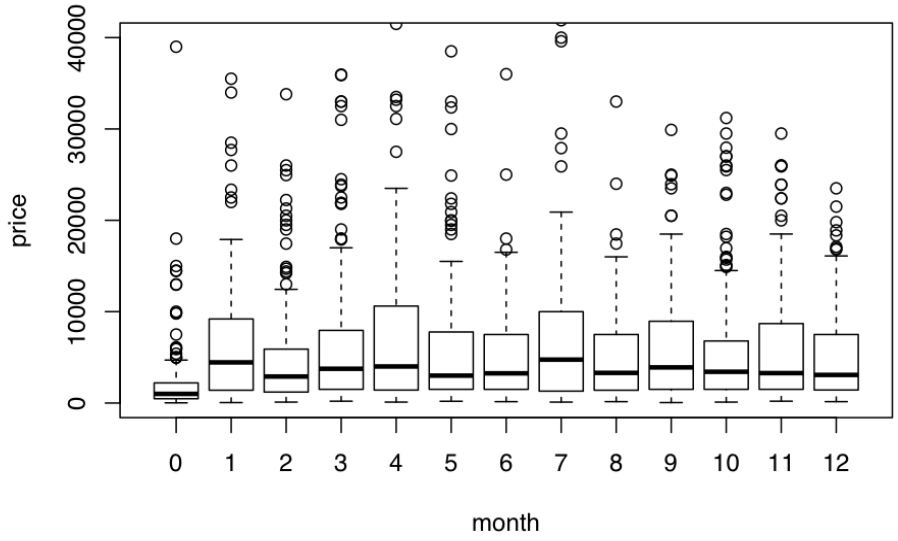

```
plot(train$price~train$gearbox,xlab="gearbox",ylab="price",ylim=c(0,40000))#manuell generally has lower
```



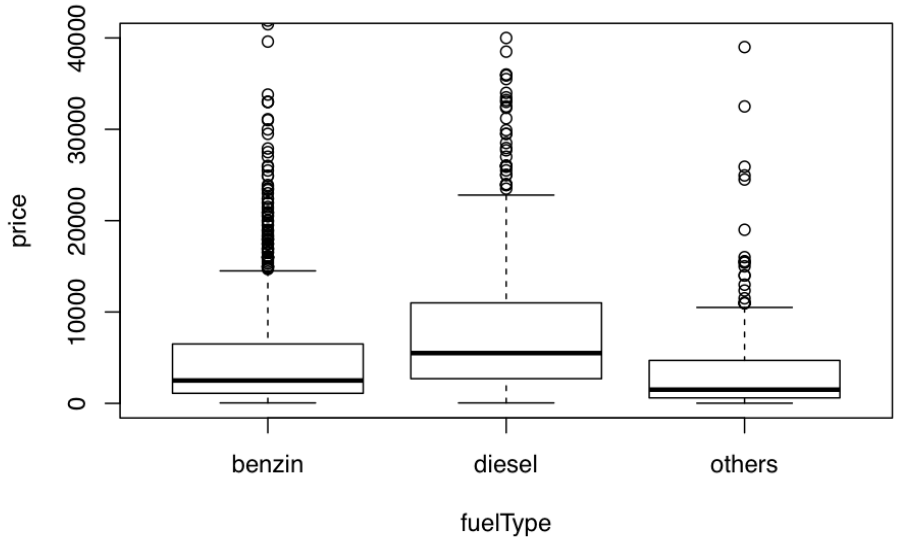

```
plot(train$price~train$powerPS,xlab="powerPS",ylab="price",ylim=c(0,40000))
```



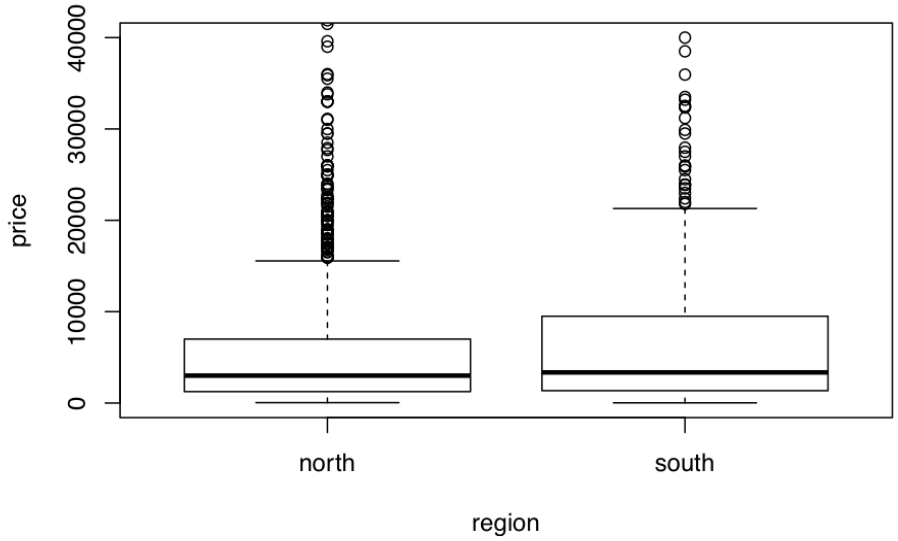

```
boxplot(train$price~train$monthOfRegistration,xlab="month",ylab="price",ylim=c(0,40000))#Not significan
```



