Random forests two ways

D G Rossiter

2023-04-28

Table of Contents

We compare two packages that build random forests: an older package `randomForest` and a much faster implementation, `ranger`. We hope the results are similar. Note that due to randomization there will always be differences, also between runs of this script.

This document also explores Quantile Regression Forests to assess uncertainty of predictions, and local measures of variable importance.

Dataset

Packages used in this section: `sp` for the sample dataset.

```
library(sp)
```

Sample point dataset: Meuse River heavy metals in soil. This comes with `sp`.

```
data(meuse)
str(meuse)
```

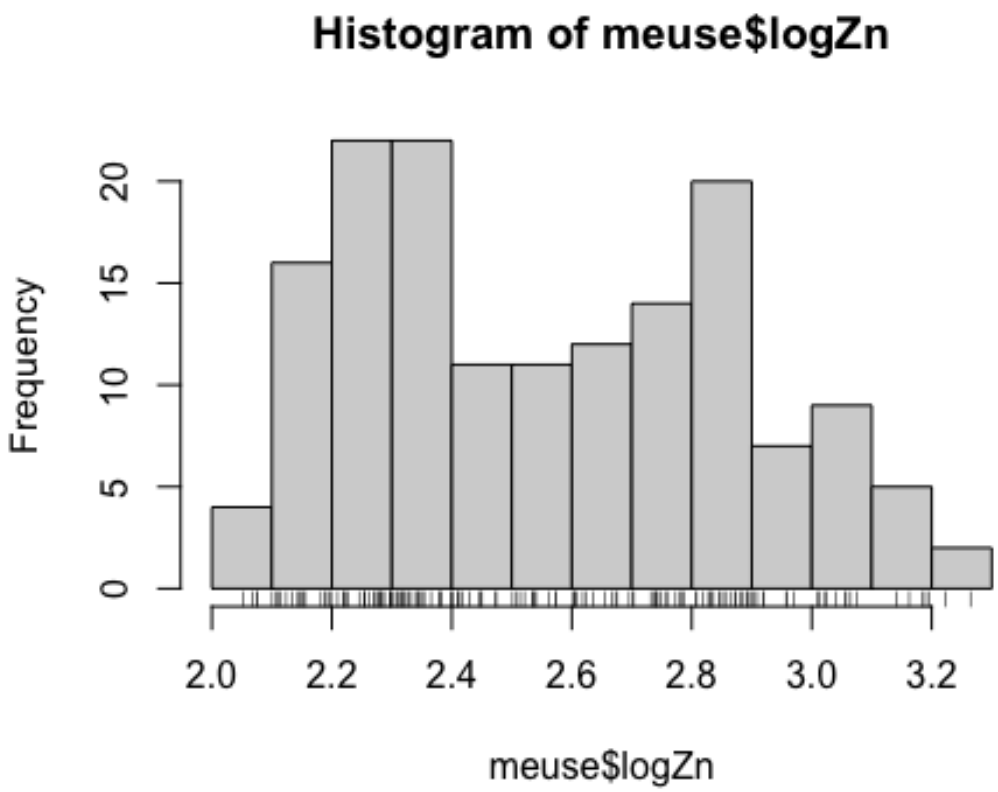
```
## 'data.frame':    155 obs. of  14 variables:
##  $ x      : num  181072 181025 181165 181298 181307 ...
##  $ y      : num  333611 333558 333537 333484 333330 ...
##  $ cadmium: num  11.7 8.6 6.5 2.6 2.8 3 3.2 2.8 2.4 1.6 ...
##  $ copper : num  85 81 68 81 48 61 31 29 37 24 ...
##  $ lead   : num  299 277 199 116 117 137 132 150 133 80 ...
##  $ zinc   : num  1022 1141 640 257 269 ...
##  $ elev   : num  7.91 6.98 7.8 7.66 7.48 ...
##  $ dist   : num  0.00136 0.01222 0.10303 0.19009 0.27709 ...
##  $ om     : num  13.6 14 13 8 8.7 7.8 9.2 9.5 10.6 6.3 ...
##  $ ffreq  : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
##  $ soil   : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
##  $ lime   : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
##  $ landuse: Factor w/ 15 levels "Aa","Ab","Ag",..: 4 4 4 11 4 11 4 2 2 15
...
##  $ dist.m : num  50 30 150 270 380 470 240 120 240 420 ...
```

We try to model one of the heavy metals (Zn) from all possibly relevant predictors:

- `ffreq` flooding frequency (3 classes)
- `x`, `y` coordinates in the RDH (Dutch) grid
- `dist.m` distance from the Meuse river (m)
- `elev` elevation above mean sea level (m)
- `soil` soil type (3 classes)
- `lime` whether agricultural lime was applied to the field (yes/no)

Make a log10-transformed copy of the Zn metal concentration to obtain a somewhat balanced distribution of the target variable:

```
meuse$logZn <- log10(meuse$zinc)    # 锌
hist(meuse$logZn); rug(meuse$logZn)
```



Random forest with `randomForest`

Packages used in this section: `randomForest` for random forests, `randomForestExplainer` for some diagnostics.

```
library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

library(randomForestExplainer)

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

Build the forest

First build the forest, using the `randomForest` function:

```
set.seed(314)
m.lzn.rf <- randomForest(logZn ~ ffreq + x + y + dist.m + elev + soil + lime,
```

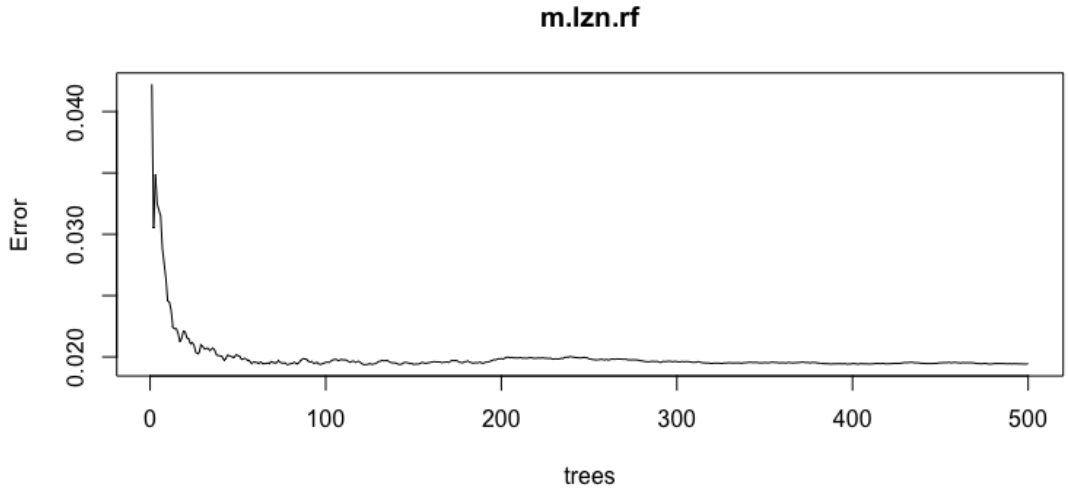
```
data=meuse,
                          # three permutations per tree to estimate importance
                          importance = TRUE, nperm = 3,
                          na.action = na.omit, mtry = 3)
print(m.lzn.rf)
```

```
##
## Call:
##  randomForest(formula = logZn ~ ffreq + x + y + dist.m + elev +      soil
+ lime, data = meuse, importance = TRUE, nperm = 3,      mtry = 3, na.action
= na.omit)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 0.01944143
##                     % Var explained: 80.09
```

The `mtry` optional argument gives the number of variables randomly sampled as candidates at each split. By default this is $\lfloor p/3 \rfloor$ where p is the number of predictors, here 5, so $\lfloor 5/3 \rfloor = 1$. We increase this to 3 to get better trees but still include weak predictors; this is matched for `ranger` (below).

Display the cross-validation error rate against the number of trees:

```
plot(m.lzn.rf)
```



Variable importance

Examine the variable importance.

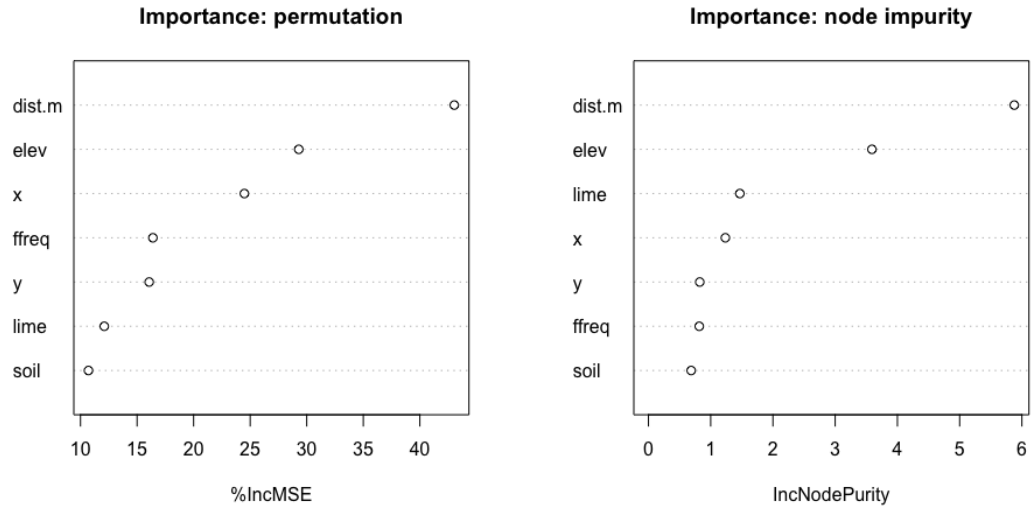
There are two kinds. From the help text:

1. “The first measure is computed from permuting OOB data: For each tree, the prediction error on the out-of-bag portion of the data is recorded (error rate for classification, MSE for regression). Then the same is done after permuting each predictor variable. The difference between the two are then averaged over all trees, and normalized by the standard deviation of the differences.”
2. “The second measure is the total decrease in node impurities from splitting on the variable, averaged over all trees. For classification, the node impurity is measured by the Gini index. For regression, it is measured by residual sum of squares.”

Here we have a regression, so Type 2 is the change in RSS due to the permutation.

We compare both:

```
randomForest::importance(m.lzn.rf, type=1)

##         %IncMSE
## ffreq  16.39898
## x      24.48868
## y      16.07052
## dist.m 43.04675
## elev   29.29422
## soil   10.70230
## lime   12.10099

randomForest::importance(m.lzn.rf, type=2)

##        IncNodePurity
## ffreq      0.8163475
## x          1.2357519
## y          0.8243707
## dist.m     5.8781655
## elev       3.5909503
## soil       0.6864054
## lime       1.4683801

par(mfrow = c(1, 2))
varImpPlot(m.lzn.rf, type=1, main = "Importance: permutation")
varImpPlot(m.lzn.rf, type=2, main = "Importance: node impurity")
```



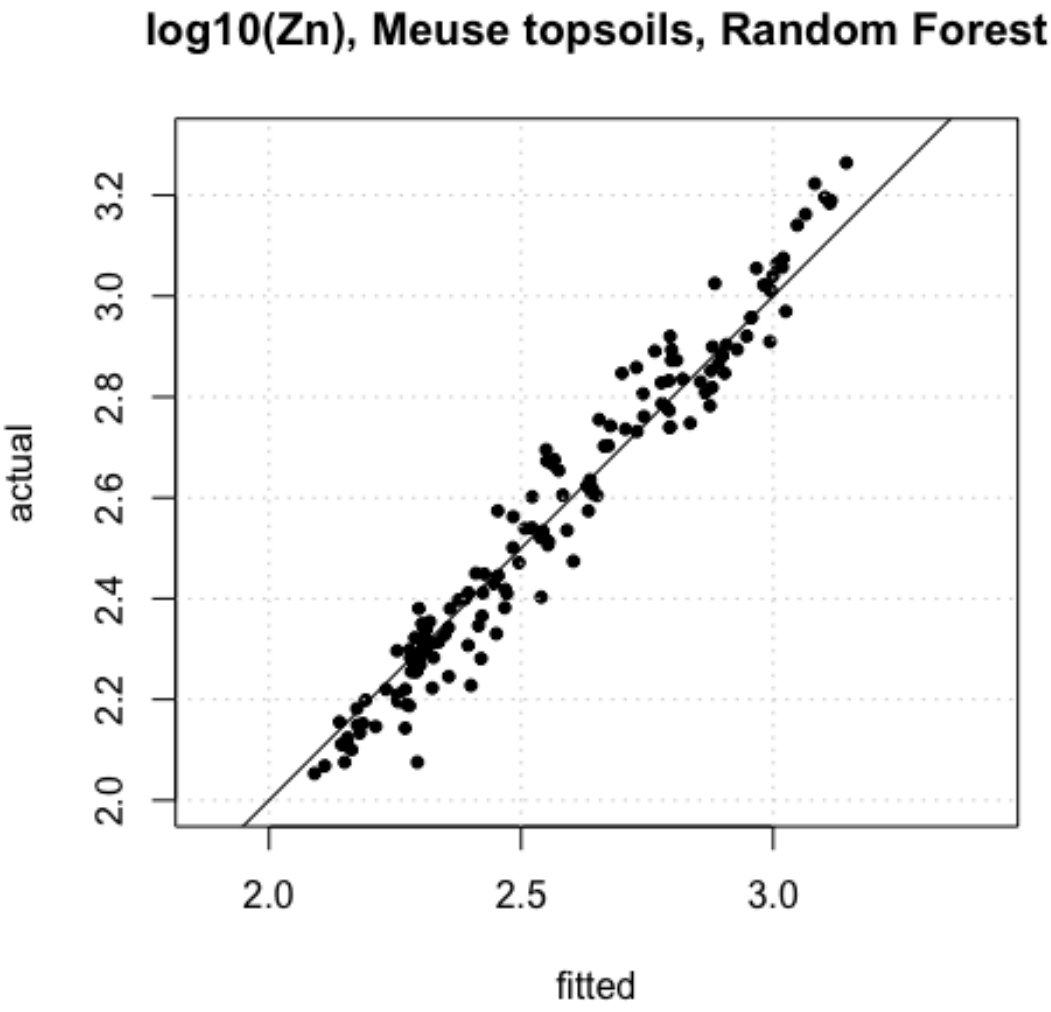

```
par(mfrow = c(1, 1))
```

Similar but not identical. After the first two `dist.m` and `elev` (obviously the most important) the others are not in the same order. There is likely substitution.

Goodness-of-fit

Predict back onto the known points, and evaluate the goodness-of-fit:

```
p.rf <- predict(m.lzn.rf, newdata=meuse)
length(unique(p.rf))

## [1] 155

summary(r.rpp <- meuse$logZn - p.rf)

##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.218752 -0.042530 -0.009149 -0.001336  0.037968  0.147015

(rmse.rf <- sqrt(sum(r.rpp^2)/length(r.rpp)))

## [1] 0.06617913

plot(meuse$logZn ~ p.rf, asp=1, pch=20, xlab="fitted", ylab="actual",
xlim=c(2,3.3),          ylim=c(2,3.3), main="log10(Zn), Meuse topsoils,
Random Forest")
grid(); abline(0,1)
```



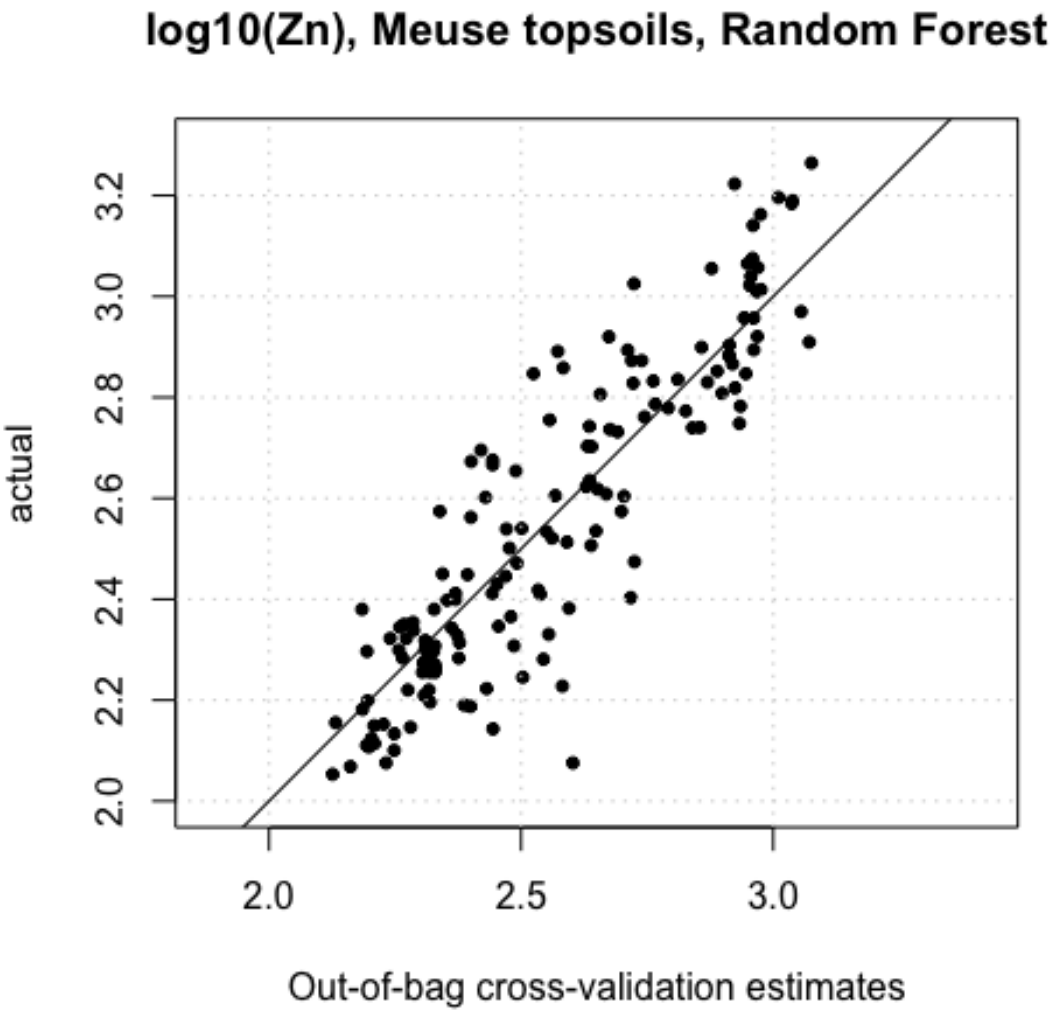
Quite a good internal fit.

Out-of bag cross-validation

A more realistic view of the predictive power of the model is the out-of-bag validation. We get this with `predict` with no dataset specified, i.e., it's the one used to build the model.

