# Tutorial - Learning the iml package

Victor Hugo C. Alexandrino da Silva

9/2/2021

This tutorial aims to undertand and learn about the `iml` (Interpretable Machine Learning) package. It is based on the UC Business Analytics website (http://uc-r.github.io/iml-pkg#rep), the `iml` documentation (https://christophm.github.io/iml/articles/intro.html) and the book from Christoph Molnar (https://christophm.github.io/interpretable-ml-book/)

Our goal is to understand the tools for analyzing any black box machine learning model such as:

- Feature importance: Which were the most important features?
- Feature effects: How does a feature influence the prediction? (Partial dependence plots and individual conditional expectation curves)
- Explanations for single predictions: How did the feature values of a single data point affect its prediction? (LIME and Shapley value)
- Surrogate trees: Can we approximate the underlying black box model with a short decision tree?
- The iml package works for any classification and regression machine learning model: random forests, linear models, neural networks, xgboost, etc.

```
library(iml)
# install.packages("mlr") # For ML models
library(mlr)
```

```
## Loading required package: ParamHelpers
```

```
## Warning message: 'mlr' is in 'maintenance-only' mode since July 2019.
## Future development will only happen in 'mlr3'
## (<https://mlr3.mlr-org.com>). Due to the focus on 'mlr3' there might be
## uncaught bugs meanwhile in {mlr} - please consider switching.
```

```
set.seed(1014)

# Loading Boston house values from the MASS package database
data("Boston", package =  "MASS")

head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
```

```
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

Let's first train a randomForest to predict the Boston median house value:

```
lrn <- makeLearner("regr.randomForest", ntree = 50)

task <- makeRegrTask(data = Boston, target = "medv")

rf <- train(learner = lrn, task = task)
```

After training our random forest `rf`, we create a `Predictor` object that holds the model and the data. Since the `iml` package uses R6 data, we can call new object by writing `Predictor$new()`:

```
# Excluding our prediction variable 'medv' and building our feature set:
X = Boston[which(names(Boston) != "medv")]

# Creating the Predictor object from our trained random forest:
predictor = Predictor$new(rf, data = X, y = Boston$medv)
```

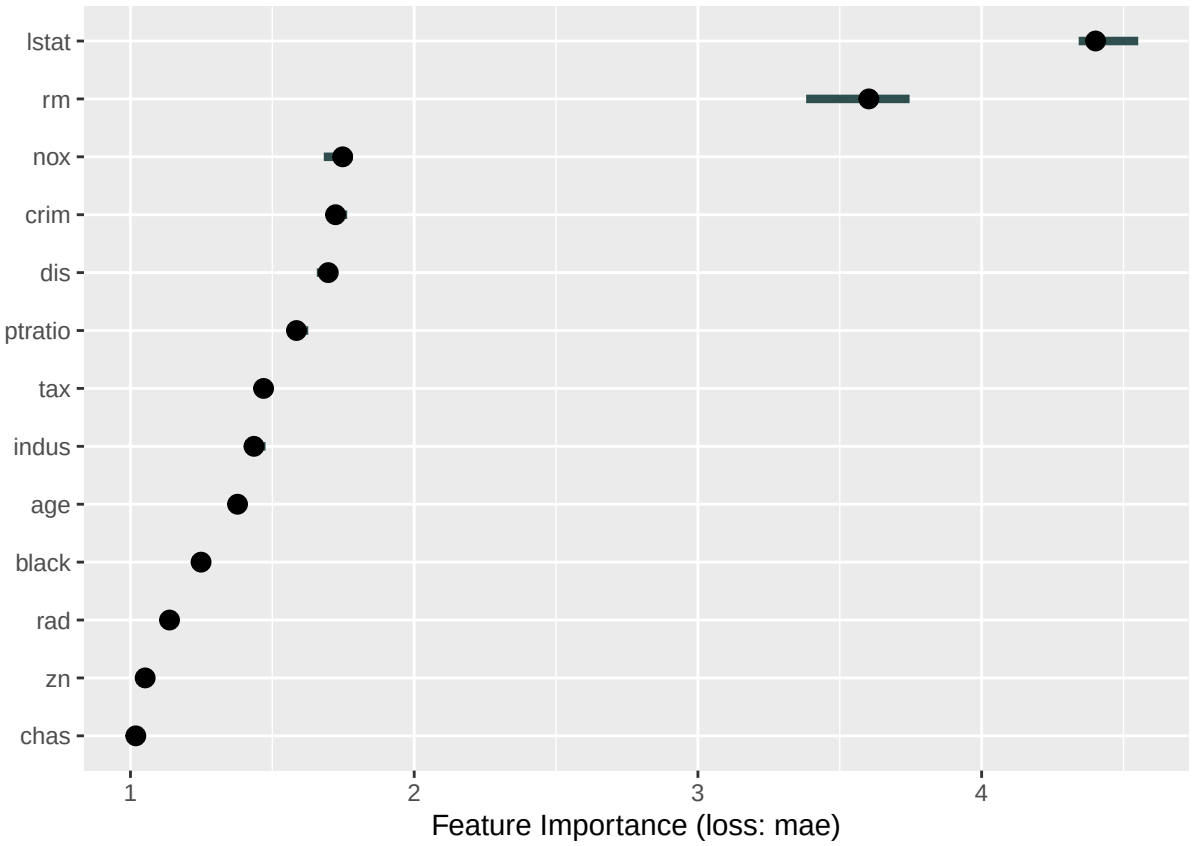
Once the predictor is created, we will use this R6 class to explore the power of the `iml` package.

# 1. Feature Importance

The function `FeatureImp` shows a measure of how important each feature was for the prediction. It shuffle each feature on the prediction measuring how much the performance drops. The way it does is up to you. In the example below, I choose to measure the loss in performance by the mean absolute error ('mae'). One could, for instance, measure the drop by the mean square error ('mse').

Thus, let's create the object `imp` using the `FeatureImp` function on our `predictor` R6 class. Then, we call the `plot()` function to look at the results in a data.frame:

```
imp <- FeatureImp$new(predictor, loss = "mae")

library(ggplot2)

plot(imp)
```




Looking at the results:

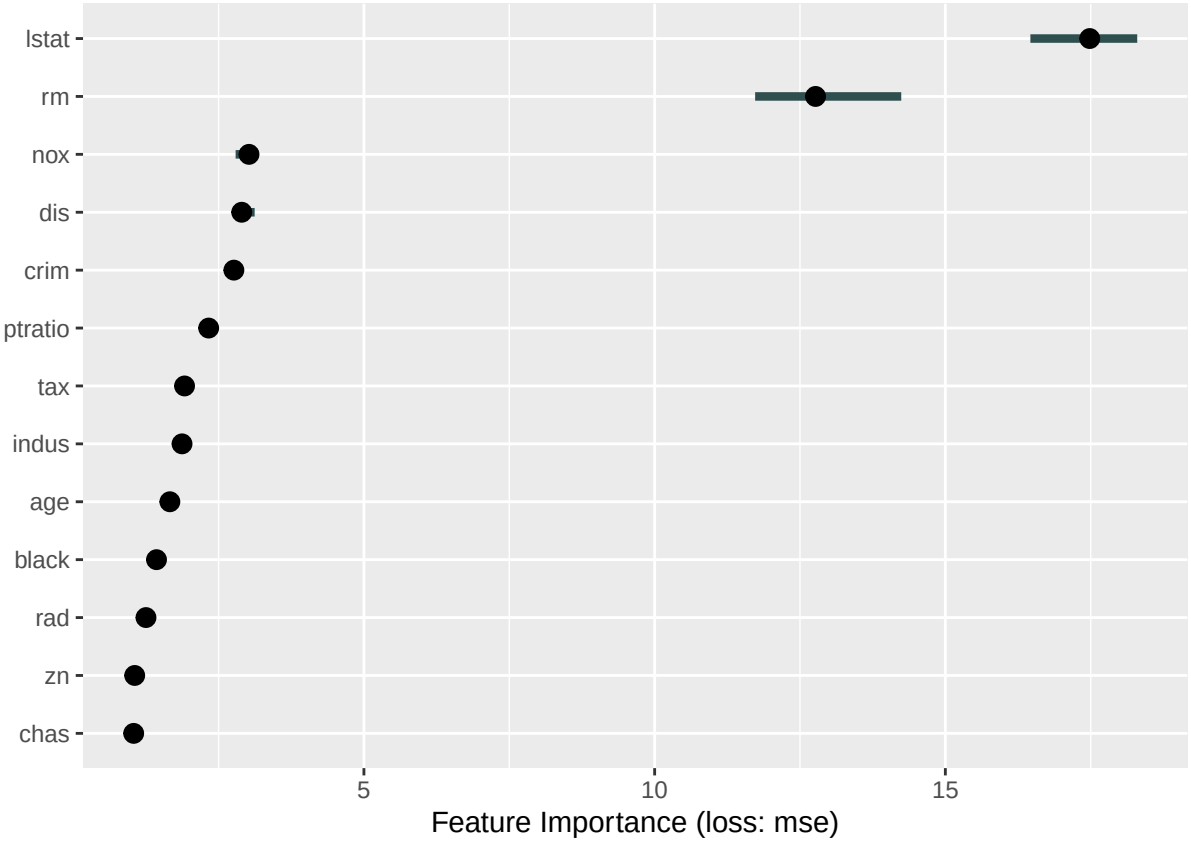
```
imp$results
```