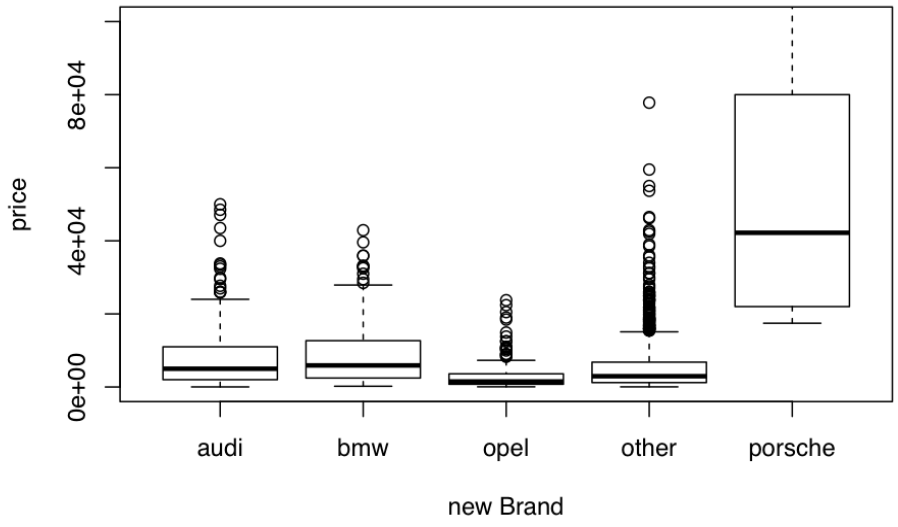

```
plot(train$price~train$fuelType,xlab="fuelType",ylab="price",ylim=c(0,40000))#Potential factor to be co
```



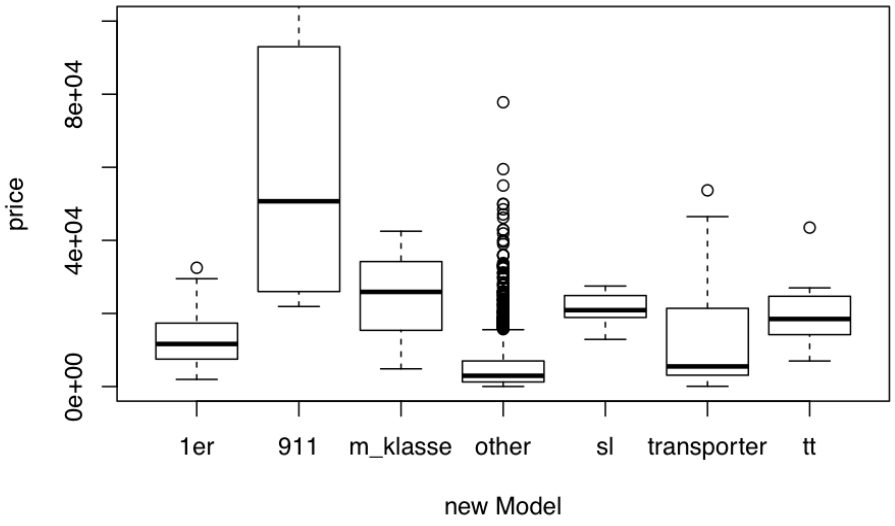

```
plot(train$price~train$region,xlab="region",ylab="price",ylim=c(0,40000))
```