```
p.rf.oob <- predict(m.lzn.rf)
summary(r.rpp.oob <- meuse$logZn - p.rf.oob)

##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.526937 -0.090441 -0.020150 -0.003373  0.071314  0.322627

(rmse.oob <- sqrt(sum(r.rpp.oob^2)/length(r.rpp.oob)))

## [1] 0.1394325
```

```
plot(meuse$logZn ~ p.rf.oob, asp=1, pch=20,
     xlab="Out-of-bag cross-validation estimates",
     ylab="actual", xlim=c(2,3.3), ylim=c(2,3.3),
     main="log10(Zn), Meuse topsoils, Random Forest")
grid()
abline(0,1)
```

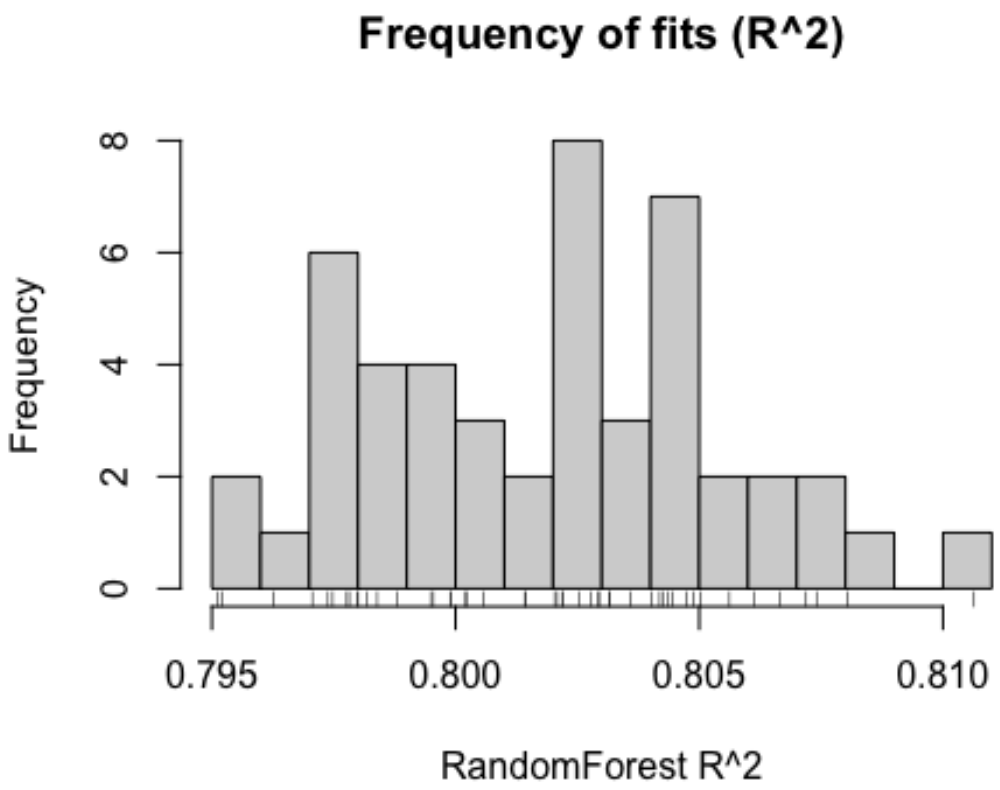


Sensitivity

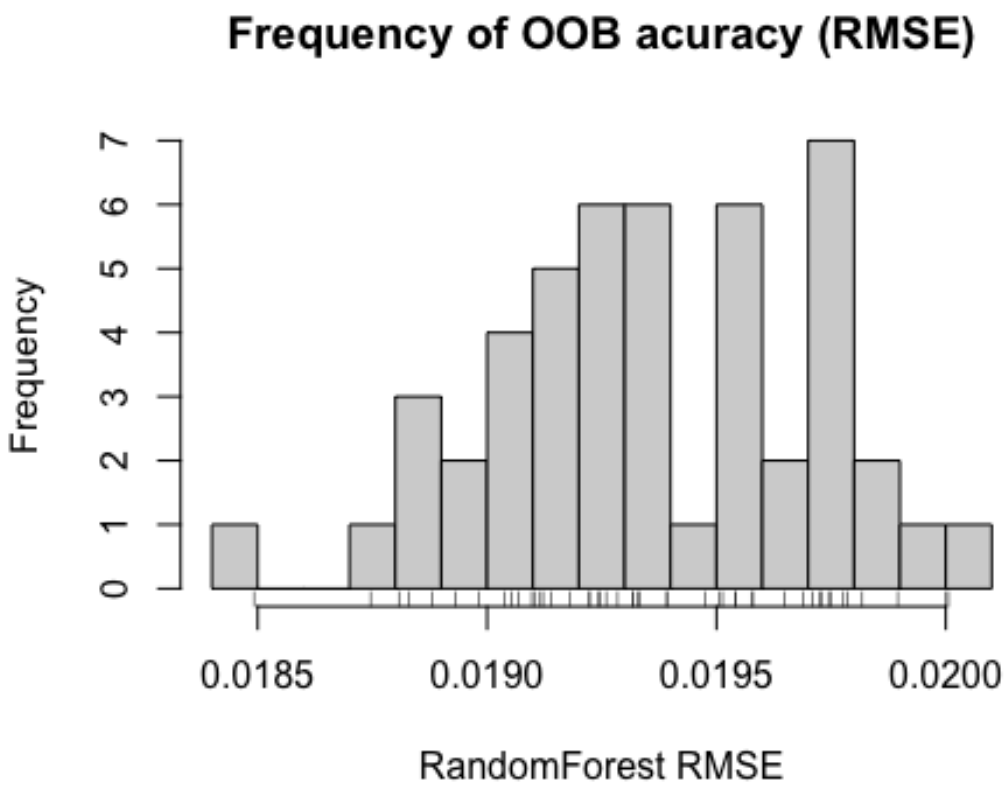
Compute the RF several times and collect statistics. This also allows a speed comparison.

```
n <- 48
rf.stats <- data.frame(rep=1:10, rsq=as.numeric(NA), mse=as.numeric(NA))
system.time(
  for (i in 1:n) {
    model.rf <- randomForest(logZn ~ ffreq + x + y + dist.m + elev + soil +
lime,
```

```
                              data=meuse, importance=T, na.action=na.omit,
mtry=5)
    summary(model.rf$rsq)
    summary(model.rf$mse)
    rf.stats[i, "rsq"] <- median(summary(model.rf$rsq))
    rf.stats[i, "mse"] <- median(summary(model.rf$mse))
  }
)

##    user  system elapsed
##   6.165   0.053   6.221

summary(rf.stats[,2:3])

##       rsq              mse
##  Min.   :0.7951   Min.   :0.01849
##  1st Qu.:0.7987   1st Qu.:0.01911
##  Median :0.8021   Median :0.01932
##  Mean   :0.8018   Mean   :0.01936
##  3rd Qu.:0.8043   3rd Qu.:0.01966
##  Max.   :0.8106   Max.   :0.02001

hist(rf.stats[,"rsq"], xlab="RandomForest R^2", breaks = 16, main =
"Frequency of fits (R^2)")
rug(rf.stats[,"rsq"])
```




```
hist(rf.stats[,"mse"], xlab="RandomForest RMSE", breaks = 16, main =
"Frequency of OOB acuracy (RMSE)")
rug(rf.stats[,"mse"])
```



As usual, much more scatter with the OOB cross-validation than with the fit.

Random forest with ranger

Packages used in this section: `ranger` for random forests and `vip` for variable importance (can be used for many model types, see https://koalaverse.github.io/vip/articles/vip.html).

The `mtry` optional argument gives the number of variables randomly sampled as candidates at each split. By default for `ranger` this is $\lfloor\sqrt{p}\rfloor$ where p is the number of predictors, here 5, so $\lfloor\sqrt{p}\rfloor = 2$. We want to try all the variables at each split for more accurate splitting. We increase this to 3 to get better trees but still include weak predictors; this is matched for `randomForest` (above).

Build the forest

First build the forest, using the `ranger` function. Ask for the `permutation` measure of variabile importance, to match `randomForest`.

The variable importance measures are of several types:

1. "impurity": variance of the responses for regression (as here)
2. "impurity_corrected": "The 'impurity_corrected' importance measure is unbiased in terms of the number of categories and category frequencies" – not relevant for regression forests
3. "permutation". For this one can specify `scale.permutation.importance = TRUE`, this should match the `randomForest` concept.

Compute two ways, for the two kinds of importance. Use the same random seed, the forests will then be identical.

```
require(ranger)
```

```
## Loading required package: ranger
```

```
##
## Attaching package: 'ranger'
```

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

```
set.seed(314)
m.lzn.ra <- ranger(logZn ~ ffreq + x + y + dist.m + elev + soil + lime,
                   data = meuse,
                   importance = 'permutation',
                   scale.permutation.importance = TRUE,
                   mtry = 3)
print(m.lzn.ra)
```

```
## Ranger result
##
## Call:
##  ranger(logZn ~ ffreq + x + y + dist.m + elev + soil + lime, data = meuse,
importance = "permutation", scale.permutation.importance = TRUE,      mtry =
3)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      155
## Number of independent variables:  7
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         permutation
## Splitrule:                        variance
## OOB prediction error (MSE):       0.01861645
## R squared (OOB):                  0.8105926
```

```
set.seed(314)
m.lzn.ra.i <- ranger(logZn ~ ffreq + x + y + dist.m + elev + soil + lime,
                   data = meuse,
                   importance = 'impurity',
```

```
                    mtry = 3)
print(m.lzn.ra.i)

## Ranger result
##
## Call:
##  ranger(logZn ~ ffreq + x + y + dist.m + elev + soil + lime, data = meuse,
importance = "impurity", mtry = 3)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      155
## Number of independent variables:  7
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.01861645
## R squared (OOB):                  0.8105926
```

Goodness-of-fit:

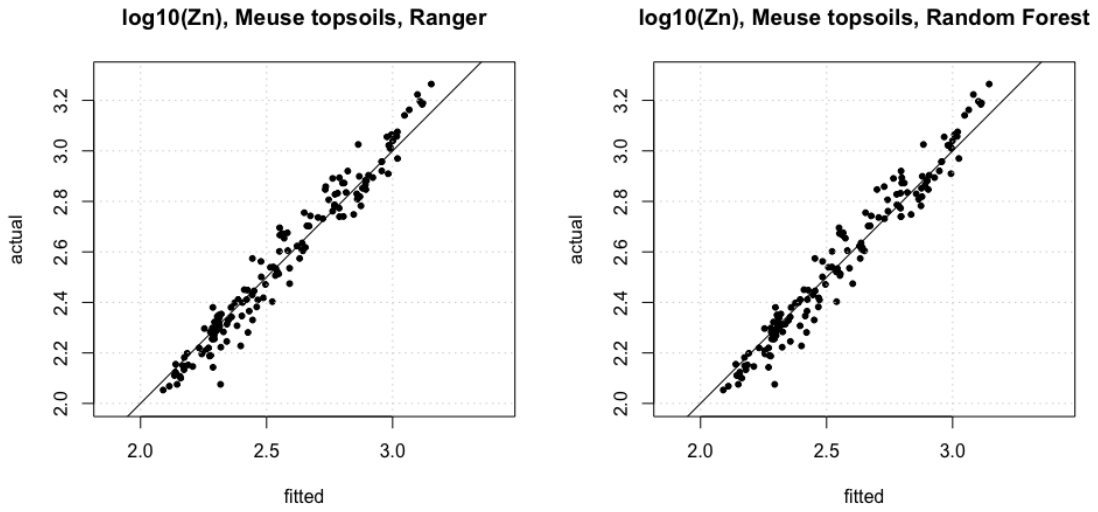
Predict with fitted model, at all observations;for this we need to specify a `data=` argument.

```
p.ra <- predict(m.lzn.ra, data=meuse)
str(p.ra)

## List of 5
##  $ predictions              : num [1:155] 2.99 3.02 2.75 2.47 2.44 ...
##  $ num.trees                : num 500
##  $ num.independent.variables: num 7
##  $ num.samples              : int 155
##  $ treetype                 : chr "Regression"
##  - attr(*, "class")= chr "ranger.prediction"

summary(r.rap <- meuse$logZn - p.ra$predictions)

##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.2419633 -0.0382925 -0.0055786 -0.0006077  0.0380941  0.1616413

(rmse.ra <- sqrt(sum(r.rap^2)/length(r.rap)))

## [1] 0.06501253
```

Compare with `RandomForest`:

```
c(rmse.ra, rmse.rf)

## [1] 0.06501253 0.06617913

par(mfrow=c(1,2))
plot(meuse$logZn ~ p.ra$predictions, asp=1, pch=20, xlab="fitted",
```

```
ylab="actual", xlim=c(2,3.3),          ylim=c(2,3.3), main="log10(Zn), Meuse
topsoils, Ranger")
grid(); abline(0,1)
plot(meuse$logZn ~ p.rf, asp=1, pch=20, xlab="fitted", ylab="actual",
xlim=c(2,3.3),          ylim=c(2,3.3), main="log10(Zn), Meuse topsoils,
Random Forest")
grid(); abline(0,1)
```




```
par(mfrow=c(1,1))
```

Almost identical.

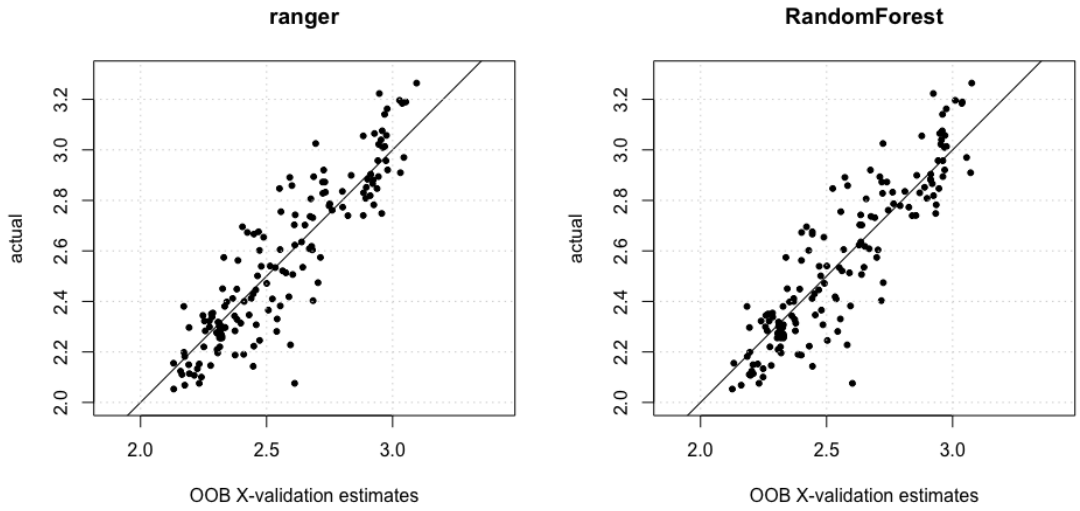
Out-of-bag cross-validation

The default model already has the OOB predictions stored in it. Compare to `RandomForest` results.

```
summary(m.lzn.ra$predictions)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.131   2.319   2.523   2.558   2.749   3.095

summary(p.rf.oob)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.126   2.323   2.524   2.560   2.764   3.076

summary(m.lzn.ra$predictions - p.rf.oob)  # difference

##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.044479 -0.013175 -0.002185 -0.001946  0.009626  0.054540
```

`ranger` has slightly lower OOB predictions than `randomForest`.`

```
par(mfrow=c(1,2))
plot(meuse$logZn ~ m.lzn.ra$predictions, asp=1, pch=20,
     ylab="actual", xlab="OOB X-validation estimates",
    xlim=c(2,3.3), ylim=c(2,3.3),
    main="ranger")
abline(0,1); grid()
plot(meuse$logZn ~ p.rf.oob, asp=1, pch=20,
     xlab="OOB X-validation estimates",
     ylab="actual", xlim=c(2,3.3), ylim=c(2,3.3),
     main="RandomForest")
grid(); abline(0,1)
```




```
par(mfrow=c(1,1))
```

Very similar. The same points are poorly-predicted by both models.

Variable importance:

First, for permutation:

```
cbind(ranger = ranger::importance(m.lzn.ra),
      rF = randomForest::importance(m.lzn.rf)[,1])
```

```
##          ranger       rF
## ffreq  16.79039 16.39898
## x      25.29951 24.48868
## y      15.34952 16.07052
## dist.m 40.71367 43.04675
## elev   29.94610 29.29422
## soil   10.32160 10.70230
## lime   14.86976 12.10099
```

Second, for impurity:

```
cbind(ranger =ranger::importance(m.lzn.ra.i),
      rF = randomForest::importance(m.lzn.rf)[,2])
```

```
##           ranger        rF
## ffreq  0.7748065 0.8163475
## x      1.2215072 1.2357519
## y      0.9253974 0.8243707
## dist.m 5.7685564 5.8781655
## elev   3.6611503 3.5909503
## soil   0.6664583 0.6864054
## lime   1.5780326 1.4683801
```

Graph these:

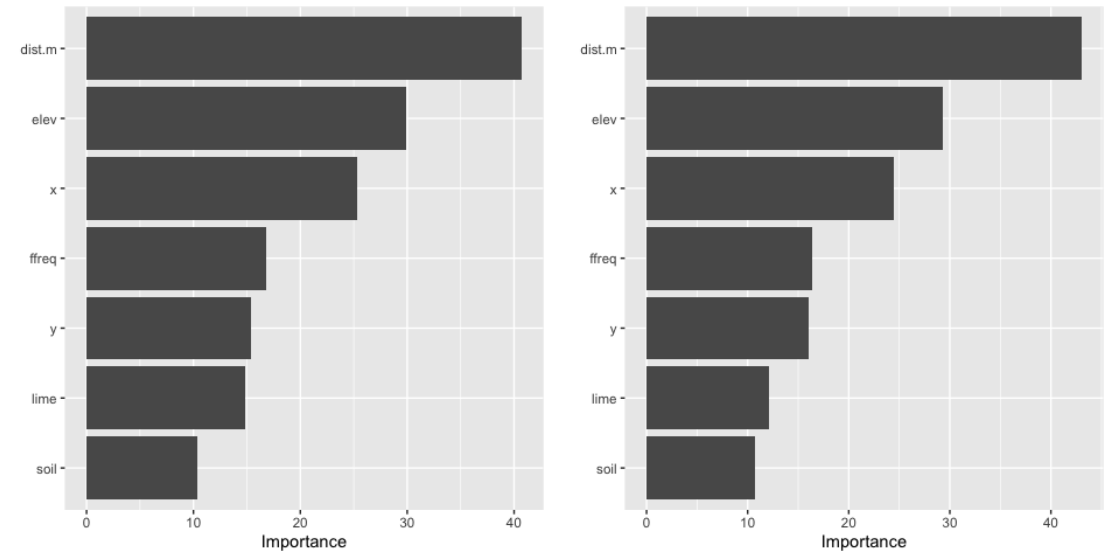
```
require(vip)
```

