Bagging and Boosting

Learning Objectives:

- Learn to train a Random Forest using ranger function
- Learn to train a XGBoost algorithm using xgboost
- Learn to run CV using XGBoost using the xgb package
- Learn to perform a grid search over hyper parameters
- Explore Partial Dependence Plots (pdp)

Import packages

Random Forests

We are going to use cervical cancer dataset to predict the `Biopsy` variable, which serves as the gold standard for diagnosing cervical cancer.

Load and preprocess the data

```
## Data processing according to https://github.com/christophM/interpretable-ml-book/blob/master/R/get-c
get.cervical.data = function(){
  cervical = read.csv('cervical.csv', na.strings = c('?'), stringsAsFactors = FALSE)
  cervical = select(cervical, -Citology, -Schiller, -Hinselmann)
  cervical$Biopsy = factor(cervical$Biopsy, levels = c(0, 1), labels=c('Healthy', 'Cancer'))

  ## subset variables to the ones that should be used in the book
  cervical = dplyr::select(cervical, Age, Number.of.sexual.partners, First.sexual.intercourse,
  Num.of.pregnancies, Smokes, Smokes..years., Hormonal.Contraceptives, Hormonal.Contraceptives..years.,
  IUD, IUD..years., STDs, STDs..number., STDs..Number.of.diagnosis, STDs..Time.since.first.diagnosis,
  STDs..Time.since.last.diagnosis, Biopsy)

  # NA imputation
  imputer = mlr::imputeMode()


  cervical_impute = mlr::impute(cervical, classes = list(numeric = imputeMode()))
  cervical = cervical_impute$data
  #cervical = relevel(cervical, "Healthy")
  cervical
}

cervical = get.cervical.data()

cervical$Biopsy = as.factor(cervical$Biopsy)
```

Fit RF model using the ranger function

```
rf.fit <- ranger(
  formula = Biopsy ~ .,
  data            = cervical,
  num.trees       = 250,
  importance      = 'impurity',
```

```
  probability = T
)
```

Create custom predict function that returns the predicted values as a vector

```
pred.rf.fun <- function(model, newdata)
  predict(model, data = newdata)$predictions
```

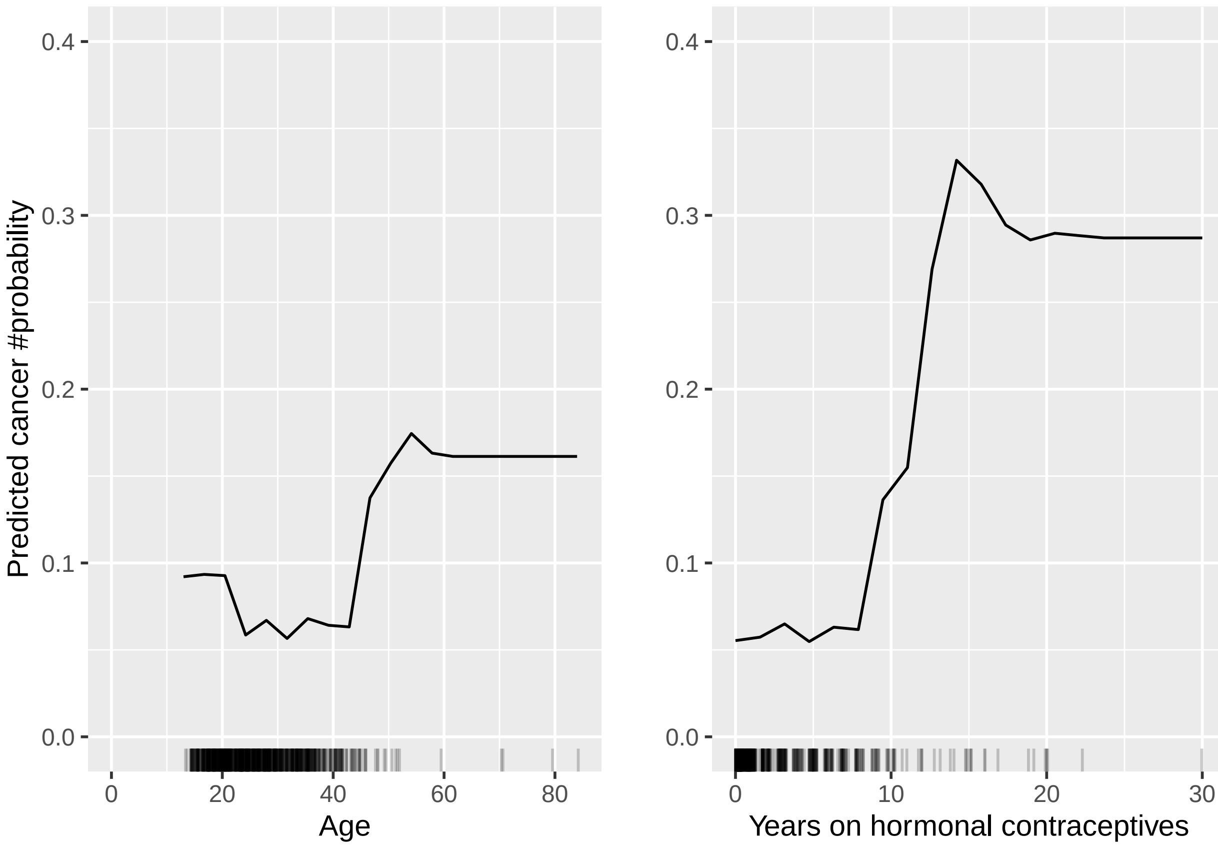
Create environment using the fitted model

```
mod <- Predictor$new(rf.fit,
                     data = cervical,
                     class = "Cancer",
                     predict.fun = pred.rf.fun
                     )
```