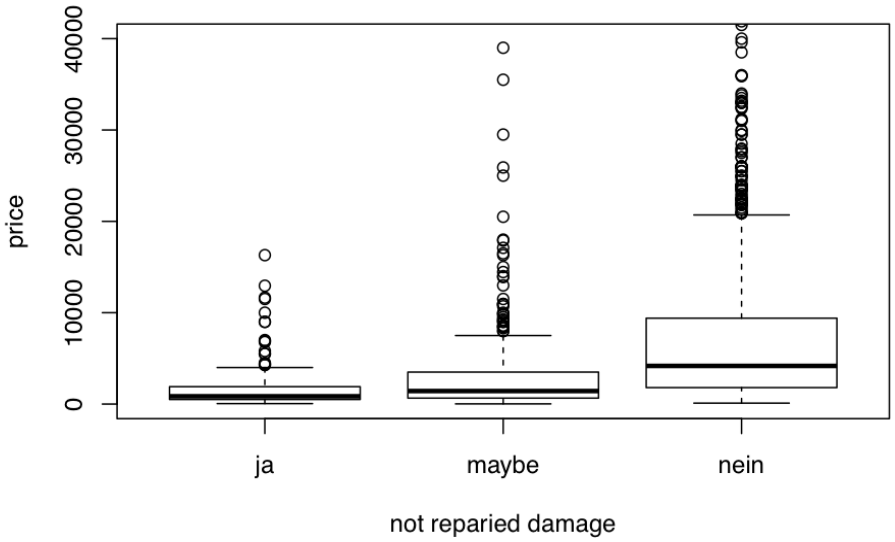

```
plot(train$price~train$newBrand,xlab="new Brand",ylab="price",ylim=c(0,100000))#Some of brand indeed ha
```




```
plot(train$price~train$newModel,xlab="new Model",ylab="price",ylim=c(0,100000))#Model could be also a g
```




```
plot(train$price~train$notRepairedDamage,xlab="not reparied damage",ylab="price",ylim=c(0,40000))#Signi
```




```
#test MSE:17975648
gams = gam(price~lo(powerPS,span=0.5)+lo(yearOfRegistration,span=0.1)+kilometer+carType+gearbox+fuelTyp
preds.gam=predict(gams,newdata=X_test)
```

```
## Warning in gam.lo(data[["lo(powerPS, span = 0.5)"]], z, w, span = 0.5,
## degree = 1, : eval 579
```

```
## Warning in gam.lo(data[["lo(powerPS, span = 0.5)"]], z, w, span = 0.5,
## degree = 1, : upperlimit 521.53
```

```
## Warning in gam.lo(data[["lo(powerPS, span = 0.5)"]], z, w, span = 0.5,
## degree = 1, : extrapolation not allowed with blending
```