```
## Loading required package: vip
```

```
##
## Attaching package: 'vip'
```

```
## The following object is masked from 'package:utils':
##
##     vi
```

```
v1 <- vip(m.lzn.ra, title = "Ranger")
v2 <- vip(m.lzn.rf, title = "randomForest")
grid.arrange(v1, v2, ncol=2)
```



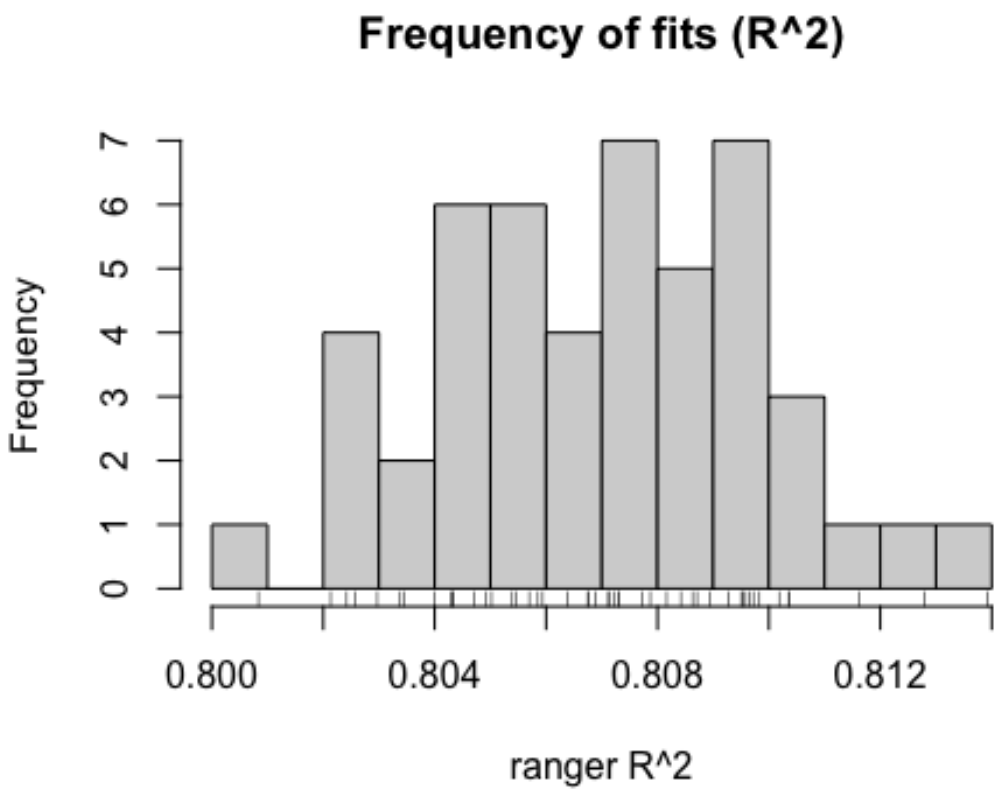
Definitely some differences here.

I don't know how to show the impurity for `randomForest` with `vip`.

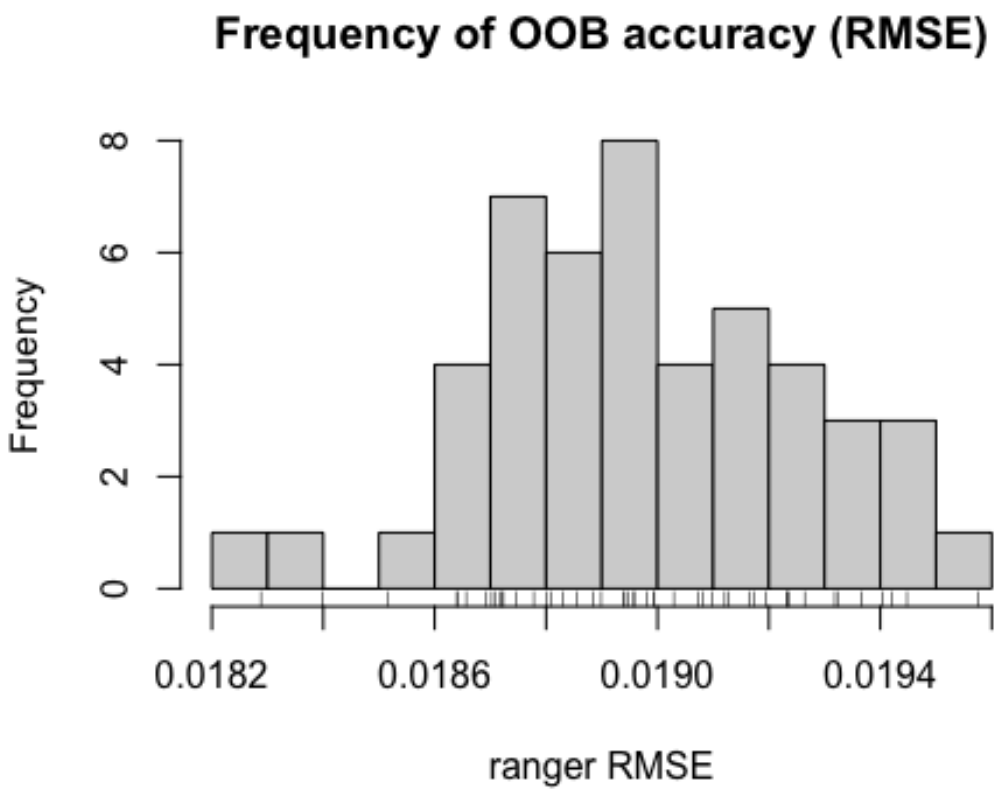
Sensitivity

Compute the ranger RF several times and collect statistics:

```
n <- 48
ra.stats <- data.frame(rep=1:10, rsq=as.numeric(NA), mse=as.numeric(NA))
system.time(
  for (i in 1:n) {
    model.ra <- ranger(logZn ~ ffreq + x + y + dist.m + elev + soil + lime,
                       data=meuse, importance="none", mtry=5,
                       write.forest = FALSE)
    ra.stats[i, "mse"] <- model.ra$prediction.error
    ra.stats[i, "rsq"] <- model.ra$r.squared
  }
)

##    user  system elapsed
##   2.097   0.206   0.451

summary(ra.stats[,2:3])

##       rsq              mse
##  Min.   :0.8008   Min.   :0.01829
##  1st Qu.:0.8049   1st Qu.:0.01874
##  Median :0.8071   Median :0.01896
##  Mean   :0.8070   Mean   :0.01897
##  3rd Qu.:0.8093   3rd Qu.:0.01918
##  Max.   :0.8139   Max.   :0.01958

hist(ra.stats[,"rsq"], xlab="ranger R^2", breaks = 16, main = "Frequency of
fits (R^2)")
rug(ra.stats[,"rsq"])
```




```
hist(ra.stats[,"mse"], xlab="ranger RMSE", breaks = 16, main = "Frequency of
OOB accuracy (RMSE)")
rug(ra.stats[,"mse"])
```



Compare the statistics to `RandomForest`:

```
summary(ra.stats[,2:3])
```

```
##       rsq              mse         
##  Min.   :0.8008   Min.   :0.01829  
##  1st Qu.:0.8049   1st Qu.:0.01874  
##  Median :0.8071   Median :0.01896  
##  Mean   :0.8070   Mean   :0.01897  
##  3rd Qu.:0.8093   3rd Qu.:0.01918  
##  Max.   :0.8139   Max.   :0.01958
```

```
summary(rf.stats[,2:3])
```

```
##       rsq              mse         
##  Min.   :0.7951   Min.   :0.01849  
##  1st Qu.:0.7987   1st Qu.:0.01911  
##  Median :0.8021   Median :0.01932  
##  Mean   :0.8018   Mean   :0.01936  
##  3rd Qu.:0.8043   3rd Qu.:0.01966  
##  Max.   :0.8106   Max.   :0.02001
```

```
sd(ra.stats[,2])
```

```
## [1] 0.002898525
```

```
sd(rf.stats[,2])
```

```
## [1] 0.003582076
```

```
sd(ra.stats[,3])
```

```
## [1] 0.0002848899
```

```
sd(rf.stats[,3])
```

```
## [1] 0.0003498031
```

No clear pattern as to which is more sensitive.

Conclusion

In this simple example the model fits and OOB statistics are a bit better with `ranger`. The importance of variables is quite different, which may be the implementation of the importance estimation.

As expected, `ranger` is much faster.

Quantile Regression Forest

With `ranger`. Specify `quantreg = TRUE`, also to keep the in-bag fits to use in the quantile predictions, with `keep.inbag = TRUE`.

```
m.lzn.qrf <- ranger(logZn ~ ffreq + x + y + dist.m + elev + soil + lime,
                    data = meuse,
                    quantreg = TRUE,
                    importance = 'permutation',
                    keep.inbag=TRUE,  # needed for QRF
                    scale.permutation.importance = TRUE,
                    mtry = 3)
pred.qrf <- predict(m.lzn.qrf, type = "quantiles",
                     # default is c(0.1, 0.5, 0.9)
                     quantiles = c(0.1, 0.25, 0.5, 0.75, 0.9))
summary(pred.qrf$predictions)
```

```
##  quantile= 0.1   quantile= 0.25  quantile= 0.5   quantile= 0.75
##  Min.   :2.053   Min.   :2.068   Min.   :2.114   Min.   :2.134
##  1st Qu.:2.191   1st Qu.:2.262   1st Qu.:2.304   1st Qu.:2.354
##  Median :2.262   Median :2.365   Median :2.521   Median :2.667
##  Mean   :2.369   Mean   :2.453   Mean   :2.554   Mean   :2.655
##  3rd Qu.:2.514   3rd Qu.:2.717   3rd Qu.:2.783   3rd Qu.:2.891
##  Max.   :2.905   Max.   :2.958   Max.   :3.184   Max.   :3.265
##  quantile= 0.9
```

```
##  Min.   :2.182
##  1st Qu.:2.501
##  Median :2.773
##  Mean   :2.751
##  3rd Qu.:3.025
##  Max.   :3.265
```

Local measures

Both methods provide local measures of importance, i.e., how much each variable influences an individual prediction.

randomForest

"The "local" (or casewise) variable importance for a random regression forest is the average increase in squared OOB residuals when the variable is permuted, for each observation separately. This is computed if `localImp = TRUE`.

```
set.seed(314)
m.lzn.rf.l <- randomForest(logZn ~ ffreq + x + y + dist.m + elev + soil +
lime, data=meuse,
                           localImp = TRUE, nperm = 3,
                           na.action = na.omit, mtry = 3)
dim(m.lzn.rf.l$localImportance)

## [1]   7 155
```

There is one importance measure for each variable for each observation.

View the local importance for the first few observation points:

```
m.lzn.rf.l$localImportance[, 1:6]

##                   1             2            3           4           5
## ffreq  0.025223725  0.0311664580  0.007239915 0.001172955 0.007573898
## x      0.003638953 -0.0005331196  0.000884300 0.023488953 0.020065479
## y      0.025311191  0.0087814013  0.014651997 0.010936406 0.010148829
## dist.m 0.152338528  0.1684531194  0.042116343 0.031325391 0.035634740
## elev   0.023182941  0.0330063161  0.003759453 0.011066830 0.003895781
## soil   0.012211841  0.0155922172  0.009372301 0.005682348 0.015348364
## lime   0.035701605  0.0443260269 -0.006084308 0.013544121 0.016970023
##                   6
## ffreq  0.006609679
## x      0.015141041
## y      0.012441277
## dist.m 0.063966507
## elev   0.016467674
## soil   0.008685162
## lime   0.010301643
```

ranger

For this package we use the `local.importance` option: "Calculate and return local importance values as in (Breiman 2001). Only applicable if importance is set to `permutation`". Reference: Breiman, L. (2001). Random forests. Mach Learn, 45:5-32. doi: 10.1023/A:1010933404324. This is just the same permutation measure as for the overall variable importance, but computed for each observation.

Use the same seed so that the overall model fit and importance match the first section above.

```
set.seed(314)
m.lzn.ra.l <- ranger(logZn ~ ffreq + x + y + dist.m + elev + soil + lime,
                     data = meuse,
                     importance = 'permutation',
                     local.importance = TRUE,
                     scale.permutation.importance = TRUE,
                     mtry = 3)
print(m.lzn.ra.l)
```

```
## Ranger result
##
## Call:
##  ranger(logZn ~ ffreq + x + y + dist.m + elev + soil + lime, data = meuse,
importance = "permutation", local.importance = TRUE,
scale.permutation.importance = TRUE,      mtry = 3)
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      155
## Number of independent variables:  7
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         permutation
## Splitrule:                        variance
## OOB prediction error (MSE):       0.01861645
## R squared (OOB):                  0.8105926
```

```
dim(m.lzn.ra.l$variable.importance.local)
```

```
## [1] 155   7
```

Notice the `variable.importance.local` field. This has the importance of each variable for each of the predictions.

Show the local importances for the first few observation points, compared the global importance. These are both the increase in OOB RMSE when the predictor is permuted.

```
dim(m.lzn.ra.l$variable.importance.local)
```

```
## [1] 155   7
```

```
m.lzn.ra.l$variable.importance.local[1:6,]
```

```
##         ffreq           x           y     dist.m          elev        soil
## 1 0.011068076 0.005452759 0.008899231 0.05850423  0.0099005893 0.003129961
## 2 0.015304762 0.001705302 0.004220736 0.05917265  0.0142170662 0.008075754
## 3 0.004277926 0.001818228 0.015349669 0.01411915  0.0017249558 0.001603407
## 4 0.001182663 0.004961601 0.003628138 0.01120759 -0.0006275804 0.005303960
## 5 0.002316050 0.005549051 0.003685398 0.01970707  0.0028770418 0.005360852
## 6 0.001043971 0.003847687 0.002211735 0.01152574  0.0035831139 0.001049993
##           lime
## 1  0.013524257
## 2  0.020399560
## 3 -0.002937614
## 4  0.001612304
## 5  0.008106699
## 6  0.005264946
```

```
ranger::importance(m.lzn.ra.l)
```

```
##       ffreq           x           y      dist.m        elev        soil
## 0.012904514 0.013116775 0.009516604 0.065130013 0.031742157 0.005236516
##        lime
## 0.007461143
```

Show the range of local variable importances for each predictor:

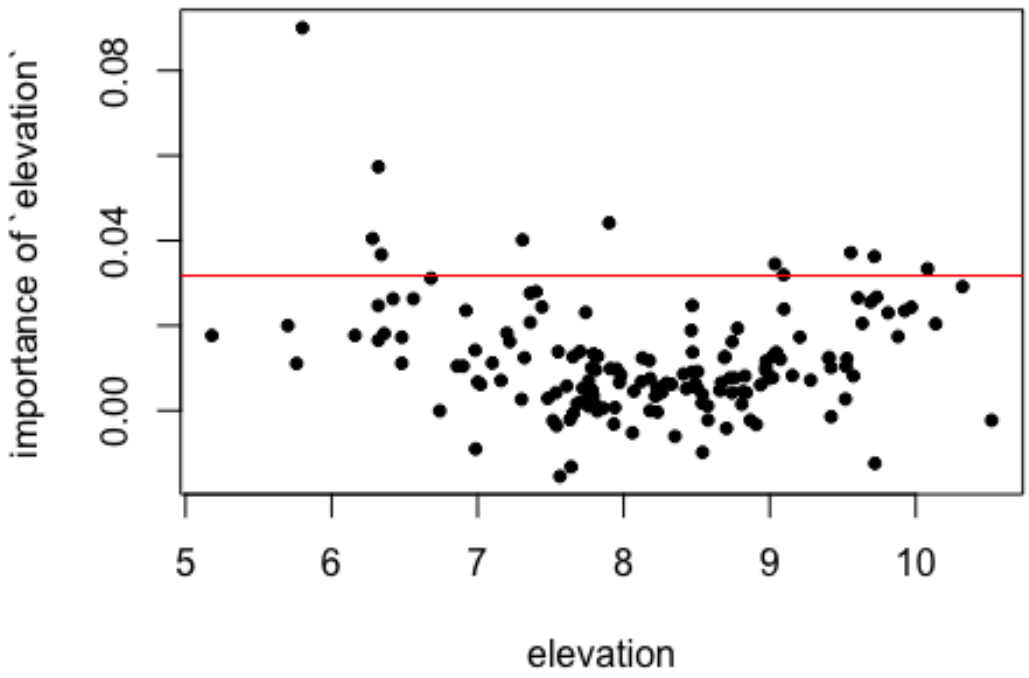
```
summary(m.lzn.ra.l$variable.importance.local[,1:7])
```

```
##      ffreq                  x                   y
##  Min.   :-0.0351568   Min.   :-0.014961   Min.   :-0.015054
##  1st Qu.: 0.0009273   1st Qu.: 0.001846   1st Qu.: 0.001168
##  Median : 0.0033483   Median : 0.004529   Median : 0.003061
##  Mean   : 0.0047760   Mean   : 0.004835   Mean   : 0.003528
##  3rd Qu.: 0.0076180   3rd Qu.: 0.007396   3rd Qu.: 0.005084
##  Max.   : 0.0311604   Max.   : 0.029015   Max.   : 0.022957
##      dist.m              elev                soil
##  Min.   :-0.029625   Min.   :-0.015450   Min.   :-0.0096089
##  1st Qu.: 0.008341   1st Qu.: 0.004190   1st Qu.: 0.0001734
##  Median : 0.020808   Median : 0.009619   Median : 0.0014919
##  Mean   : 0.023971   Mean   : 0.011747   Mean   : 0.0019462
##  3rd Qu.: 0.034208   3rd Qu.: 0.017547   3rd Qu.: 0.0031633
##  Max.   : 0.163139   Max.   : 0.090050   Max.   : 0.0098893
##       lime
##  Min.   :-0.0438236
##  1st Qu.: 0.0005832
##  Median : 0.0019866
##  Mean   : 0.0028013
##  3rd Qu.: 0.0044497
##  Max.   : 0.0478122
```

There is quite some range in the importance, even for `dist.m` and `elev`. Notice that the overall importance of these is betweem the 3rd quartile and maximum, i.e., they are quite influential in less than 1/4 of cases.

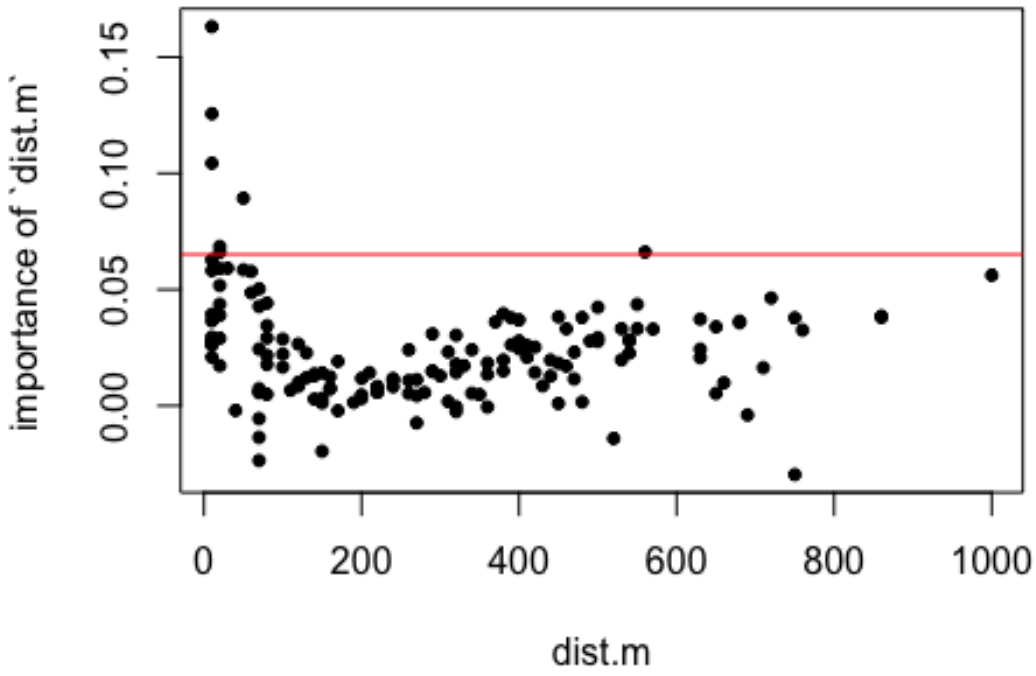
What is the relation of the importance with the factor itself? Show a scatterplot, with a horizontal line at the overall importance.