PDP: - Partial dependence plots allow us to examine the extent to which our target is affected by features - Create partial dependence plot for "Age" and "Hormonal.Contraceptives..years."

Create FeatureEffect using environment

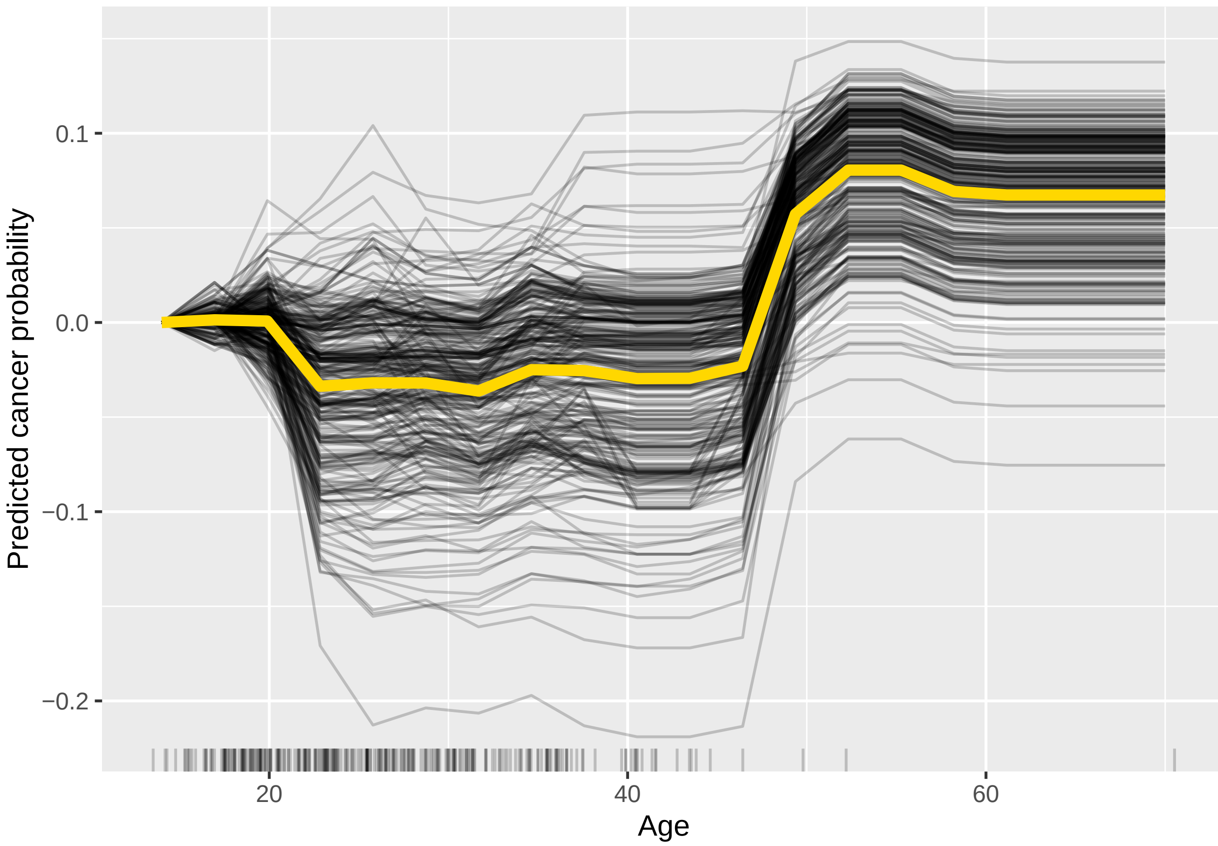
```
# Examine the effect of Age on Cancer
pdp = FeatureEffect$new(mod, 'Age', method = "pdp")
p1 = pdp$plot(ylim=c(0, 0.4)) +
  scale_x_continuous(limits = c(0, NA)) +
  scale_y_continuous('Predicted cancer #probability', limits=c(0, 0.4))

# Examine the effect of Years on Hormonal Contraceptives

pdp$set.feature("Hormonal.Contraceptives..years.")
p2 = pdp$plot(ylim=c(0, 0.4)) +
  scale_x_continuous("Years on hormonal contraceptives", limits = c(0, NA)) +
  scale_y_continuous('', limits=c(0, 0.4))

# Plot two plots side by side
gridExtra::grid.arrange(p1, p2, ncol = 2)
```