```
##    feature importance.05 importance importance.95 permutation.error
## 1    lstat      4.341962   4.401540      4.551845          4.359104
## 2       rm      3.381429   3.602421      3.746106          3.567690
## 3      nox      1.681345   1.747909      1.782312          1.731057
## 4     crim      1.691240   1.722756      1.763857          1.706146
## 5      dis      1.656761   1.697176      1.713095          1.680814
## 6  ptratio      1.567830   1.585116      1.626562          1.569833
## 7      tax      1.453067   1.469143      1.499579          1.454979
## 8    indus      1.407134   1.435271      1.476127          1.421433
## 9      age      1.346282   1.377350      1.382774          1.364071
## 10   black      1.235182   1.248831      1.251088          1.236791
## 11     rad      1.135840   1.137611      1.147442          1.126643
## 12      zn      1.047159   1.051767      1.055649          1.041626
## 13    chas      1.013249   1.018770      1.025039          1.008948
```

That is, `lstat` seems to be the variable that, after permuting it, increases the mean absolute error of prediction by a factor 4.4, with CI between 4.34 and 4.55. The feature with lowest importance is `chas`, with a factor equals to 1 (no change in mean absolute error)

If we do the same using the `mse` argument:

```r
imp_mse <- FeatureImp$new(predictor, loss = "mse")