```
plot(meuse$elev,
     m.lzn.ra.l$variable.importance.local[,"elev"],
     xlab = "elevation", ylab = "importance of `elevation`",
     pch = 20)
abline(h = ranger::importance(m.lzn.ra.l)["elev"], col = "red")
```



Weak relation to no relation.

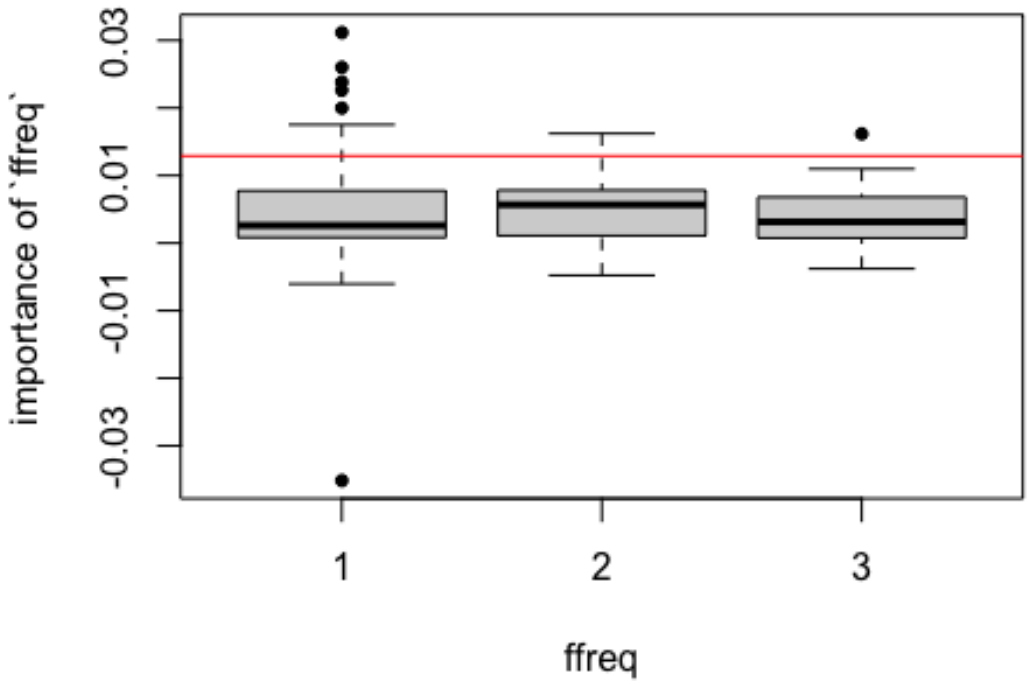
```
plot(meuse$dist.m,
     m.lzn.ra.l$variable.importance.local[,"dist.m"],
     xlab = "dist.m", ylab = "importance of `dist.m`",
     pch = 20)
abline(h = ranger::importance(m.lzn.ra.l)["dist.m"], col = "red")
```



Quite important for a few points near the river, but not most.

What about flood frequency?

```
plot(meuse$ffreq,
     m.lzn.ra.l$variable.importance.local[,"ffreq"],
     xlab = "ffreq", ylab = "importance of `ffreq`",
     pch = 20)
abline(h = ranger::importance(m.lzn.ra.l)["ffreq"], col = "red")
```



Again, important for a few points near the river.

`iml` package: Interpretable Machine Learning

Text: Molnar, C. (2022). Interpretable Machine Learning.
https://christophm.github.io/interpretable-ml-book/

```
require(iml)

## Loading required package: iml
```

Interpret one of the random forests built above.

`Predictor` object

First, make a `Predictor` object: "… holds any machine learning model (mlr, caret, randomForest, …) and the data to be used for analyzing the model. The interpretation methods in the iml package need the machine learning model to be wrapped in a Predictor object."

```
vars <- c("ffreq","x","y","dist.m","elev","soil","lime")
X <-  meuse[, vars]
```

```
predictor <- Predictor$new(model = m.lzn.ra, data = X, y = meuse$logZn)
str(predictor)
```

```
## Classes 'Predictor', 'R6' <Predictor>
##   Public:
##     batch.size: 1000
##     class: NULL
##     clone: function (deep = FALSE)
##     data: Data, R6
##     initialize: function (model = NULL, data = NULL, predict.function =
NULL,
##     model: ranger
##     predict: function (newdata)
##     prediction.colnames: NULL
##     prediction.function: function (newdata)
##     print: function ()
##     task: unknown
##   Private:
##     predictionChecked: FALSE
```

For QRF:

```
predictor.qrf <- Predictor$new(model = m.lzn.qrf, data = X, y = meuse$logZn)
str(predictor.qrf)
```

```
## Classes 'Predictor', 'R6' <Predictor>
##   Public:
##     batch.size: 1000
##     class: NULL
##     clone: function (deep = FALSE)
##     data: Data, R6
##     initialize: function (model = NULL, data = NULL, predict.function =
NULL,
##     model: ranger
##     predict: function (newdata)
##     prediction.colnames: NULL
##     prediction.function: function (newdata)
##     print: function ()
##     task: unknown
##   Private:
##     predictionChecked: FALSE
```

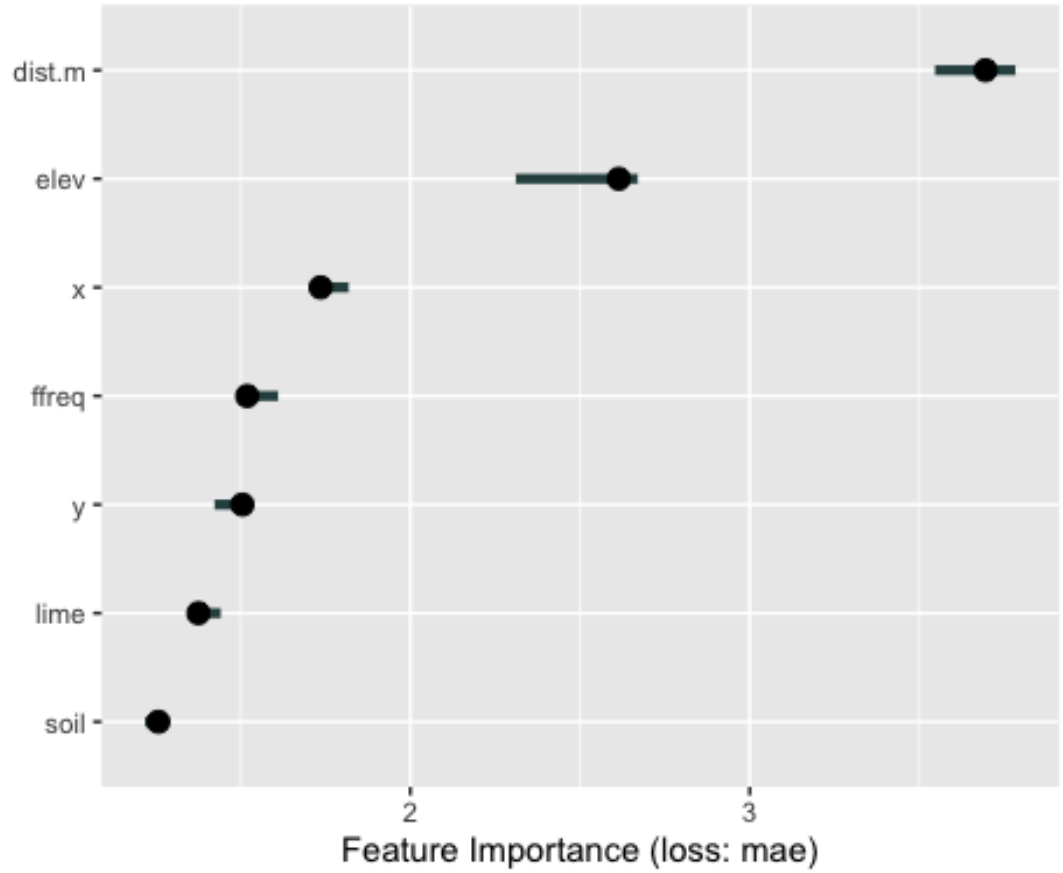
Feature importance

Feature importance, based on `mae` (mean absolute error) or `mse` (mean squared error). Plot the median and the distribution (5–95% quantiles over all the cases).

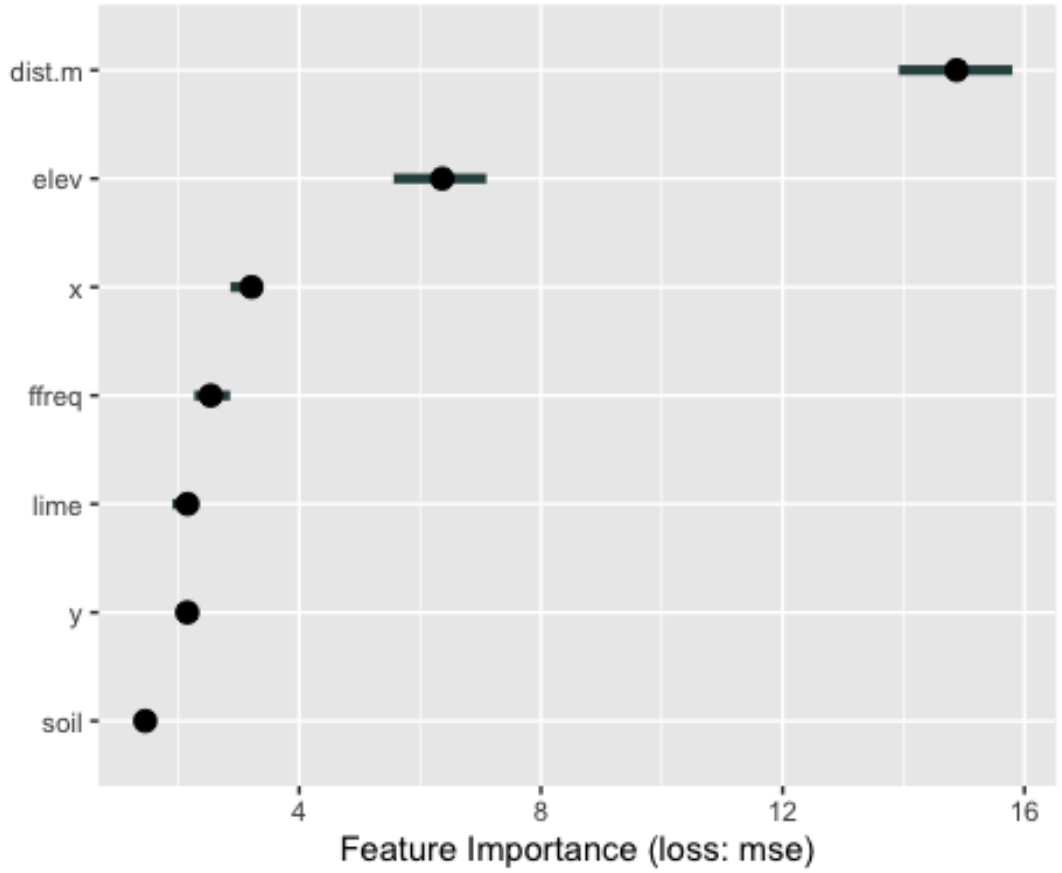
This uses `plot.FeatureImp`.

```
imp.mae <- iml::FeatureImp$new(predictor, loss = "mae")
imp.mse <- iml::FeatureImp$new(predictor, loss = "mse")
```

```
require("ggplot2")
plot(imp.mae)
```




```
plot(imp.mse)
```




```
print(imp.mae)
```

```
## Interpretation method:  FeatureImp
## error function: mae
##
## Analysed predictor:
## Prediction task: unknown
##
##
## Analysed data:
## Sampling from data.frame with 155 rows and 7 columns.
##
##
## Head of results:
##   feature importance.05 importance importance.95 permutation.error
## 1  dist.m      3.545983   3.694893      3.783031        0.18562503
## 2    elev      2.310812   2.614338      2.670165        0.13133980
## 3       x      1.716875   1.735522      1.818265        0.08718962
## 4   ffreq      1.499996   1.519180      1.610322        0.07632099
## 5       y      1.423054   1.505106      1.523621        0.07561394
## 6    lime      1.360322   1.375500      1.441221        0.06910274
```