PDP + Individual Conditional Expectation (ICE)

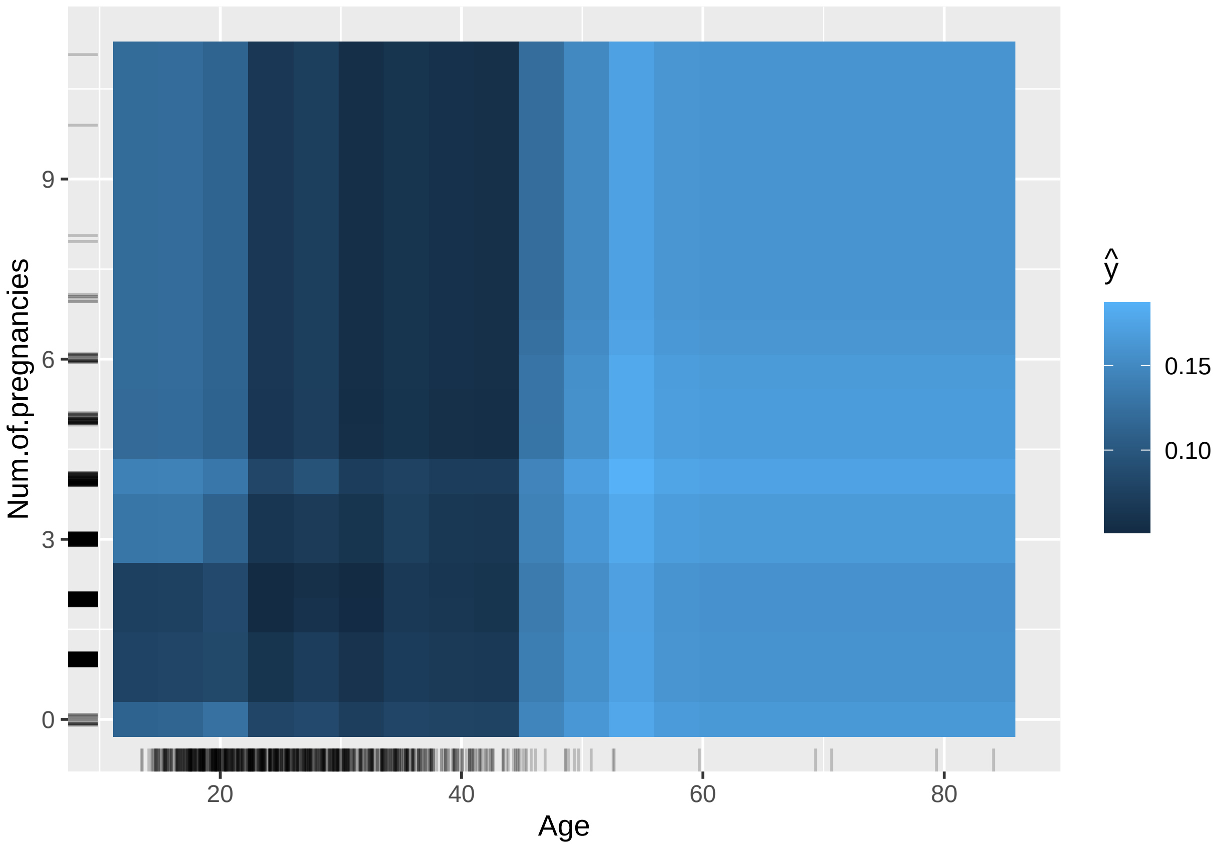
- ICE shows the effect of feature of interest on target variable per sample as opposed to average

```
# Get sample of size 300 to plot
cervical_subset_index = sample(1:nrow(cervical), size = 300)
cervical_subset = cervical[cervical_subset_index, ]

pred.cervical = Predictor$new(rf.fit,
                              data = cervical_subset,
                              class = "Cancer",
                              predict.fun = pred.rf.fun
                              )

ice = FeatureEffect$new(pred.cervical,
                        "Age",
                        center.at = min(cervical_subset$Age),
                        method = "pdp+ice")      ##method to create ICE

ice$plot() +
  scale_color_discrete(guide='none') +
  scale_y_continuous('Predicted cancer probability')
```