plot(imp_mse)
```




Results are similar compared to `mae`.

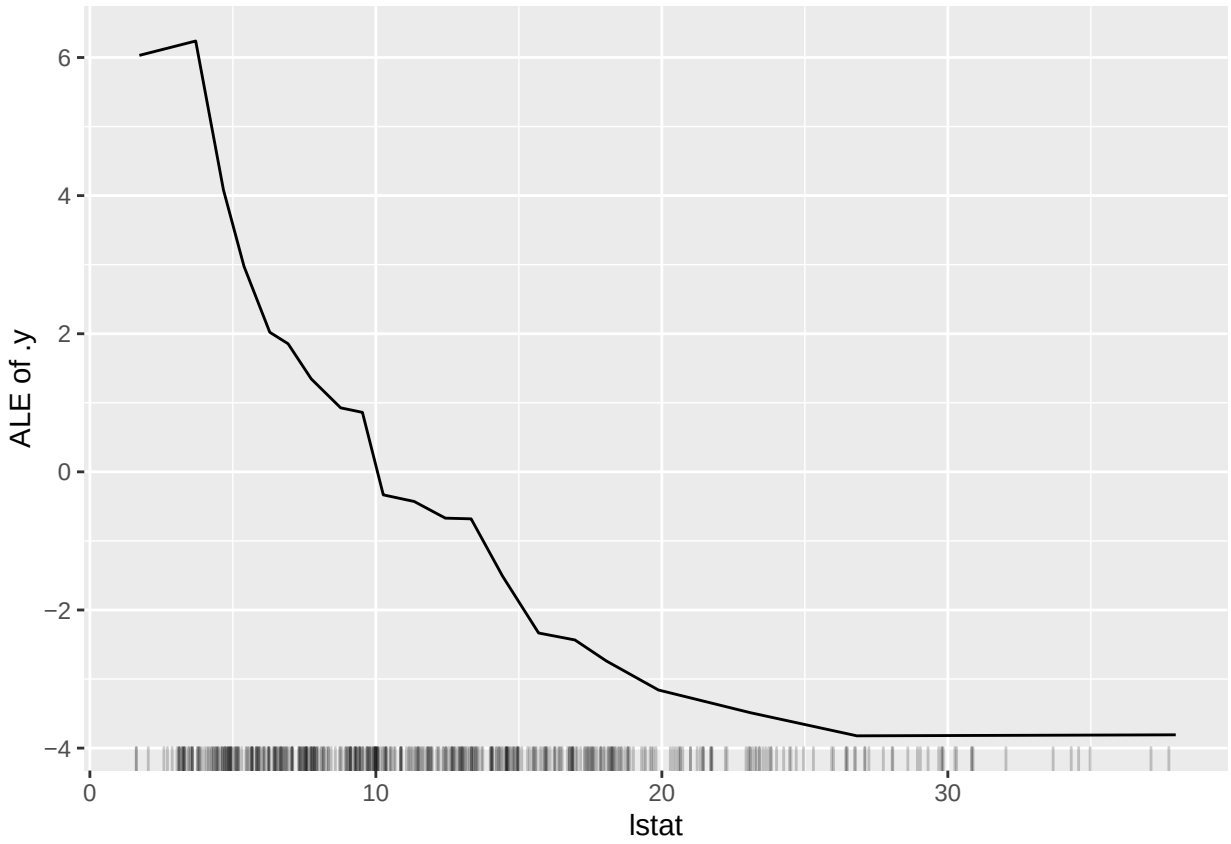
It provides nice interpretation, by splitting how each feature increases the model error when its information is dropped.

## 2. Feature Effects

Now we are interested to see how the features influence the predicted outcome. The function `FeatureEffect` implements accumulated local effect plots, partial dependence plots and individual condition expectation curves.

Let's start with the ALE (accumulated local effects) for the feature `lstat`. The ALE shows how the prediction changes locally, when the feature is varied:

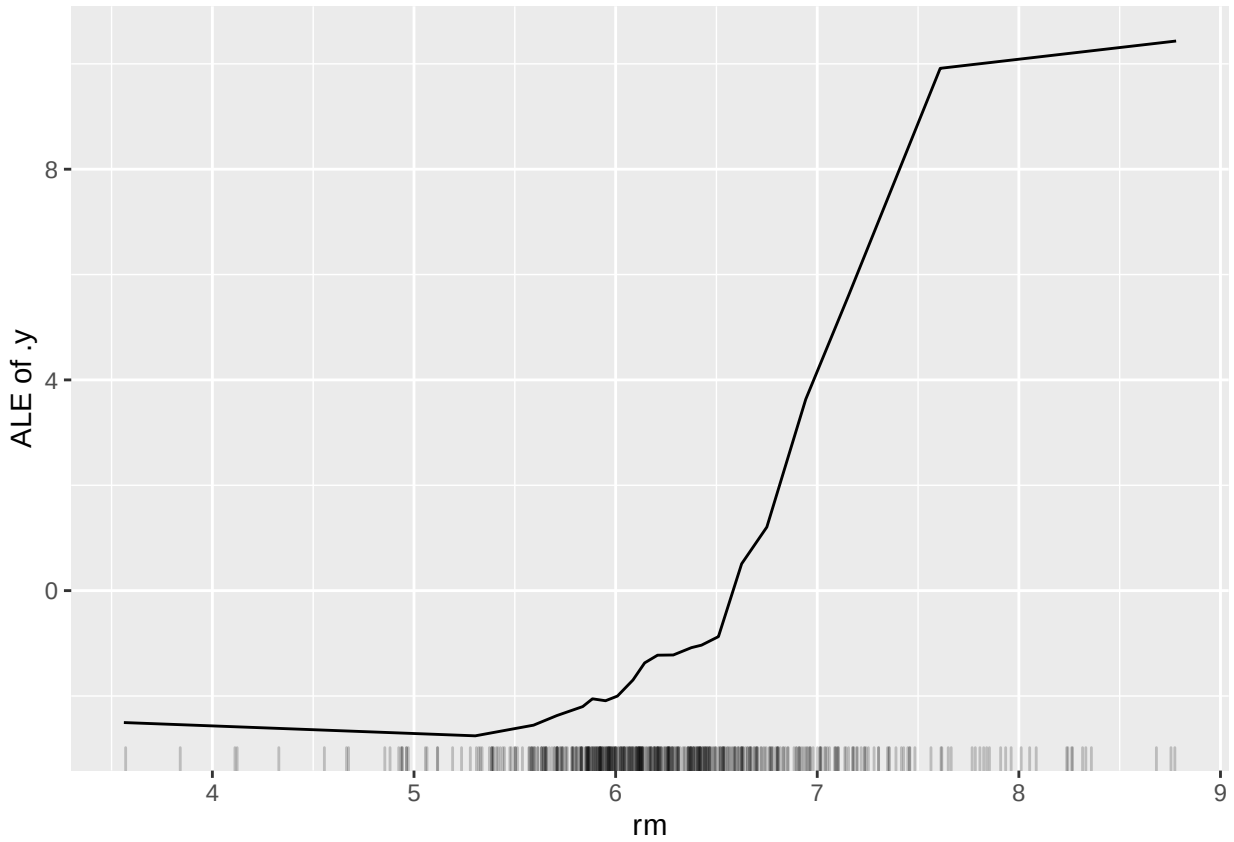
```r
ale <- FeatureEffect$new(predictor, feature = "lstat")