```
print(imp.mse)
```

```
## Interpretation method:  FeatureImp
## error function: mse
##
## Analysed predictor:
## Prediction task: unknown
##
##
## Analysed data:
## Sampling from data.frame with 155 rows and 7 columns.
##
##
## Head of results:
##   feature importance.05 importance importance.95 permutation.error
## 1  dist.m     13.921234  14.877915     15.801162       0.062883429
## 2    elev      5.563535   6.367359      7.097176       0.026912465
## 3       x      2.862070   3.211992      3.338609       0.013575897
## 4   ffreq      2.263704   2.536549      2.857294       0.010721054
## 5    lime      1.903443   2.151855      2.274674       0.009095095
## 6       y      2.062338   2.148588      2.313753       0.009081283
```

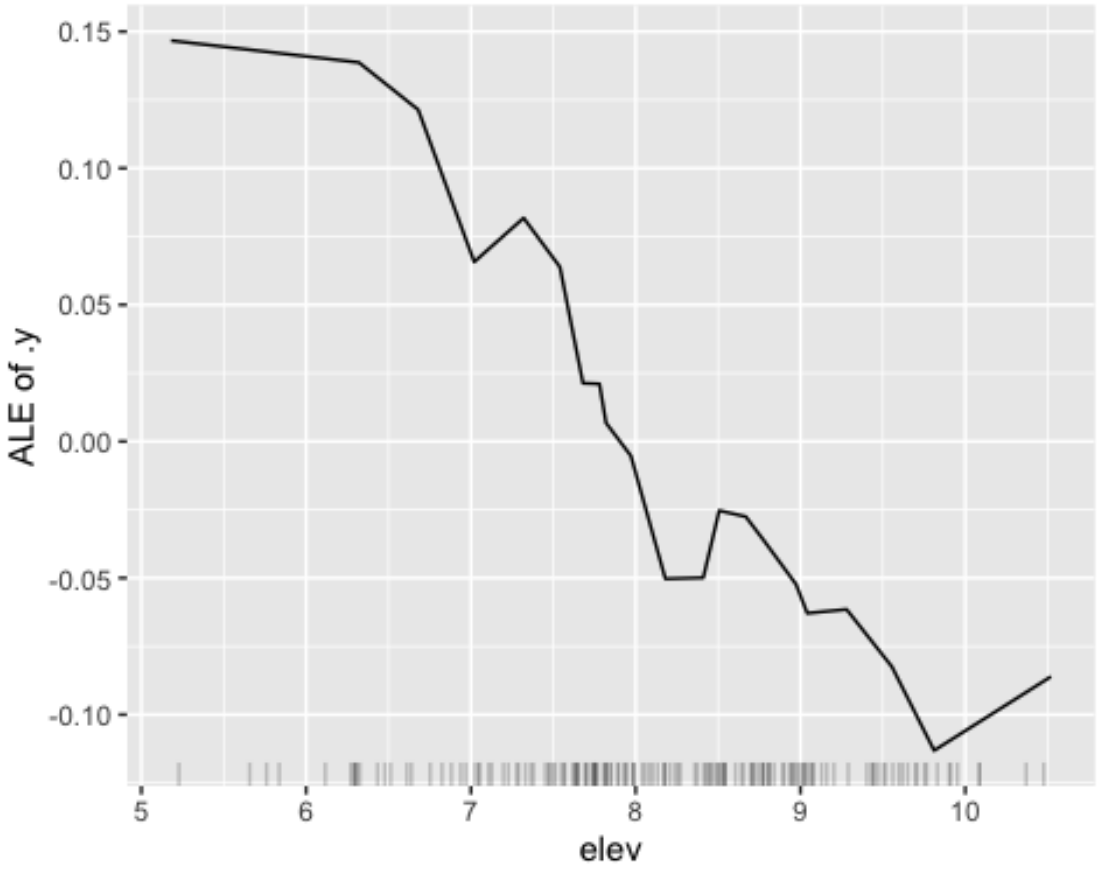
The `permutation.error` is (?) the s.d. of this distribution.

Feature effects:

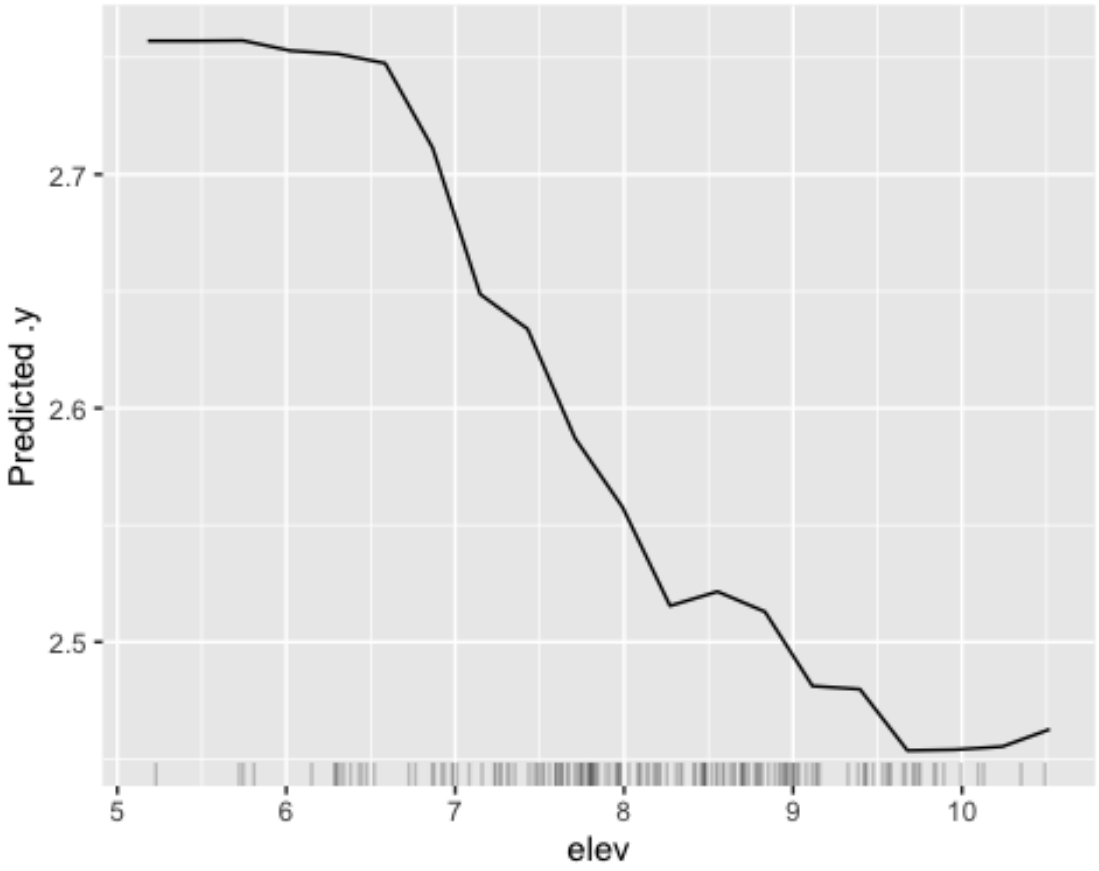
1. accumulated local effect (ALE); — see https://christophm.github.io/interpretable-ml-book/ale.html "ALE plots are a faster and unbiased alternative to partial dependence plots (PDPs)."
2. Partial Dependence Plots (PDP): this shows the prediction with other factors kept at their medians (?) — see https://christophm.github.io/interpretable-ml-book/pdp.html
3. Individual Conditional Expectation (ICE) curves – one per observation — see https://christophm.github.io/interpretable-ml-book/ice.html "one line per instance that shows how the instance's prediction changes when a feature changes" This is a PDP plot per observation.

See with `plot.FeatureEffect`. Create the object with `$new`, specifying a method, then plot.

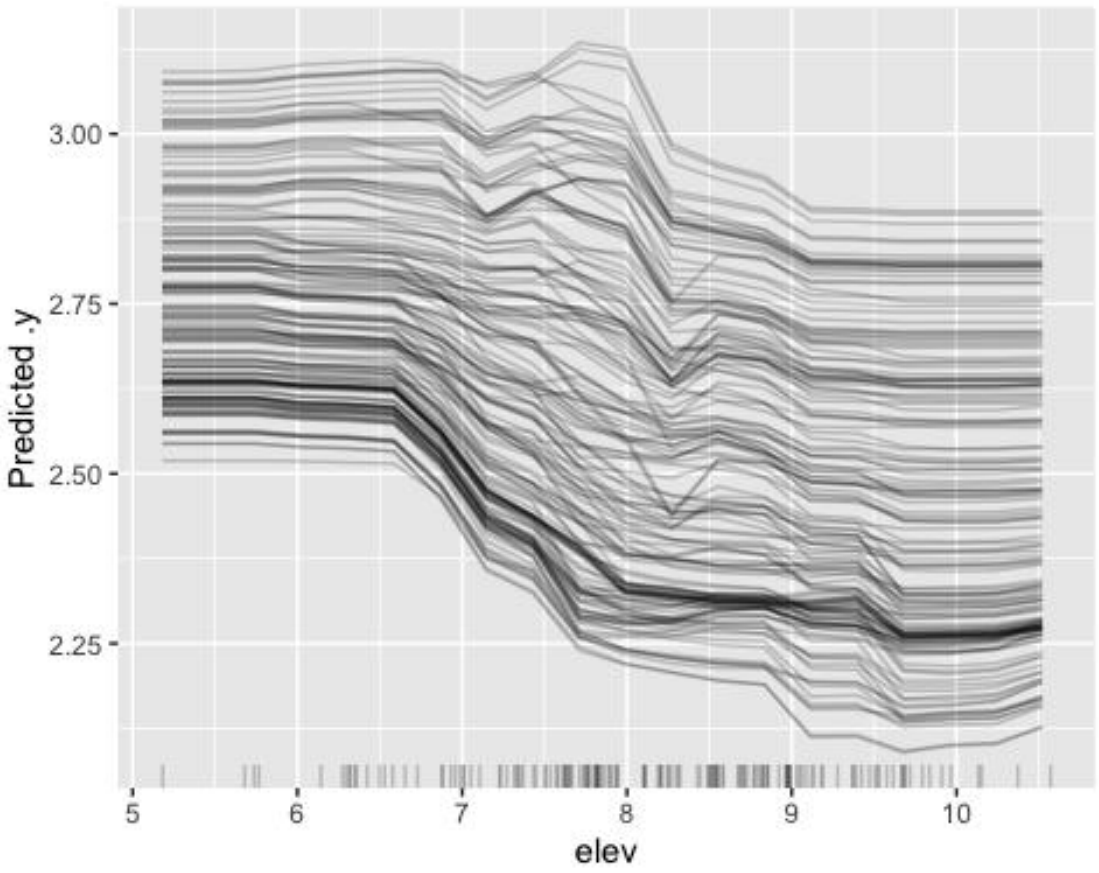
```
ale <- FeatureEffect$new(predictor, feature = "elev")
ale$plot()
```




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



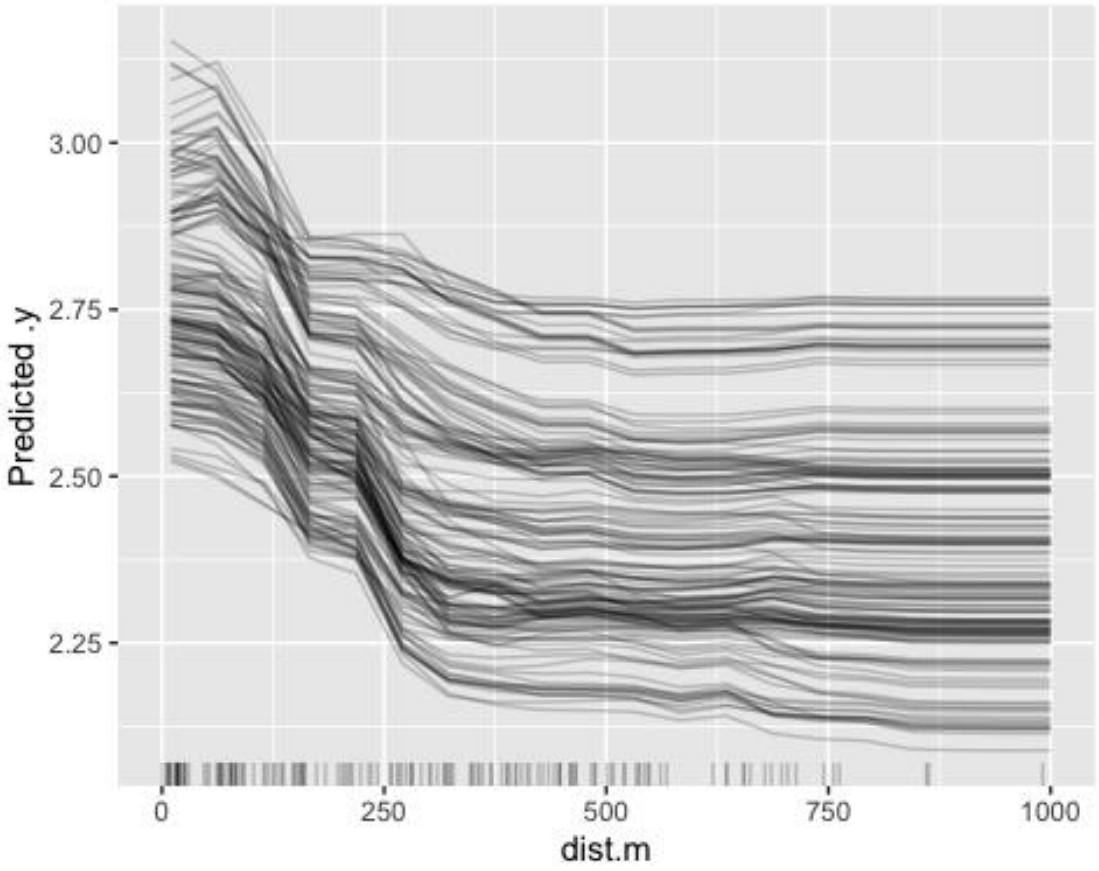

```
ice <- FeatureEffect$new(predictor, feature = "elev", method = "ice")
ice$plot()
```



In the ICE graph some interesting behaviour in the 7-8.5 m elevation range. But in general this follows the average PDP plot, we don't have any really unusual observations in terms of their dependence on elevation.

See this for distance:

```
ice.dist.m <- FeatureEffect$new(predictor, feature = "dist.m", method =
"ice")
ice.dist.m$plot()
```



Interesting different behaviour from 0 to 50 m – some with high values at 0 increase substantially at 50, while most decreases, as expected.

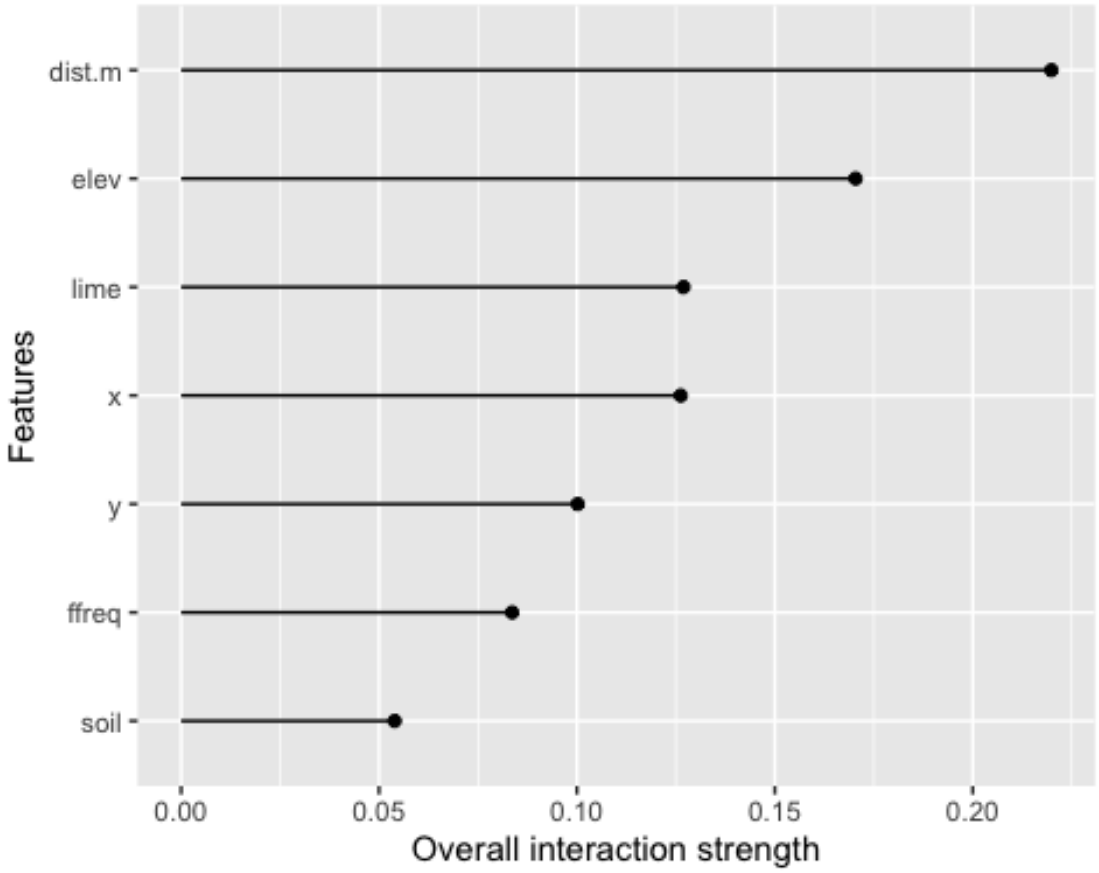
Interactions

"The interaction measure regards how much of the variance of f(x) is explained by the interaction. The measure is between 0 (no interaction) and 1 (= 100% of variance of f(x) due to interactions). For each feature, we measure how much they interact with any other feature."

See https://christophm.github.io/interpretable-ml-book/interaction.html

"When features interact with each other in a prediction model, the prediction cannot be expressed as the sum of the feature effects, because the effect of one feature depends on the value of the other feature."

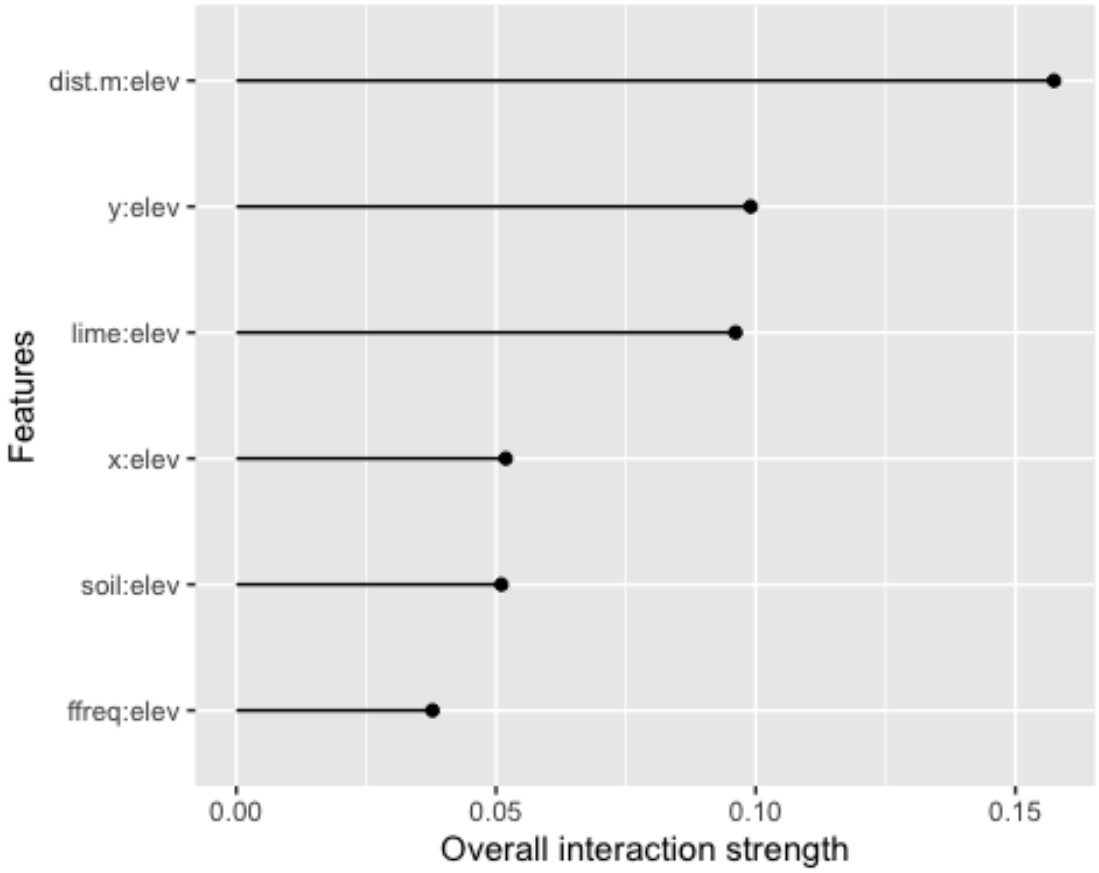
```
interact <- Interaction$new(predictor)
plot(interact)
```



This shows that the predictors interact a fair amount – none are independent.

Look at the 2-way interactions with a predictor:

```
interact.elev <- Interaction$new(predictor, feature = "elev")
plot(interact.elev)
```



This shows strong interactions between elevation and the others, especially `dist.m`, in prediction $\log_{10}$Zn.

Explain single predictions with a local model

This uses the `glmnet` "Lasso and Elastic-Net Regularized Generalized Linear Models" packagel, also the `gower` "Gower's Distance" package (Gower, John C. "A general coefficient of similarity and some of its properties." Biometrics (1971): 857-871).

"The … model fitted by `LocalModel` is a linear regression model and the data points are weighted by how close they are to the data point for w[h]ich we want to explain the prediction." Here 'close' means in multivariate feature space, I think (hence the use of Gower's distance).

Yes, from `?LocalModel`: "A weighted glm is fitted with the machine learning model prediction as target. Data points are weighted by their proximity to the instance to be explained, using the Gower proximity measure. L1-regularization is used to make the results sparse." (hence the use of `glmnet`).

So this is a linear regression model (hence we can interpet the coefficients) but only using close-by points in feature space.

Look at this for the first point, which happens to be close to the river and with a high level of Zn.