Visualize how the interaction of age and number of pregnancies affect cervical cancer probability

```
pd = FeatureEffect$new(mod, c("Age", "Num.of.pregnancies"), method = "pdp")
pd$plot()
```



Model Tuning - Random Forest and Boosting

Here we examine `UsedCars` data which contains factors. Download dataset if it doesn't exist

```
# download the file if it does not exist
if (!file.exists("UsedCars.csv"))
  download.file('https://github.com/ChicagoBoothML/MLClassData/raw/master/UsedCars/UsedCars.csv', 'Used
uc = read.csv('UsedCars.csv')
n = nrow(uc)
```

Examine our dataset

```
str(uc)
```

```
## 'data.frame':    20063 obs. of  11 variables:
##  $ price       : int  2988 6595 7993 5995 3000 7400 10850 8990 7950 7995 ...
##  $ trim        : chr  "320" "320" "320" "420" ...
##  $ isOneOwner  : chr  "f" "f" "f" "f" ...
##  $ mileage     : num  193296 129948 140428 113622 167673 ...
##  $ year        : int  1995 1995 1997 1999 1999 2002 2000 2001 2000 2000 ...
##  $ color       : chr  "Black" "other" "White" "Silver" ...
##  $ displacement: num  3.2 3.2 3.2 4.2 4.2 4.3 4.3 4.3 4.3 4.3 ...
##  $ fuel        : chr  "Gasoline" "Gasoline" "Gasoline" "Gasoline" ...
##  $ region      : chr  "SoA" "Mid" "Mid" "Mid" ...
##  $ soundSystem : chr  "unsp" "Premium" "Bose" "unsp" ...
##  $ wheelType   : chr  "Alloy" "Alloy" "Alloy" "Alloy" ...
```

Transform 'chr' datatypes to factor

```r
uc <- mutate_if(uc, is.character, as.factor)
str(uc)
```

```
## 'data.frame':    20063 obs. of  11 variables:
##  $ price       : int  2988 6595 7993 5995 3000 7400 10850 8990 7950 7995 ...
##  $ trim        : Factor w/ 11 levels "320","350","400",..: 1 1 1 4 4 5 5 5 5 5 ...
##  $ isOneOwner  : Factor w/ 2 levels "f","t": 1 1 1 1 1 1 1 1 1 1 ...
##  $ mileage     : num  193296 129948 140428 113622 167673 ...
##  $ year        : int  1995 1995 1997 1999 1999 2002 2000 2001 2000 2000 ...
##  $ color       : Factor w/ 7 levels "Black","Blue",..: 1 4 7 5 5 7 7 1 5 1 ...
##  $ displacement: num  3.2 3.2 3.2 4.2 4.2 4.3 4.3 4.3 4.3 4.3 ...
##  $ fuel        : Factor w/ 3 levels "Diesel","Gasoline",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ region      : Factor w/ 9 levels "ENC","ESC","Mid",..: 7 3 3 3 7 3 7 7 2 2 ...
##  $ soundSystem : Factor w/ 5 levels "Bang Olufsen",..: 5 4 2 5 5 2 5 5 2 5 ...
##  $ wheelType   : Factor w/ 4 levels "Alloy","other",..: 1 1 1 1 1 1 4 1 4 1 ...
```

Sample 5000 datapoints at random to construct training and test sets

```r
train.index = sample(nrow(uc), 5000)
uc.train = uc[train.index,]
uc.test = uc[-train.index,]
```

Random Forest Hyper-parameter tuning First, create a grid.

```r
# start time
tic()
# create a grid of all the possible HP combinations
hyper_grid <- expand.grid(
  mtry        = seq(2, 10, by = 2),         ## Number of features to split on
  node_size   = c(25, 50, 100, 150, 200),  ## Minimal node size
  sample_size = c(.55, .632, .70, .80),    ## Fraction of observation to sample - bootstrapping
  OOB_RMSE    = 0                          ## Variable to dump results
)

# loop over all permutations and train Random Forest model

for(i in 1:nrow(hyper_grid)) {

  # train model
  model <- ranger(
    formula         = price ~ .,
    data            = uc.train,
    num.trees       = 500,
    mtry            = hyper_grid$mtry[i],
    min.node.size   = hyper_grid$node_size[i],
    sample.fraction = hyper_grid$sample_size[i],
    seed            = 1246
  )

  # add OOB RMSE to grid
  hyper_grid$OOB_RMSE[i] <- sqrt(model$prediction.error)
}

# sort grid by lowest OOB RMSE
(oo = hyper_grid %>%
  dplyr::arrange(OOB_RMSE) %>%
```

```
  head(10))
```

```
##    mtry node_size sample_size OOB_RMSE
## 1     6        25       0.800     4112
## 2     6        25       0.700     4114
## 3     6        25       0.632     4120
## 4     8        25       0.800     4127
## 5     8        25       0.632     4130
## 6     8        25       0.700     4130
## 7     6        25       0.550     4130
## 8     8        25       0.550     4133
## 9     4        25       0.800     4136
## 10    4        25       0.700     4145
```

```
toc()
```

```
## 12.488 sec elapsed
```