ale$plot()
```




X-axis is indicates the distribution of `lstat` feature, showing how relevant a region is for interpretation. We have that the feature has the majority of its observations around 10. In this point, the prediction for y is around -0.25.

We can plot the partial dependence curve on another feature like `rm` by only resetting the feature, without creating a new R6 object:

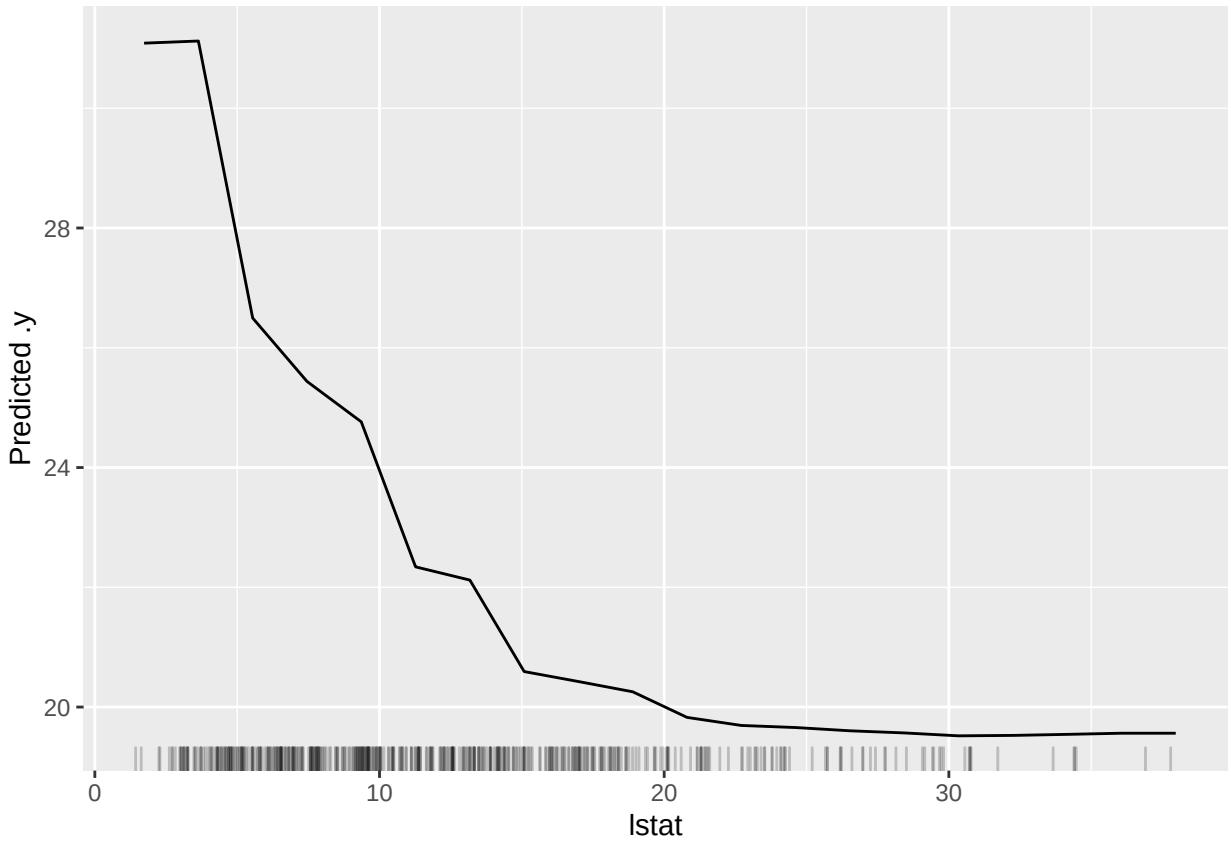
```
ale$set.feature("rm")