```
meuse[1,]
```

```
##        x      y cadmium copper lead zinc  elev       dist   om ffreq soil
lime
## 1 181072 333611    11.7     85  299 1022 7.909 0.00135803 13.6     1    1
1
##   landuse dist.m    logZn
## 1      Ah     50 3.009451
```

```
lime.explain <- iml::LocalModel$new(predictor, x.interest = X[1, ])
```

```
## Loading required package: glmnet
```

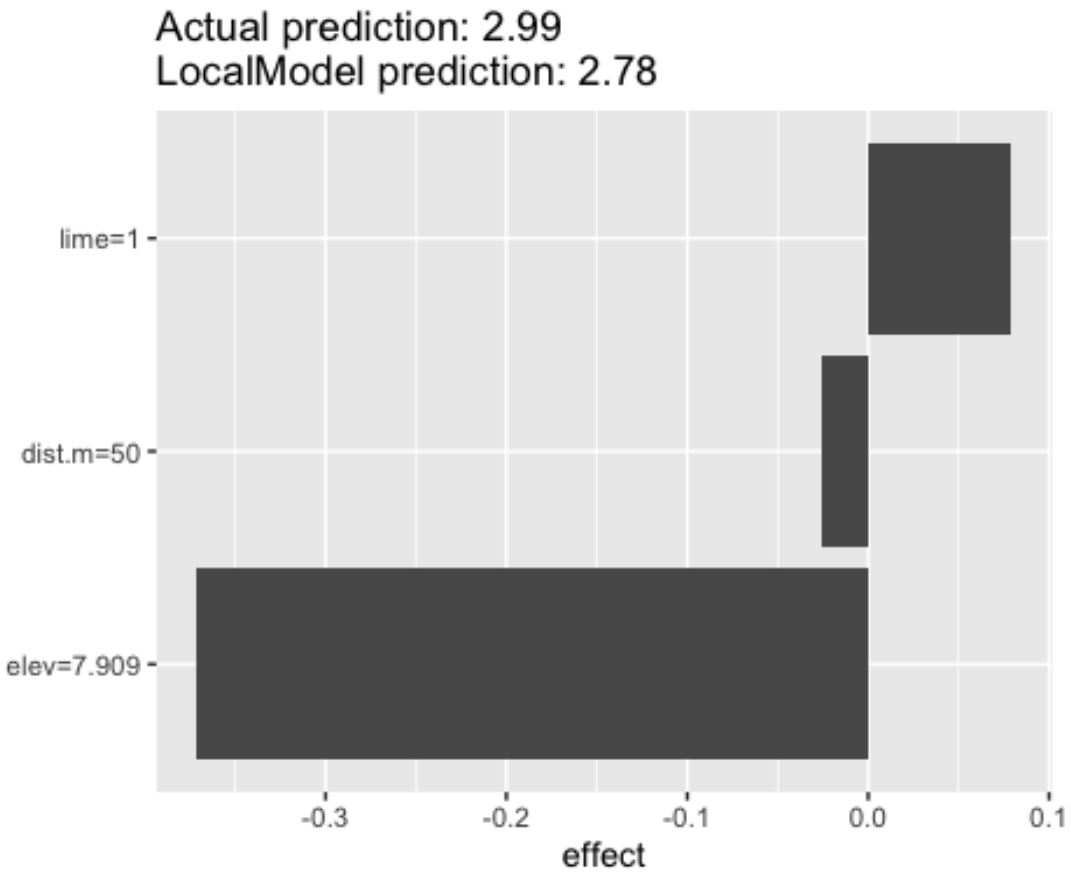
```
## Loaded glmnet 4.1-7
```

```
## Loading required package: gower
```

```
lime.explain$results
```

```
##                 beta x.recoded      effect x.original feature
feature.value
## dist.m -0.0005252513    50.000 -0.02626257         50  dist.m
dist.m=50
## elev   -0.0469173108     7.909 -0.37106901      7.909    elev
elev=7.909
## lime=1  0.0790714563     1.000  0.07907146          1  lime=1
lime=1
```

```
plot(lime.explain)
```

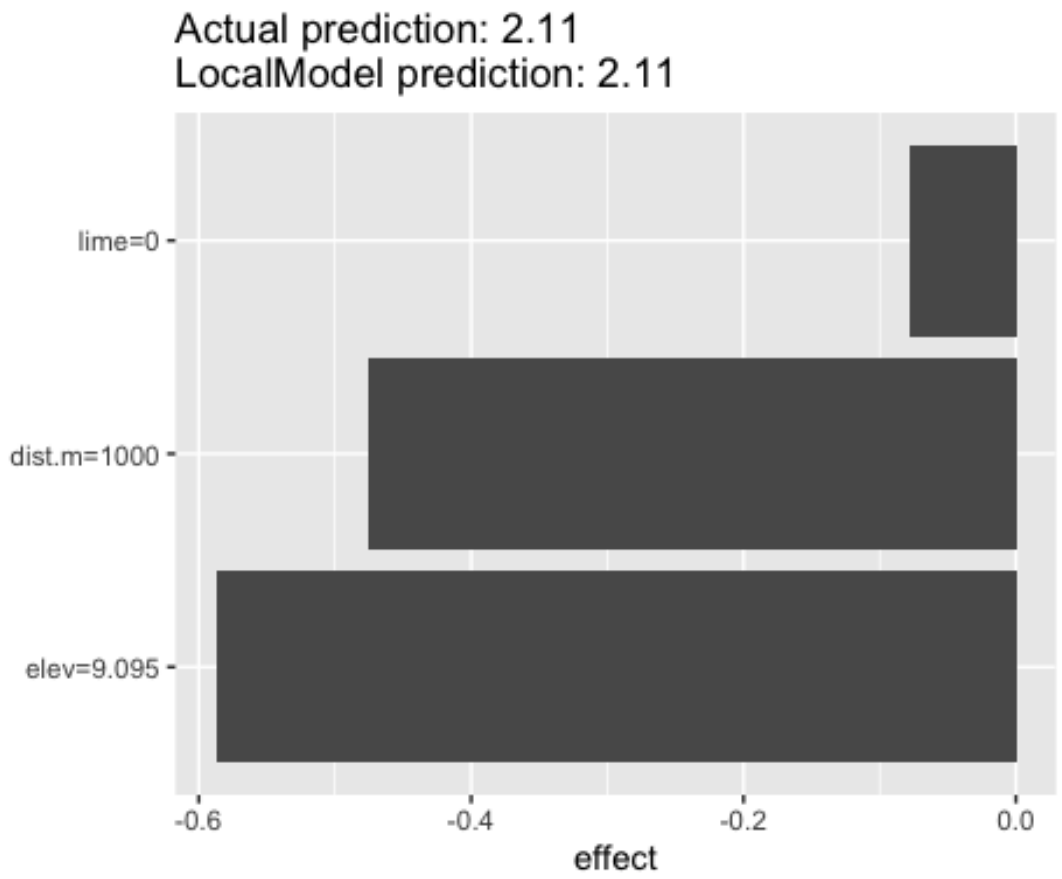


So here the local model predicts a lower value than actual, and the main reason for that is the elevation of the surrounding points: these differ from the target point but affect the prediction.

Look at this for one of the furthest points from the river:

```
ix.maxdist <- which.max(meuse$dist.m)
meuse[ix.maxdist, ]

##          x      y cadmium copper lead zinc  elev     dist  om ffreq soil
lime
## 111 180451 331473     0.2     18   50  117 9.095 0.809742 5.3     2    3
0
##     landuse dist.m    logZn
## 111      Fw   1000 2.068186

lime.explain <- LocalModel$new(predictor, x.interest = X[ix.maxdist, ])
lime.explain$results

##                 beta x.recoded      effect x.original feature
feature.value
## dist.m -0.0004749539  1000.000 -0.47495392       1000  dist.m
dist.m=1000
## elev   -0.0646175574     9.095 -0.58769668      9.095    elev
```

```
elev=9.095
## lime=0 -0.0772825118     1.000 -0.07728251          0  lime=0
lime=0

plot(lime.explain)
```



Here both distance and elevation have a strong effect but the predictions are the same.

Shapley values

The local importance metrics explored in the previous sections do not account for the full set of interactions at a data point. The Shapley method corrects this. It is available in several packages, including `iml`.

"[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 "game" is the prediction task for a single instance of the dataset. The "gain" is the actual prediction for this instance minus the average prediction for all instances. The "players" are the feature values of the instance that collaborate to receive the gain (= predict a certain value).

"The Shapley value is the average marginal contribution of a feature value across all possible coalitions.

"The Shapley value is the only explanation method with a solid theory. The axioms – efficiency, symmetry, dummy, additivity – give the explanation a reasonable foundation."[1]

See https://christophm.github.io/interpretable-ml-book/shapley.html for more details

The Shapley value is a solution for computing feature contributions for single predictions for any machine learning model.

9.5.3.1 The Shapley Value

The Shapley value is defined via a value function val of players in S.

The Shapley value of a feature value is its contribution to the payout, weighted and summed over all possible feature value combinations:

$$\phi_j(val) = \sum_{S \subseteq \{1,\dots,p\} \setminus \{j\}} \frac{|S|!\,(p-|S|-1)!}{p!}\big(val(S \cup \{j\}) - val(S)\big)$$

where S is a subset of the features used in the model, x is the vector of feature values of the instance to be explained and p the number of features. $val_x(S)$ is the prediction for feature values in set S that are marginalized over features that are not included in set S:

$$val_x(S) = \int \hat{f}(x_1, \dots, x_p) d\mathbb{P}_{x \notin S} - E_X\left(\hat{f}(X)\right)$$

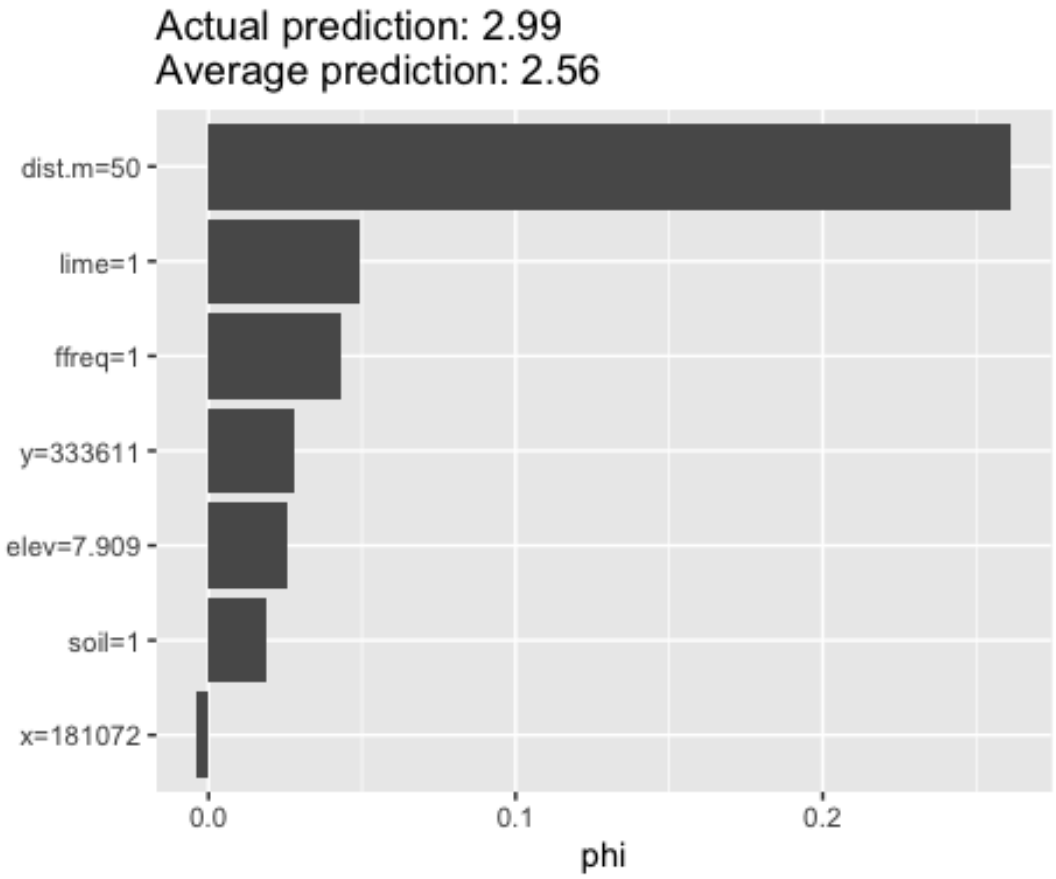
Compute and display the Shapley values for the predictive model in `predictor`, i.e. the fitted `ranger` random forest model, for the first-listed observation, and for the maximum-distance observation.

```
str(predictor)
```

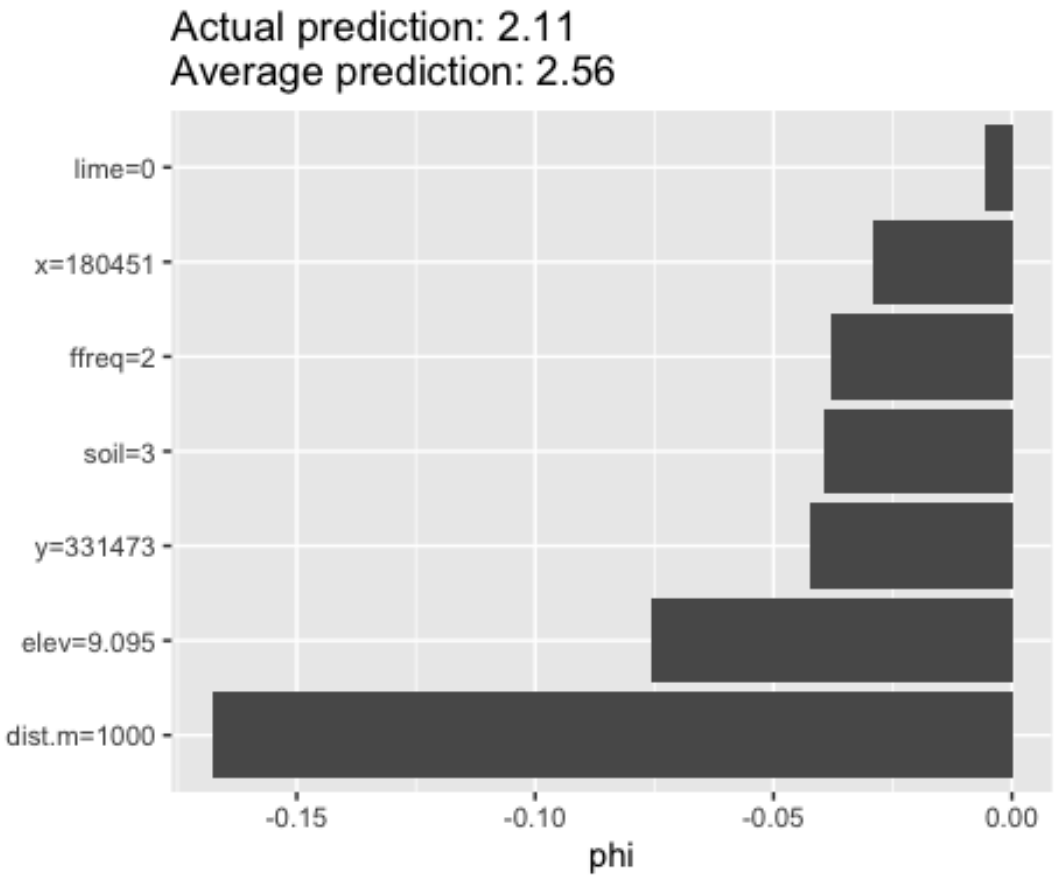
```
## Classes 'Predictor', 'R6' <Predictor>
##   Public:
##     batch.size: 1000
##     class: NULL
##     clone: function (deep = FALSE)
##     data: Data, R6
##     initialize: function (model = NULL, data = NULL, predict.function =
NULL,
##     model: ranger
##     predict: function (newdata)
##     prediction.colnames: NULL
##     prediction.function: function (newdata)
##     print: function ()
```

[1] https://christophm.github.io/iml/articles/intro.html

```
##     task: unknown
##   Private:
##     predictionChecked: TRUE

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




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



Notice that the figure shows the actual values of each predictor at the observation point, and the ϕ value, i.e., numerical contribution to the difference between actual and average prediction. These sum to the difference.

See the results as a table:

```
(results <- shapley$results)
```

```
##   feature          phi      phi.var feature.value
## 1   ffreq  0.043455359 0.0029537994       ffreq=1
## 2       x -0.003733284 0.0023425201      x=181072
## 3       y  0.027773645 0.0013254031      y=333611
## 4  dist.m  0.261370623 0.0356227811     dist.m=50
## 5    elev  0.025451407 0.0072659732    elev=7.909
## 6    soil  0.018860935 0.0007555224        soil=1
## 7    lime  0.049295269 0.0039463700        lime=1
```

```
sum(shapley$results$phi)
```

```
## [1] 0.422474
```

```
(results <- shapley.maxdist$results)
```

```
##   feature          phi      phi.var feature.value
## 1   ffreq -0.037829891 0.0021321660       ffreq=2
## 2       x -0.029083529 0.0029515665      x=180451
## 3       y -0.042511788 0.0009694630      y=331473
## 4  dist.m -0.167879673 0.0230261387   dist.m=1000
## 5    elev -0.075549723 0.0081161122    elev=9.095
## 6    soil -0.039593450 0.0008929302        soil=3
## 7    lime -0.005728082 0.0006475838        lime=0
```

```
sum(shapley.maxdist$results$phi)
```

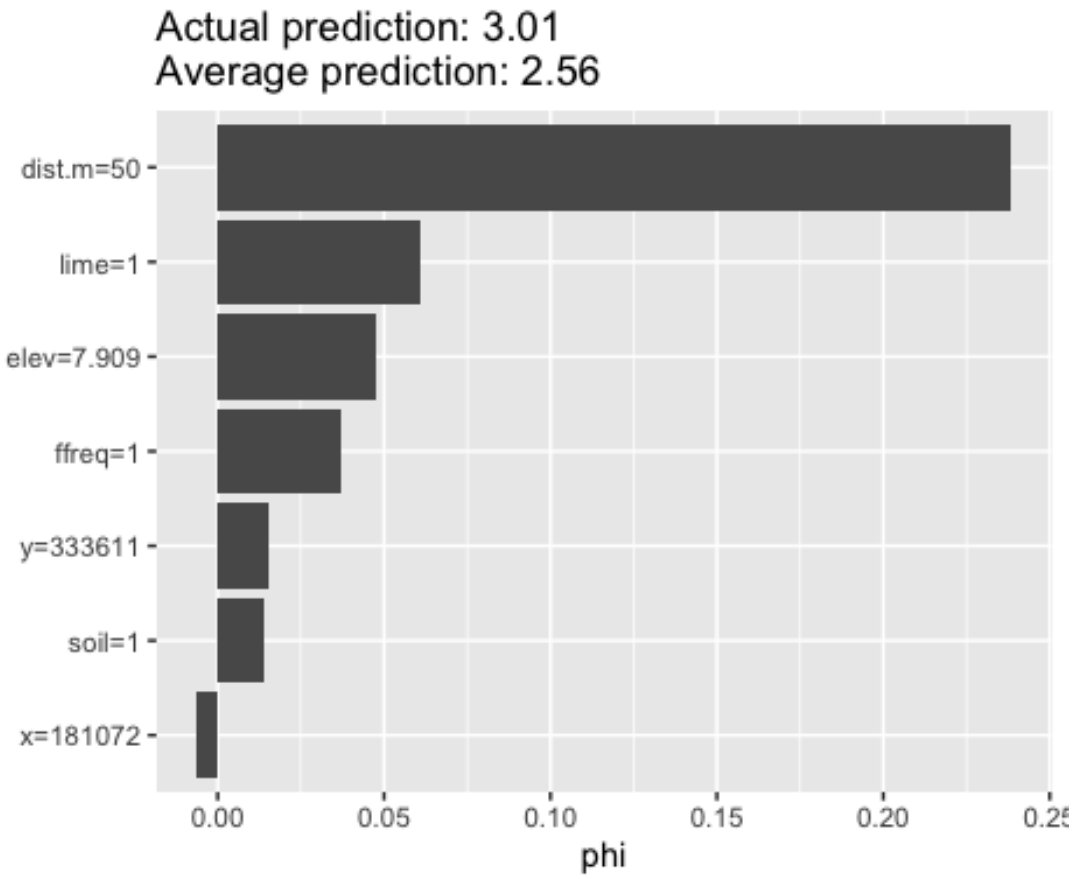
```
## [1] -0.3981761
```

For the point near the river the actual is higher than expected; this is the sum of the individual ϕ values. The main positive contribution is from distance: it is closer than most points. For the distant point the actual is lower than the prediction; this is mainly because of distance: the point is further, and also lower and these have the most influence on the lower actual value.

"Be careful to interpret the Shapley value correctly: The Shapley value is the *average contribution of a feature value to the prediction in different coalitions*. The Shapley value is NOT the difference in prediction when we would remove the feature from the model."