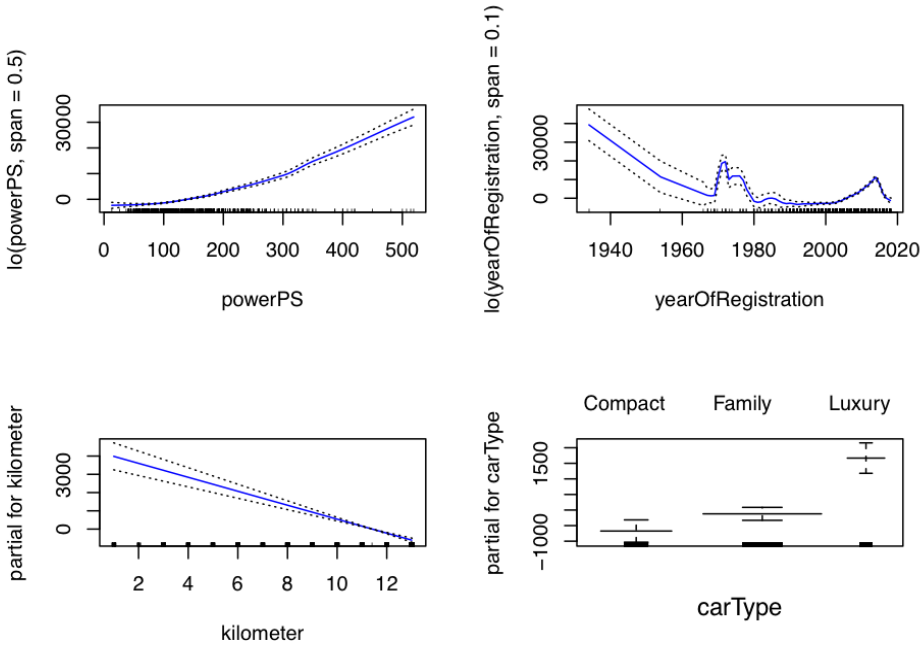
```
mean((preds.gam-Y_test)^2)
```

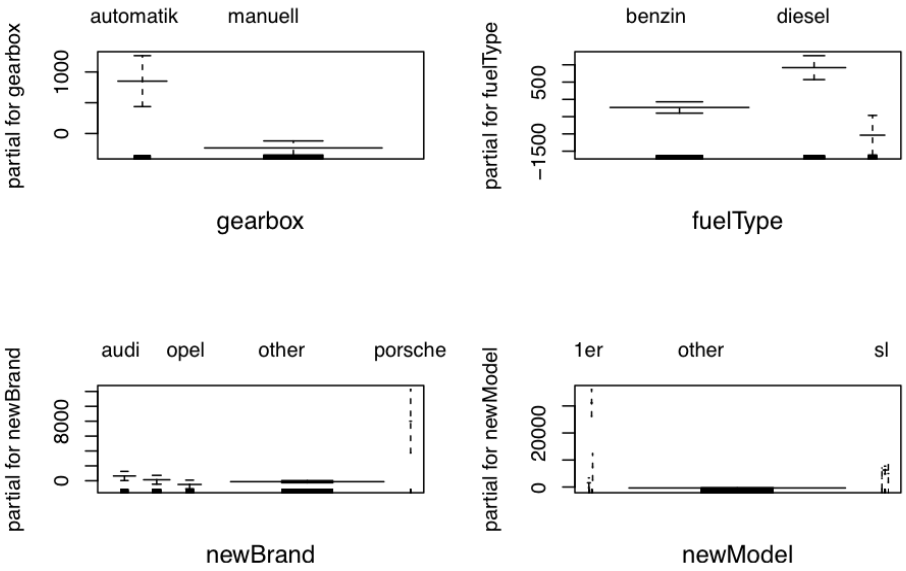
```
## [1] 17973430
```

```
#train MSE:17023615
pred.train=predict(gams,newdata=train)
mean((pred.train-Y_train)^2)
```

```
## [1] 17365033
```

```
par(mfrow=c(2,2))
plot(gams, se=T, col='blue')
```





This GAMs model was quite useful, since it allowed us to implement the local spline fit in our EDA. To keep the model simple we utilized factor variables previously discovered important, both via EDA and model creation. The GAMS model treats factor variables/levels similar to the step function, where certain levels cause a increase or decrease in predicted outcome (regardless of any interactions, since they aren't discovred by GAMS).

With this model, we can examine the effect of each predictor on the response (see charts above) when holding the rest of the predictors constant. Thus, inference is easily done on this model.

$$Y(x) = \beta_0 + f_1(x_1) + f_2(x_2) + f_3 * x_3 + f_4 * (x_4) + f_5(x_5) + f_6(x_6) + f_7 * x_7 + f_8 * (x_8) + \epsilon$$

Local Regression for Year: for each x1

$$f_1(x_1) = \beta_0 + \beta_1 * x_1 + \epsilon$$

Local Regression for powerPS: for each x2

$$f_2(x_2) = \beta_0 + \beta_1 * x_2 + \epsilon$$

Step Function for kilometer:

$$f_3(x_3) = \beta_0 + \beta_1 * C_1 * x_3 + ... + \beta_k * C_k * x_3$$

Step Function for carType:

$$f_4(x_4) = \beta_0 + \beta_1 * C_1 * x_4 + ... + \beta_k * C_k * x_4$$

Step Function for gearbox:

$$f_5(x_5) = \beta_0 + \beta_1 * C_1 * x_5 + ... + \beta_k * C_k * x_5$$

Step Function for fuelType:

$$f_6(x_6) = \beta_0 + \beta_1 * C_1 * x_6 + ... + \beta_k * C_k * x_6$$

Step Function for newBrand:

$$f_7(x_7) = \beta_0 + \beta_1 * C_1 * x_7 + ... + \beta_k * C_k * x_7$$

Step Function for newModel:

$$f_8(x_8) = \beta_0 + \beta_1 * C_1 * x_8 + ... + \beta_k * C_k * x_8$$

Motivation As we conduct our EDA, we realized that the relationship of each variable against the price differs. Then, we think it may be unreasonable to fit a same function for all of predictor variable. Therefore, we believes the GAMS model can be a good tool to build a predictive model for us. However, because it will not conduct the feature selection for us like LASSO, we will carefully explore the interesting variable by making boxplot and scatter for each variables, to avoid overfitting issue. After exploring the each plot carefully, I think it would be helpful if we can fit local regression for year because the plot seems follow different distribution for differeant ranges. Also, the powerPS is clustered together, and could be potentially used to predict price. Additionally, kilometer, carType, gearBox,fuelType,newBrand, and newModel seems to have significantly different means for different level.

Assumption Speaking of the model for local regression, we actually assume it is linear around the region of x0 for year because the coefficient are all computed by weighted least square near each value of X0. But we think it is sufficient, because we can clearly see the linear pattern of year within some range. For the PowerPS, we also fit it by using local regression, but the linearity is apparent. For the assumption of step function, we actually assuming for each level of factor, the mean is constant. I think it is a safe assumption to be made, because we can clearly see that the mean is significantly different for each level of factor (kilometer, carType, gearBox,fuelType,newBrand, and newModel)

Validation train MSE:17023615 test MSE:17975648

Which is reasonably good, but achieve the worst performance among three models.

In conclusion, the random forest model outperformed the other two. We beleive this was primarily caused by its ability its one-discover and utilize interaction effects accuractly, and two-find proper cutoff points for variables whose values can dramatically impact price before/after certain values.A well constructed log-lasso came in close second, and the GAMS model third.