ale$plot()
```




That is, there is a decreasing effect of `lstat` in the prediction of y, but a negative effect for `rm`. The effect on prediction depends on the level of each covariate.

We may may want also compute the Partial Dependence Plot (PDP) instead of the ALE:

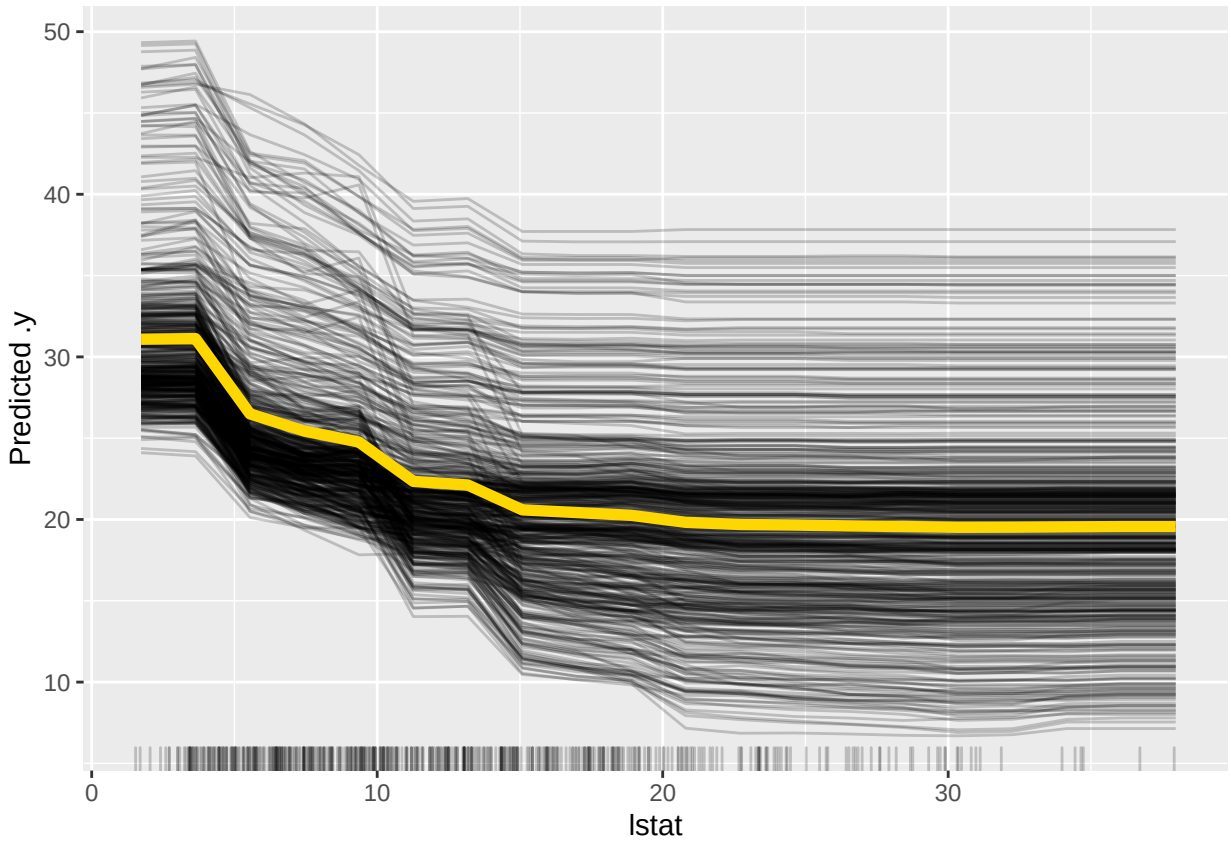
```
pdp <- FeatureEffect$new(predictor,
                         feature = "lstat",
                         method = "pdp")