For QRF, using `predictor.qrf` for the fitted model:

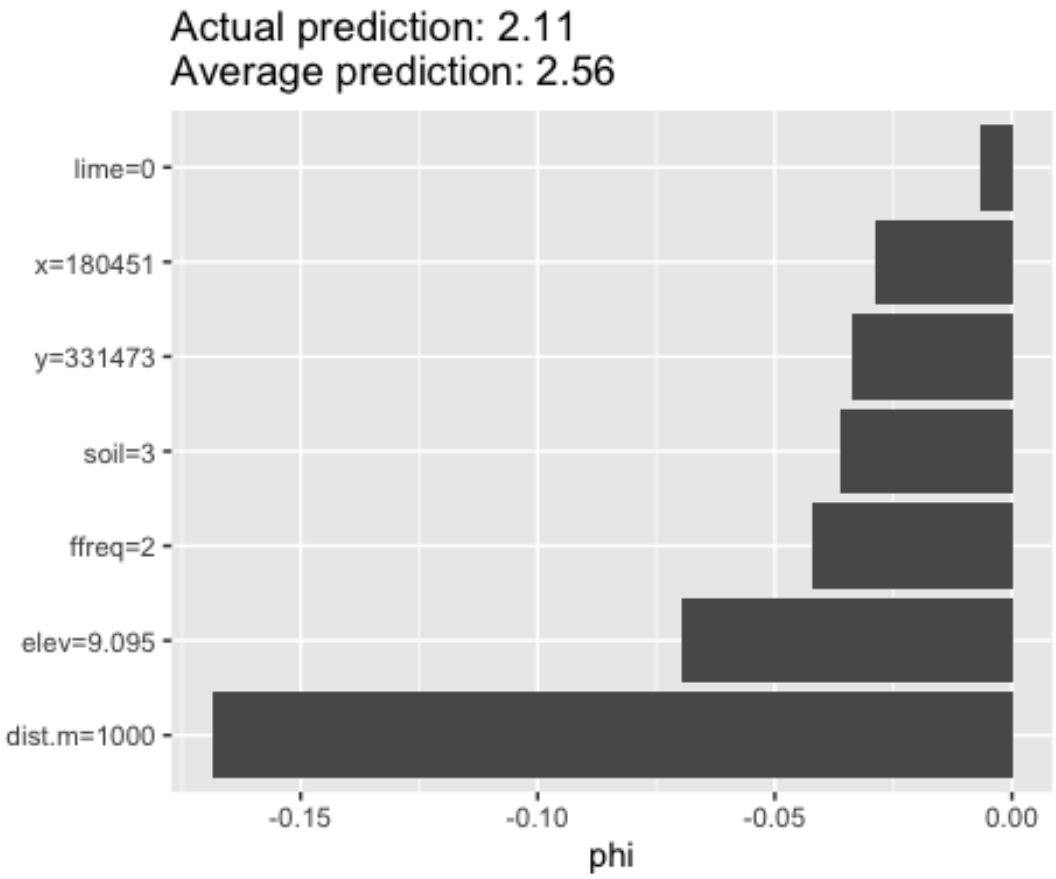
```
shapley.qrf <- iml::Shapley$new(predictor.qrf, x.interest = X[1, ])
shapley.qrf$plot()
```




```
(results <- shapley.qrf$results)
```

```
##   feature          phi      phi.var feature.value
## 1   ffreq  0.037061606 0.0028923989       ffreq=1
## 2       x -0.006194138 0.0020300314      x=181072
## 3       y  0.015083155 0.0006810239      y=333611
## 4  dist.m  0.238427604 0.0355260869     dist.m=50
## 5    elev  0.047515789 0.0051210461    elev=7.909
## 6    soil  0.014018198 0.0005725186        soil=1
## 7    lime  0.060824871 0.0044906113        lime=1
```

```
shapley.maxdist.qrf <- Shapley$new(predictor.qrf, x.interest = X[ix.maxdist,
])
shapley.maxdist.qrf$plot()
```




```
(results <- shapley.maxdist.qrf$results)
```

```
##   feature          phi      phi.var feature.value
## 1   ffreq -0.042017308 0.0027422539       ffreq=2
## 2       x -0.029000858 0.0037320164      x=180451
## 3       y -0.033532601 0.0007193414      y=331473
## 4  dist.m -0.168784922 0.0212001094   dist.m=1000
## 5    elev -0.069793813 0.0088289128    elev=9.095
## 6    soil -0.036159301 0.0007557849        soil=3
## 7    lime -0.006769676 0.0004852137        lime=0
```

I think this uses only the overall prediction. Just the model form is used. Any differences with the first model are because of randomness in computing the coalitions.

SHAP – (SHapley Additive exPlanations)

"SHAP (SHapley Additive exPlanations) by Lundberg and Lee (2017)[2] is a method to explain individual predictions. SHAP is based on the game theoretically optimal Shapley values"[3].

So this method uses Shapley values but from a different perspective.

"[T]he Shapley value explanation is represented as an additive feature attribution method, a linear model."

"The big difference to `LIME` is the **weighting** of the instances in the regression model. LIME weights the instances according to how close they are to the original instance. The more 0's in the coalition vector, the smaller the weight in `LIME`. SHAP weights the sampled instances according to the weight the coalition would get in the Shapley value estimation. Small coalitions (few 1's) and large coalitions (i.e. many 1's) get the largest weights."

The original method is `KernelSHAP`, but "Lundberg et al.[4] proposed `TreeSHAP`, a variant of SHAP for tree-based machine learning models such as decision trees, random forests and gradient boosted trees. `TreeSHAP` was introduced as a fast, model-specific alternative to `KernelSHAP`".

How to compute these? One options is `shapper` which uses `reticulate` to access Python library `SHAP`. See http://smarterpoland.pl/index.php/2019/03/shapper-is-on-cran-its-an-r-wrapper-over-shap-explainer-for-black-box-models/.

Another is the native R `fastshap`, see https://github.com/bgreenwell/fastshap. This is on CRAN.

The `fastshap::explain` function requires a prediction function, which in this case is just `ranger::predict.ranger`, which is automatically called from the generic `predict` when the object is a `ranger` model.

Note the use of `nsim`: "To obtain the most accurate results, nsim should be set as large as feasibly possible.".

```
require(fastshap)

## Loading required package: fastshap
```

---

[2] Lundberg, Scott M., and Su-In Lee. "A unified approach to interpreting model predictions." Advances in Neural Information Processing Systems (2017)

[3] https://christophm.github.io/interpretable-ml-book/shap.html

[4] Lundberg, Scott M., Gabriel G. Erion, and Su-In Lee. "Consistent individualized feature attribution for tree ensembles." arXiv preprint arXiv:1802.03888 (2018)

```
##
## Attaching package: 'fastshap'

## The following object is masked from 'package:dplyr':
##
##     explain

## The following object is masked from 'package:generics':
##
##     explain

## The following object is masked from 'package:vip':
##
##     gen_friedman
```

```
pfun <- function(object, newdata) {
  predict(object, data = newdata)$predictions
}
fshap <- fastshap::explain(object = m.lzn.ra,
                  X = X, pred_wrapper = pfun,
                  nsim = 24)
head(fshap)
```

```
## # A tibble: 6 × 7
##    ffreq        x      y  dist.m    elev    soil    lime
##    <dbl>    <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 0.0448 -0.0105  0.0175  0.208   0.0233  0.0138  0.0376
## 2 0.0327 -0.00315 0.0147  0.233   0.106   0.0349  0.0632
## 3 0.0355 -0.00392 0.0390  0.0446  0.0559  0.0125  0.0390
## 4 0.0301 -0.0261  0.0190 -0.113   0.0177 -0.0218 -0.0418
## 5 0.0251 -0.0105  0.0213 -0.106   0.0331 -0.0265 -0.0260
## 6 0.0211 -0.0222  0.0209 -0.108  -0.0124 -0.0186 -0.0181
```

Each observation has a set of Shapley values. These are the contribution to the difference between the observed value and the average value of all observations. For the first (closest) point, there is a large positive contribution to the difference from `dist.m` and `elev`.

How much do these differ from those computed by `iml`? Examine for the first observation

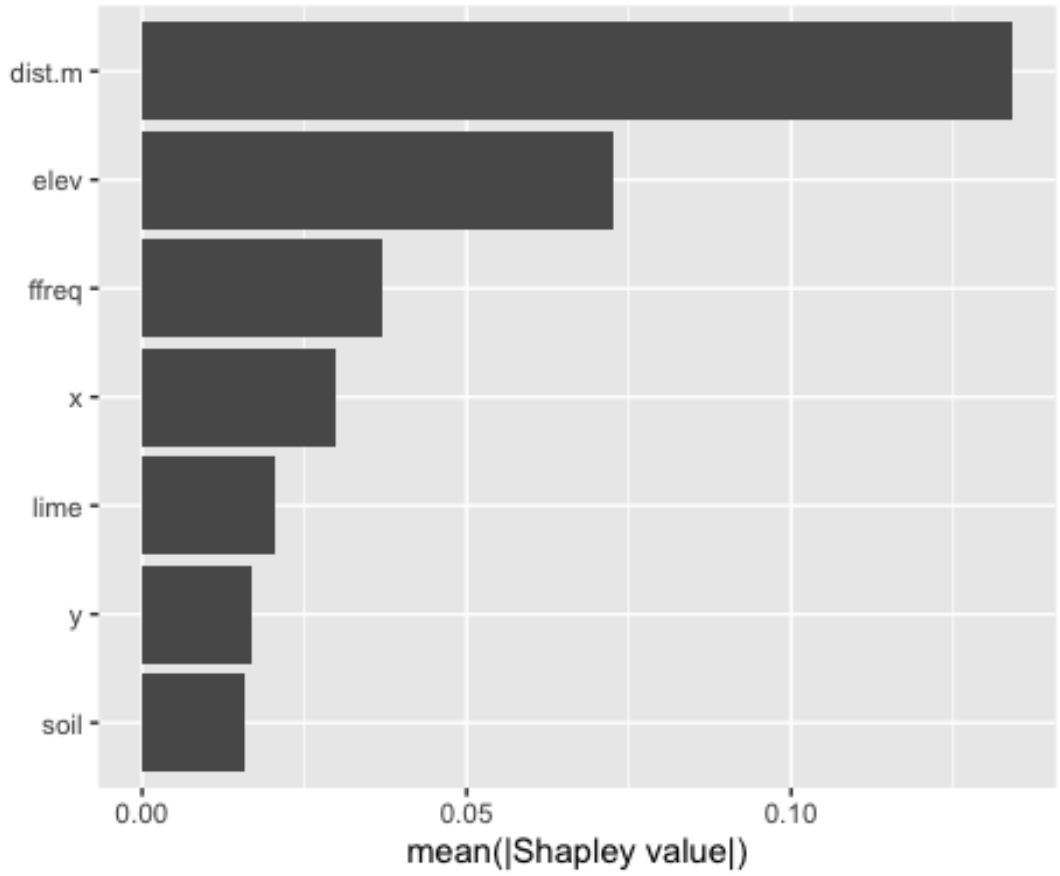
```
shapley <- iml::Shapley$new(predictor, x.interest = X[1, ])
rbind(fshap[1,], shapley$results$phi)
```

```
## # A tibble: 2 × 7
##    ffreq        x      y dist.m   elev   soil   lime
##    <dbl>    <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
## 1 0.0448 -0.0105  0.0175  0.208 0.0233 0.0138 0.0376
## 2 0.0266 -0.00382 0.0157  0.236 0.0387 0.0156 0.0647
```

Similar but not identical. The SHAP method finds more importance for `y` and `dist.m`.

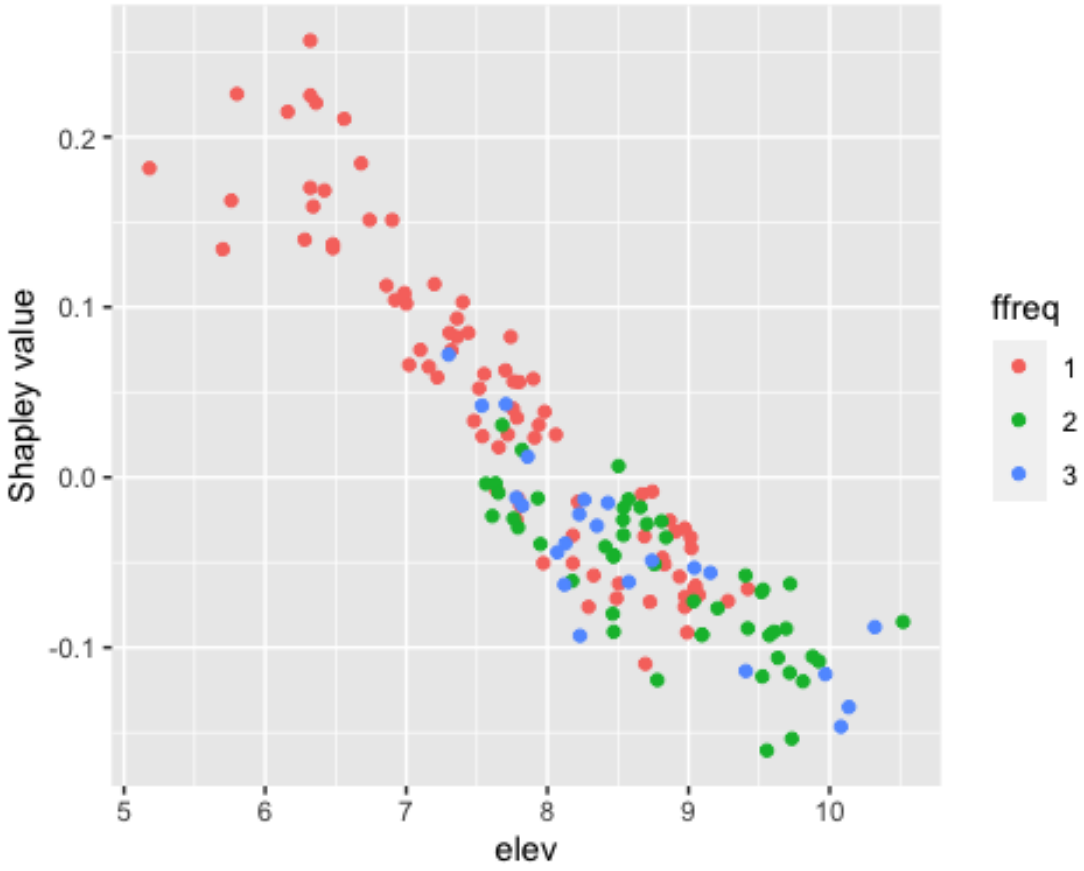
Plot the mean values over all points, i.e., global variable importance but calculated from all individuals:

```
autoplot(fshap)
```



Plot the dependence on elevation for all observations. The `ggplot2::autoplot` method can be used by packages (here, `fshap`) to customize `ggplot` plots.

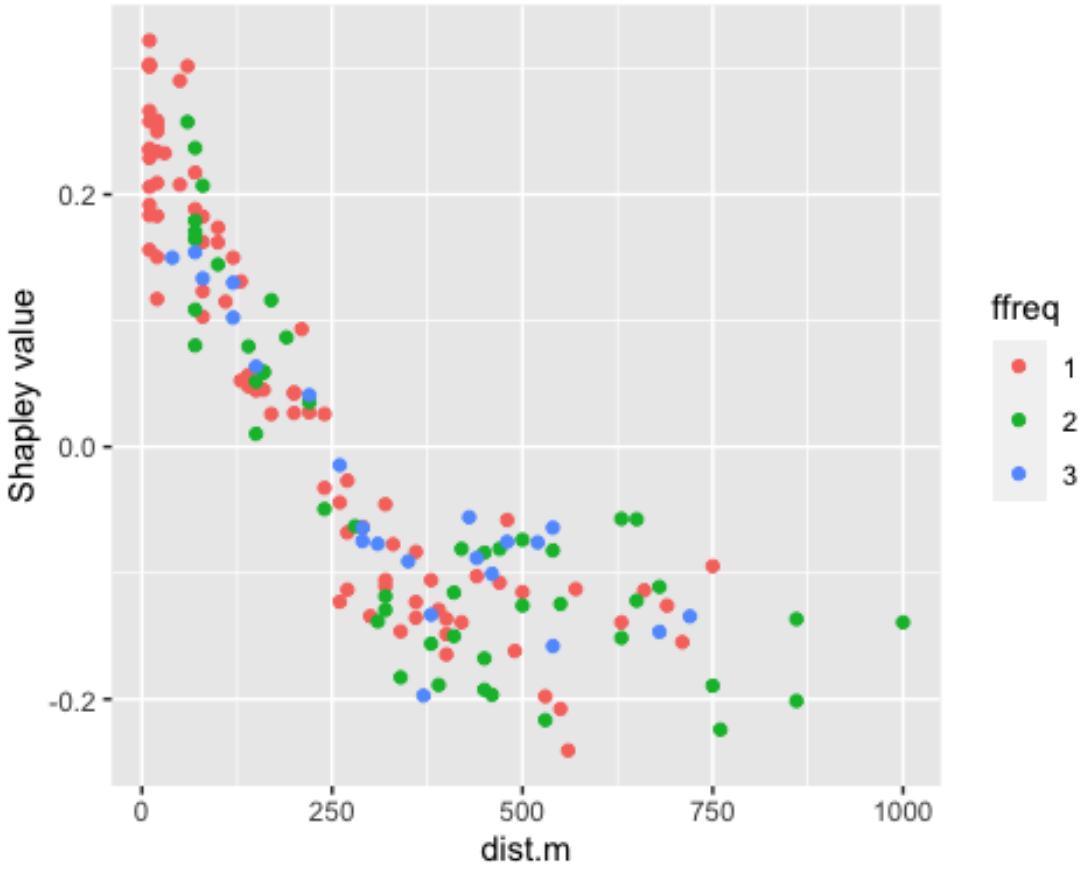
```
autoplot(fshap,
         type = "dependence",
         feature = "elev",
         X = X,
         color_by = "ffreq")
```



A consistent story: influence of elevation on the prediction is significant for both low and high points and the relation is almost linear. The value is positive for low elevations; the reverse is true for high elevations.