Train model using best Hyperparameters combination - First row in grid

```
rf.fit.final <- ranger(
    formula         = price ~ .,
    data            = uc.train,
    num.trees       = 500,
    mtry            = oo[1,]$mtry,
    min.node.size   = oo[1,]$node_size,
    sample.fraction = oo[1,]$sample_size,
    importance      = 'impurity'
    )
```

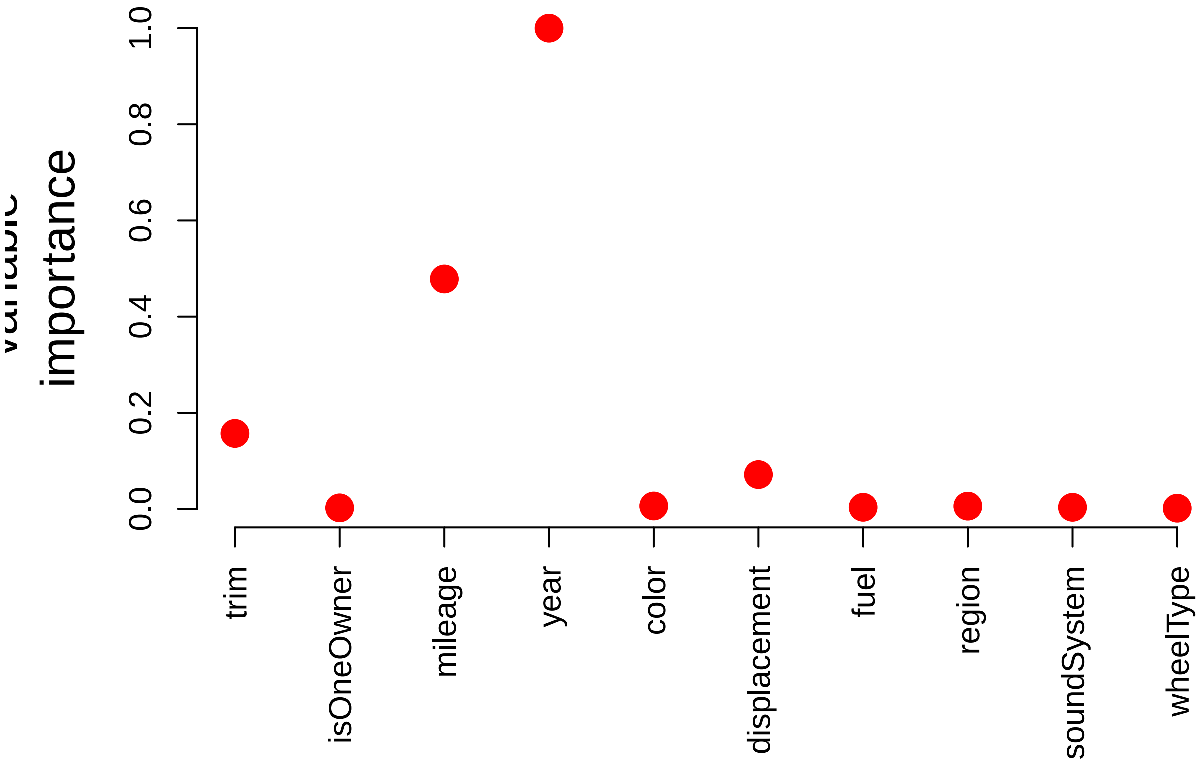
Use trained model to make predictions on test data Use true prices to calculate RMSE

```
yhat.rf = predict(rf.fit.final, data = uc.test)$predictions
error <- sqrt( (var(uc.test$price - yhat.rf)) )                            # RMSE on the test
error
```

```
## [1] 4317
```