pdp$plot()
```




Which is very similar to the ALE, since we have randomized data without heterogeneity. However, it may be the case that the PDP, by showing only marginal effects, hides the heterogeneous effect among covariates. Moreover, our data here is not correlated among each other, which bring value to the PDP.

Let's, for illustration, compute the Individual Conditional Expectation (ICE) together with the PDP:

```
ice <- FeatureEffect$new(predictor,
                         feature = "lstat",
                         method = "pdp+ice")

ice$plot()
```




The ICE plots display one line per instance that shows the instance's prediction changes when a feature changes. It is different from the PDP, that shows the average effect of a feature on the overall prediction. This is the PDP equivalent for individual data.

However, only the ALE predicts feature effects unbiased for the case when features are correlated with each other.

## 3. Shapley Values

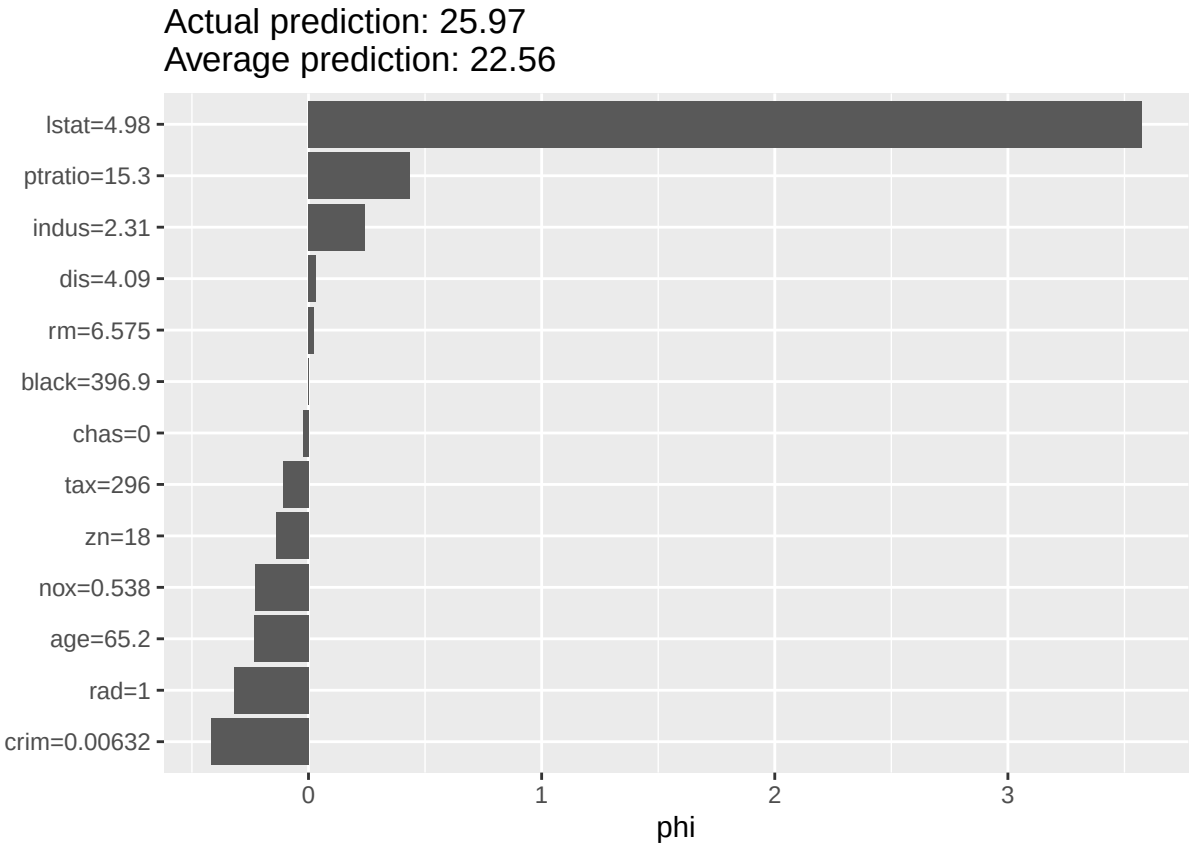
An alternative for explaining individual predictions is a method from coalitional game theory named Shapley value. Assume that for one data point, the feature values play a game together, in which they get the prediction as a payout. The Shapley value tells us how to fairly distribute the payout among the feature values.

The `iml` package does that using the function `Shapley`:

```
shapley <- Shapley$new(predictor, x.interest = X[1,])

mean(X$lstat)
```

```
## [1] 12.65306
```

```
shapley$plot()
```




That is, the average prediction is 22.56. The feature that shows higher Shapley is `lstat` which contributes 4.98 higher than the average prediction. Similar `rm` shows a 6.575 deviation from the average prediction.