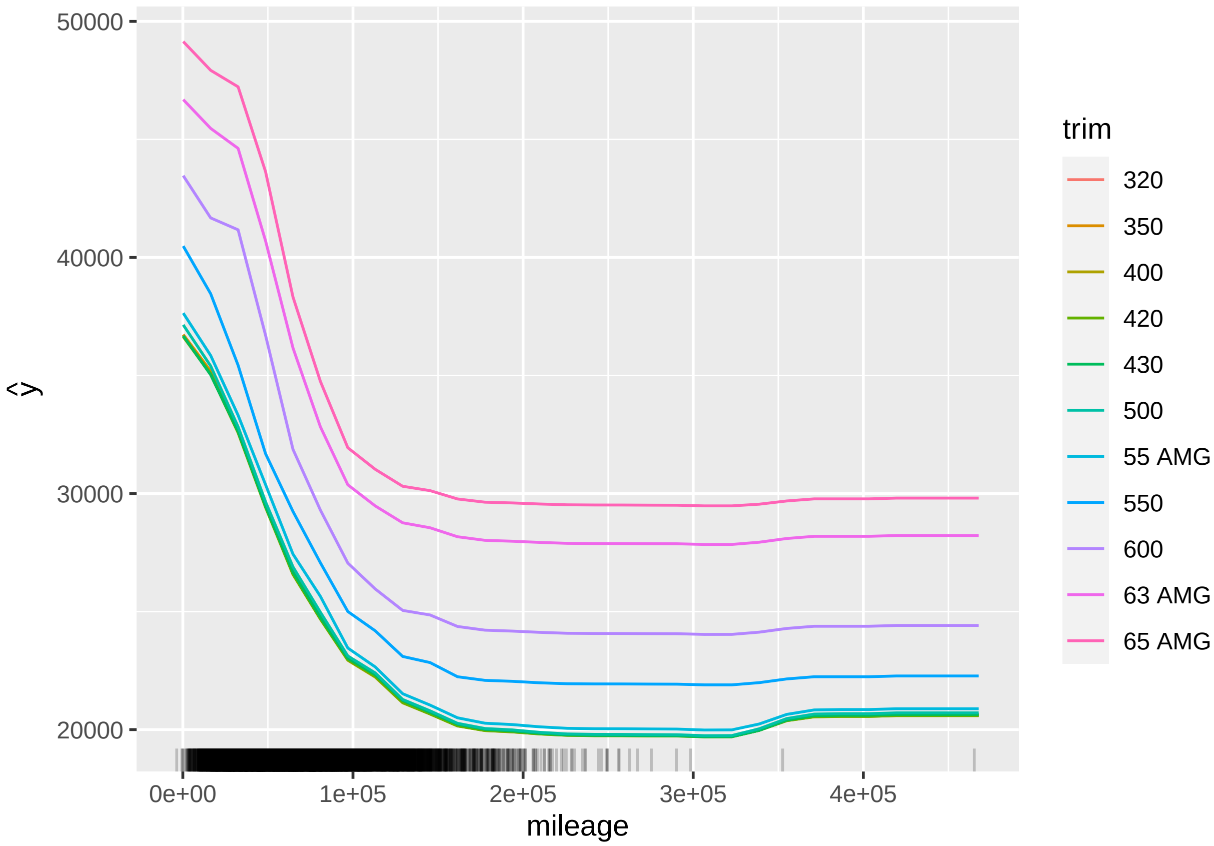
Examine feature importance on trained model

```
tvimp = importance(rf.fit.final)
par(mar=c(8,5,1,1))
plot(tvimp/max(tvimp),axes=F,pch=16,col='red',xlab="",ylab="variable
importance",cex=2,cex.lab=1.5)
axis(1,labels=names(tvimp),at=1:length(tvimp),cex=0.5,las=2)
axis(2)
```

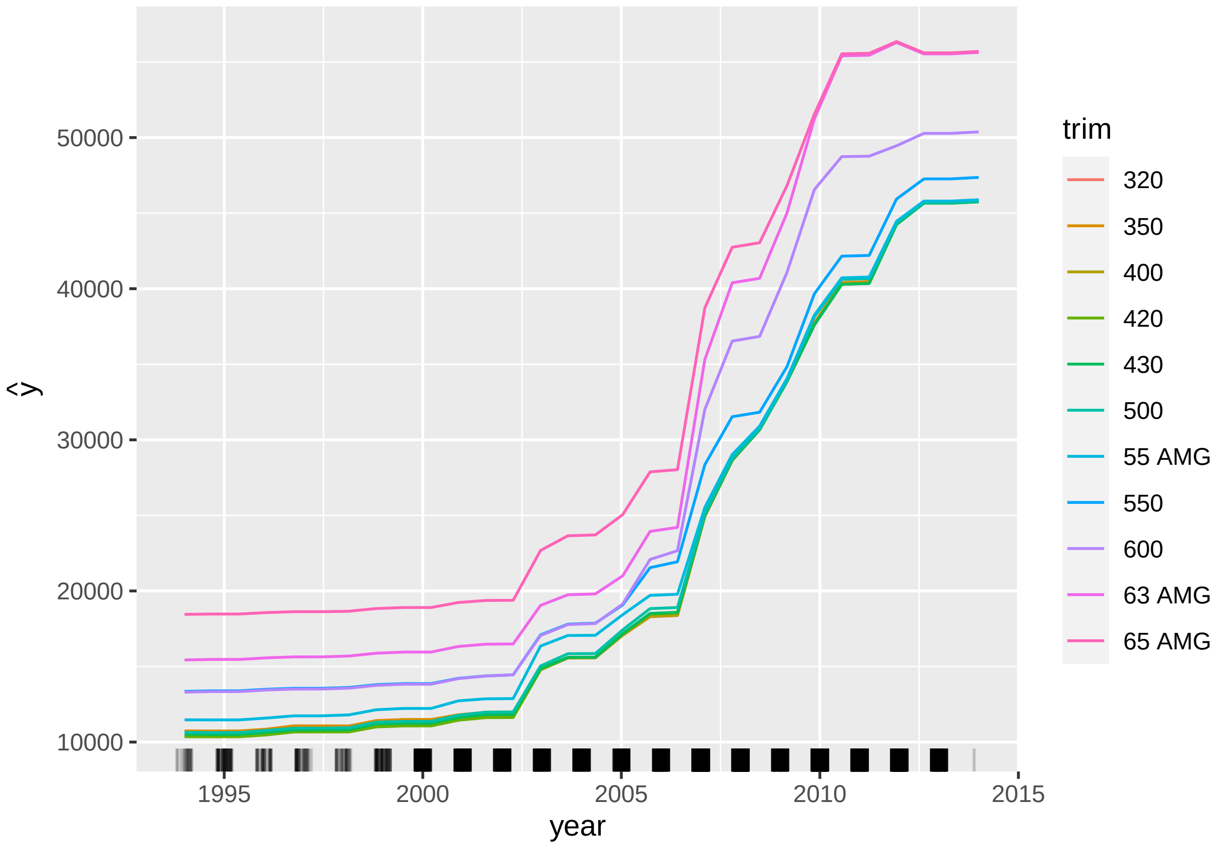



```r
# Create custom predict function that returns the predicted values as a vector
pred.rf.fun <- function(model, newdata)
  predict(model, data = newdata)$predictions


mod <- Predictor$new(rf.fit.final,
                     data = uc.train,
                     predict.fun = pred.rf.fun     # we need a custom predict function
                     )

eff <- FeatureEffect$new(mod,
          feature = c("mileage", "trim"),
          method = "pdp",
          grid.size = 30
          )
```

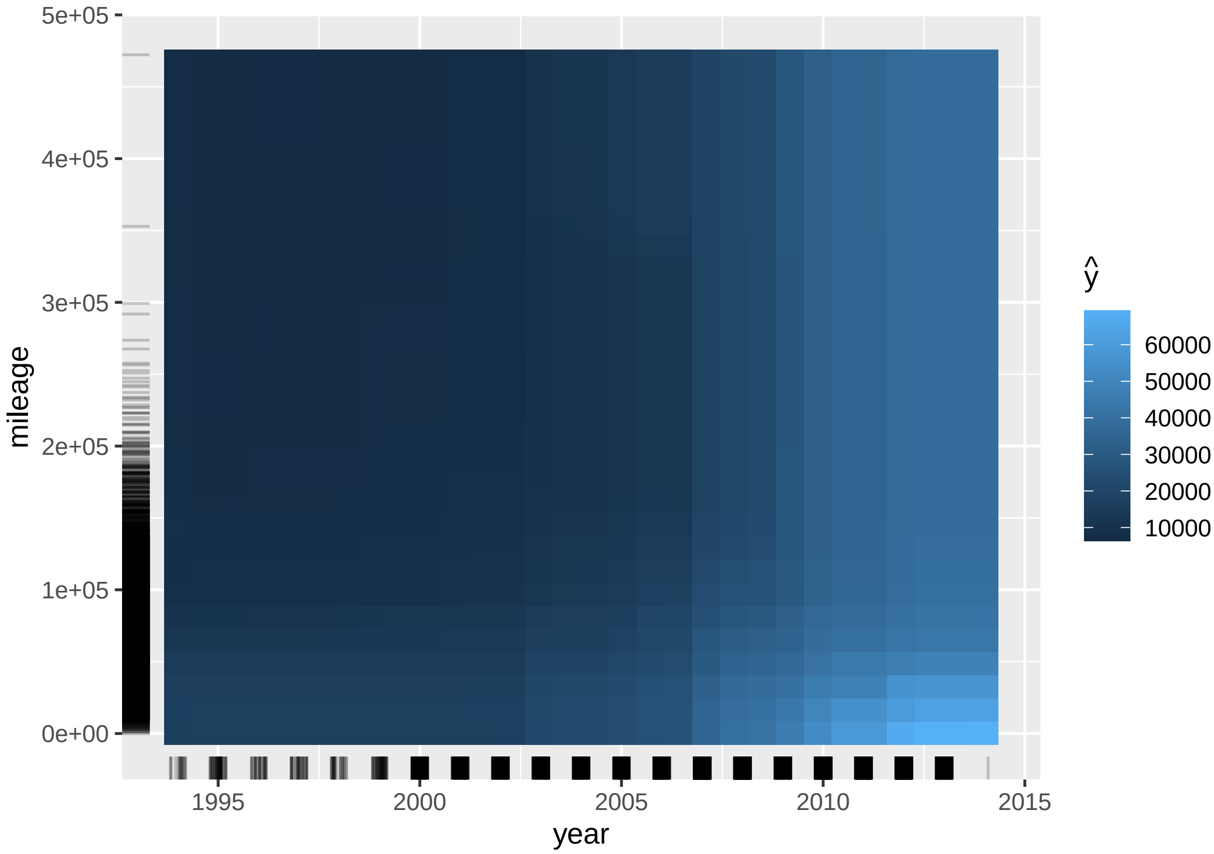
```r
plot(eff)
```




```r
eff$set.feature( c("year", "trim") )
plot(eff)
```




```
eff$set.feature( c("year", "mileage") )
plot(eff)
```



Boosting and tuning XGBoost - XGBoost only works with matrices that contain all numeric variables - We need to one hot encode our data. - There are different ways to do this in R. - `Matrix::sparse.model.matrix` - `caret::dummyVars`

```
X = sparse.model.matrix(price ~ ., data = uc)[,-1]
X.train = X[train.index, ]
Y.train = uc$price[train.index]
X.test = X[-train.index, ]
Y.test = uc$price[-train.index]
dim(X.train)
```

```
## [1] 5000   37
```

First, create a grid.

```
# create hyperparameter grid
hyper_grid <- expand.grid(
  shrinkage = c(.01, .1, .3),      ## controls the learning rate
  interaction.depth = c(1, 3, 5), ## tree depth
  n.minobsinnode = c(10, 30, 50), ##  minimum number of observations required in each terminal node
  bag.fraction = c(.5, .65, .8),  ##  percent of training data to sample for each tree
  optimal_trees = 0,               ## a place to dump results
  min_RMSE = 0                     ## a place to dump results
)
```

Now we can perform the grid search We will aslo use cross validation to

```
for(i in 1:nrow(hyper_grid)) {
```

```r
  # create parameter list
  params <- list(
    eta = hyper_grid$shrinkage[i],
    max_depth = hyper_grid$interaction.depth[i],
    min_child_weight = hyper_grid$n.minobsinnode[i],
    subsample = hyper_grid$bag.fraction[i]
  )

  # reproducibility
  set.seed(123)

  # train model using Cross Validation
  xgb.tune <- xgb.cv(
    params = params,
    data = X.train,
    label = Y.train,
    nrounds = 3000,
    nfold = 5,
    objective = "reg:squarederror",     # for regression models
    verbose = 0,                        # silent,
    early_stopping_rounds = 10          # stop if no improvement for 10 consecutive trees
  )

  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(xgb.tune$evaluation_log$test_rmse_mean)
  hyper_grid$min_RMSE[i] <- min(xgb.tune$evaluation_log$test_rmse_mean)
}
```

Arrange grid by RMSE

```r
(oo = hyper_grid %>%
      dplyr::arrange(min_RMSE) %>%
      head(10))
```

```
##    shrinkage interaction.depth n.minobsinnode
## 1       0.01                 5             10
## 2       0.01                 5             10
## 3       0.01                 5             10
## 4       0.01                 3             10
## 5       0.10                 5             10
## 6       0.10                 5             10
## 7       0.01                 3             10
## 8       0.10                 3             10
## 9       0.10                 5             10
## 10      0.01                 5             30
##    bag.fraction optimal_trees min_RMSE
## 1          0.50           701     4039
## 2          0.65           722     4039
## 3          0.80           677     4043
## 4          0.80          1519     4054
## 5          0.80            81     4055
## 6          0.65            67     4058
## 7          0.65          1398     4059
## 8          0.65           178     4061
## 9          0.50            68     4073
```

```
## 10          0.80             799      4074
```

Extract best parameters

```r
# parameter list
params <- list(
  eta = oo[1,]$shrinkage,
  max_depth = oo[1,]$interaction.depth,
  min_child_weight = oo[1,]$n.minobsinnode,
  subsample = oo[1,]$bag.fraction
)
```

Train Final Model

```r
xgb.fit.final <- xgboost(
  params = params,
  data = X.train,
  label = Y.train,
  nrounds = oo[1,]$optimal_trees,
  objective = "reg:squarederror",
  verbose = 0
)
```

Use trained model to make predictions on test data Use true prices to calculate RMSE

```r
yhat.xgb <- predict(xgb.fit.final, newdata=X.test)
sqrt( var(Y.test - yhat.xgb) )
```

```
## [1] 4278
```

Examine feature importance on trained model

```r
# create importance matrix
importance_matrix <- xgb.importance(model = xgb.fit.final)
# variable importance plot
xgb.plot.importance(importance_matrix, top_n = 10, measure = "Gain")
```