And on distance to river for all observations:

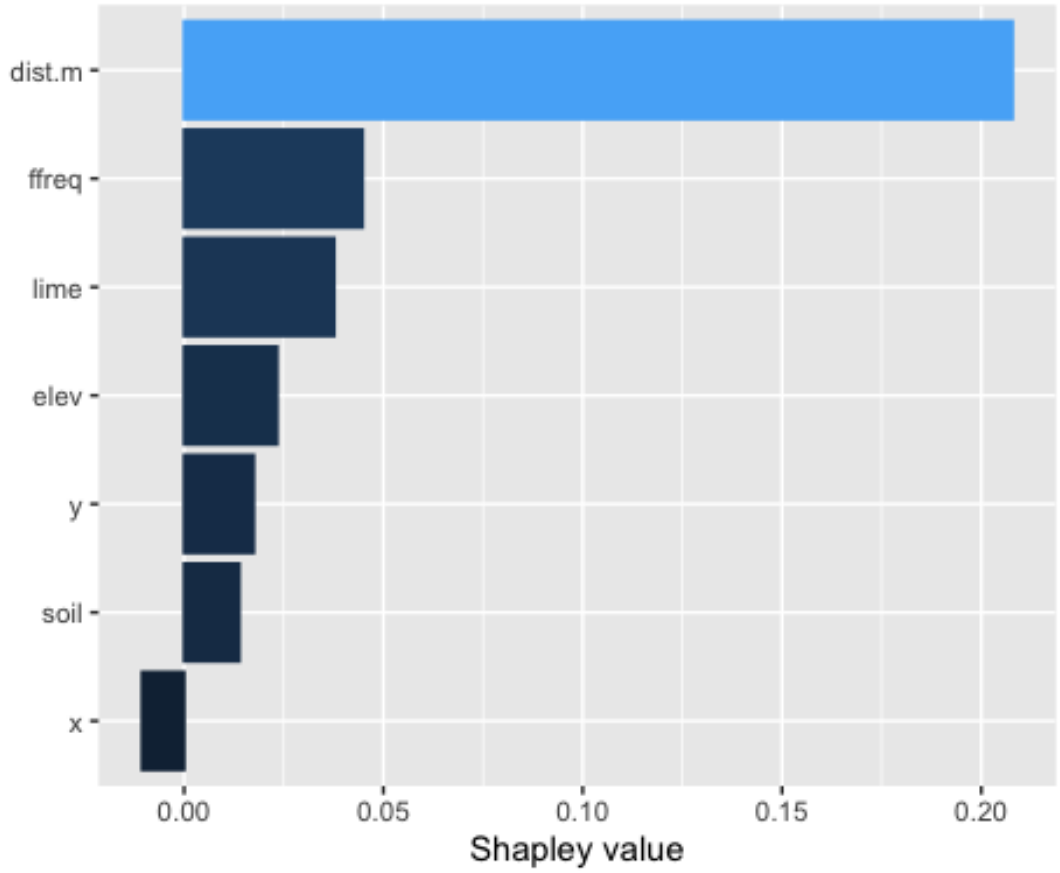
```
autoplot(fshap,
         type = "dependence",
         feature = "dist.m",
         X = X,
         color_by = "ffreq")
```



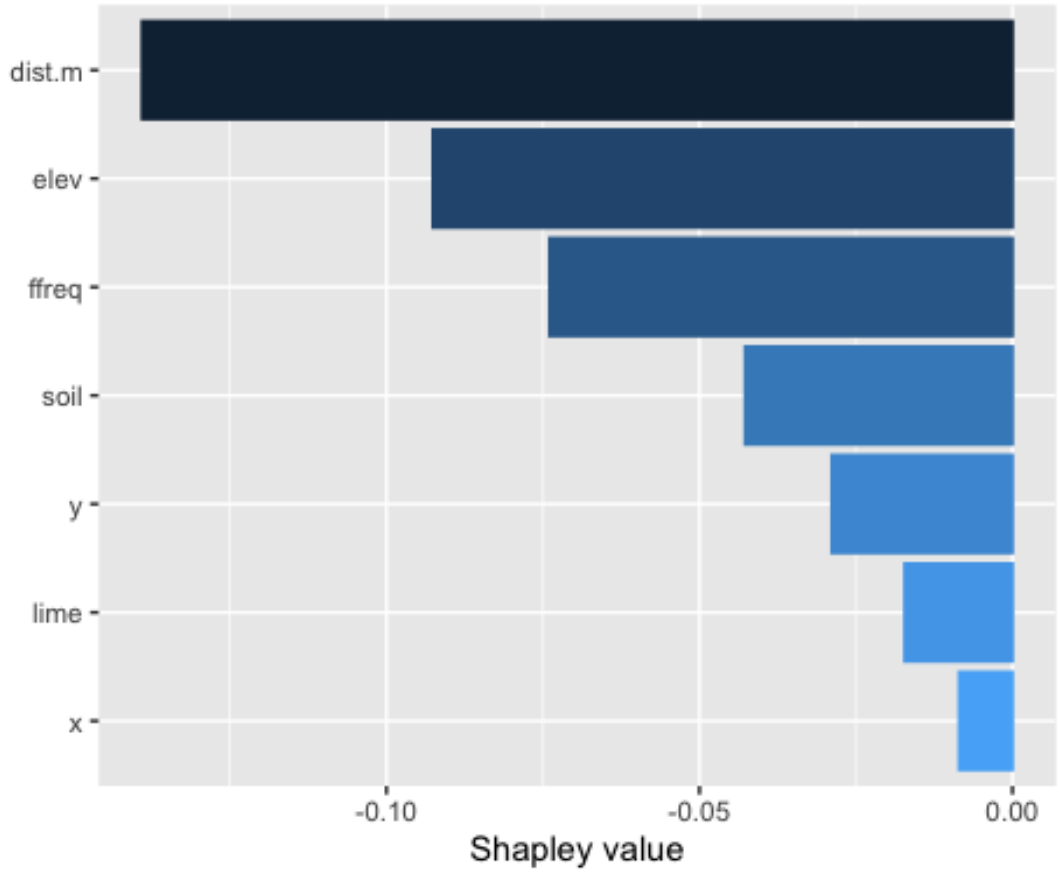
We see that for nearby point there is a strong positive influence of distance, i.e., the distance is very important to that prediction. For points > 250 m there is also an influence but negative.

The contribution for the closest and furthest point:

```
autoplot(fshap,
         type = "contribution",
         row_num = 1)
```




```
autoplot(fshap,
         type = "contribution",
         row_num = ix.maxdist)
```



For this simple model these do not differ much from the Shapley values computed by `iml`.

Visualization of all individual Shapley values

This requires the `shapviz` library, which can accept many forms of computed Shapley values, and then make some interesting and colourful visualizations.

The `shapviz::shapviz` function requires (1) a matrix of computed Shapley values, (2) a set of features for which to display the values, (2) a list of observations.

First, we set up the matrix from the Shapley values computed by `fastshap`, above.

```
str(fshap)
```

```
## tibble[,7] (S3: tbl_df/tbl/data.frame/explain)
##  $ ffreq : num [1:155] 0.0448 0.0327 0.0355 0.0301 0.0251 ...
##  $ x     : num [1:155] -0.01053 -0.00315 -0.00392 -0.0261 -0.01049 ...
##  $ y     : num [1:155] 0.0175 0.0147 0.039 0.019 0.0213 ...
##  $ dist.m: num [1:155] 0.2078 0.2328 0.0446 -0.1132 -0.1056 ...
##  $ elev  : num [1:155] 0.0233 0.1057 0.0559 0.0177 0.0331 ...
##  $ soil  : num [1:155] 0.0138 0.0349 0.0125 -0.0218 -0.0265 ...
##  $ lime  : num [1:155] 0.0376 0.0632 0.039 -0.0418 -0.026 ...
```

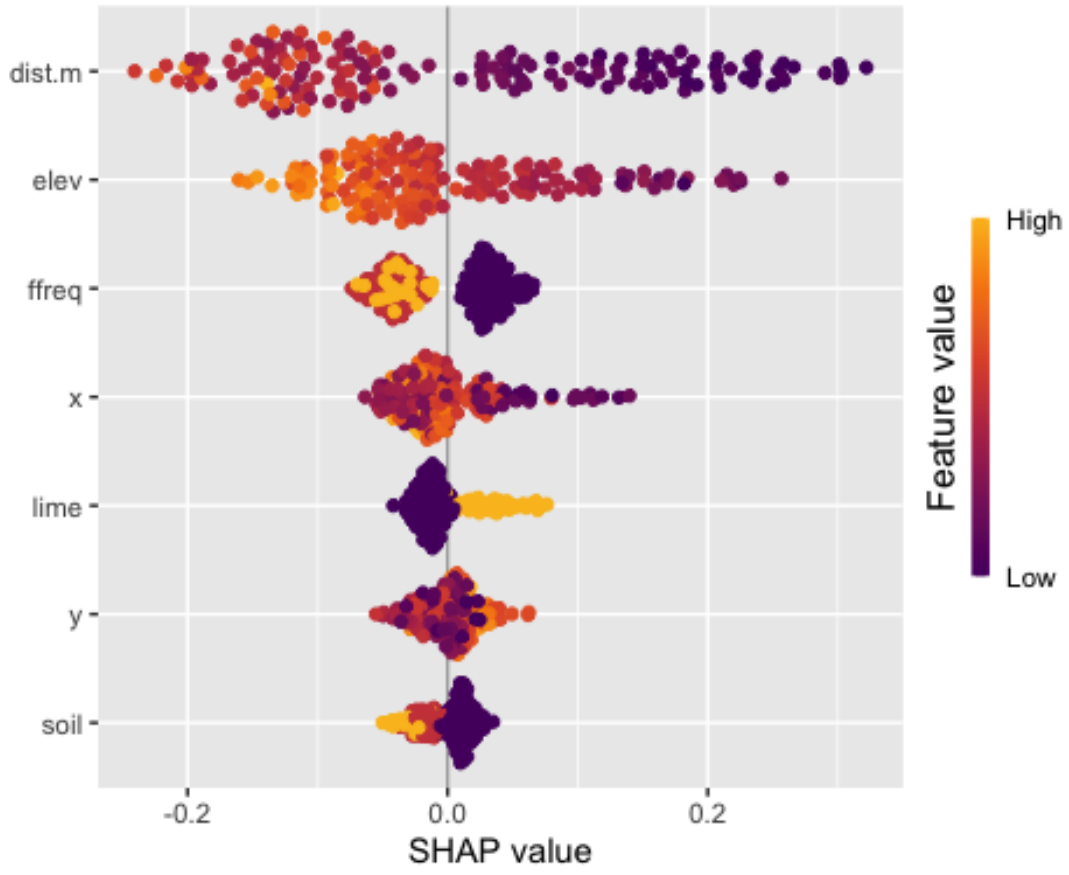
```
fshap.m <- as.matrix(fshap)
dim(fshap.m)

## [1] 155   7
```

Now set up the visualizaton object:

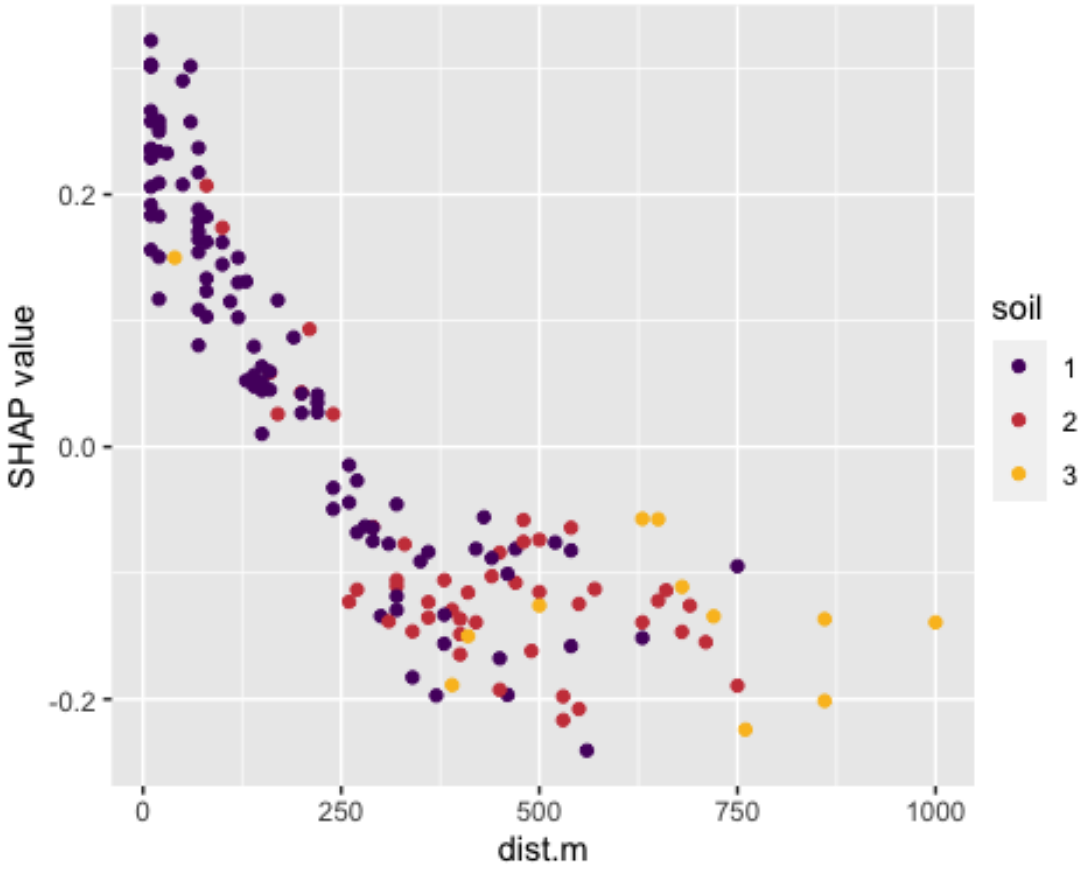
```
library(shapviz)
sv.fshap <- shapviz(fshap.m,
  X = meuse[ , c("ffreq", "x", "y", "dist.m", "elev", "soil",  "lime")],
  bg_X = meuse) # small dataset, can see all of them
```

And here are some interesting visualizations:

```
sv_importance(sv.fshap, kind = "bee")
```




```
# auto-select most important interacting feature
sv_dependence(sv.fshap, v = "dist.m", color_var = "auto")
```



The first plot shows the feature value of each observation in colour (scale bar on right) and the SHAP value for each observation on the x-axis. In this example the lowest feature values for `dist.m` (i.e., closest to the river) get high SHAP values, i.e., distance is quite important to their value.

The second plot shows this for just one feature.

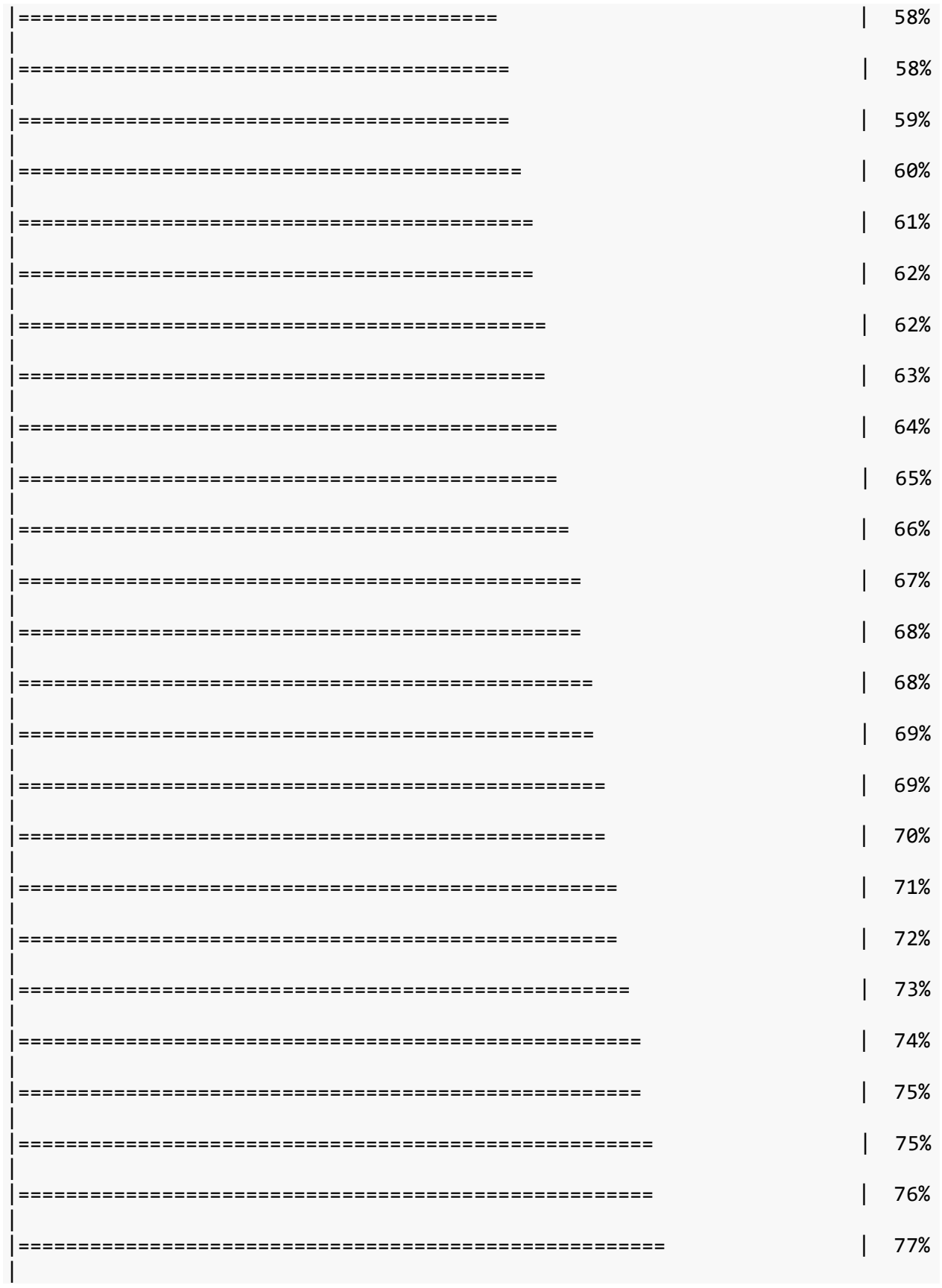
Another approach is using the `kernelshap` package, which also computes according to the Lundberg and Lee algorithm. This requires (1) an existing fitted model, (2) a set of features for which to display the values, (2) a list of observations (the latter two as required by `shapviz`).

```
library(kernelshap)
# use existing ranger fits
s <- kernelshap(m.lzn.ra.l,
                X = meuse[ , c("ffreq", "x", "y", "dist.m", "elev", "soil",
"lime")],
                bg_X = meuse) # small dataset, can see all of them

## Exact Kernel SHAP values

##   |
|                                                                      |   0%
|
```

```
|                                                                      |   1%
|
|=                                                                     |   1%
|
|=                                                                     |   2%
|
|==                                                                    |   3%
|
|===                                                                   |   4%
|
|===                                                                   |   5%
|
|====                                                                  |   5%
|
|====                                                                  |   6%
|
|=====                                                                 |   6%
|
|=====                                                                 |   7%
|
|=====                                                                 |   8%
|
|======                                                                |   8%
|
|======                                                                |   9%
|
|=======                                                               |  10%
|
|========                                                              |  11%
|
|========                                                              |  12%
|
|=========                                                             |  12%
|
|=========                                                             |  13%
|
|==========                                                            |  14%
|
|==========                                                            |  15%
|
|===========                                                           |  16%
|
|============                                                          |  17%
|
|============                                                          |  18%
|
|=============                                                         |  18%
|
|=============                                                         |  19%
|
```

```
|==============                                                  |  19%
|
|==============                                                  |  20%
|
|===============                                                 |  21%
|
|===============                                                 |  22%
|
|================                                                |  23%
|
|=================                                               |  24%
|
|=================                                               |  25%
|
|==================                                              |  25%
|
|==================                                              |  26%
|
|===================                                             |  27%
|
|====================                                            |  28%
|
|====================                                            |  29%
|
|=====================                                           |  30%
|
|=====================                                           |  31%
|
|======================                                          |  31%
|
|======================                                          |  32%
|
|=======================                                         |  32%
|
|=======================                                         |  33%
|
|========================                                        |  34%
|
|========================                                        |  35%
|
|=========================                                       |  36%
|
|=========================                                       |  37%
|
|==========================                                      |  38%
|
|==========================                                      |  38%
|
|==========================                                      |  39%
|
```

```
|============================                                          |  40%
|
|=============================                                         |  41%
|
|=============================                                         |  42%
|
|==============================                                        |  42%
|
|==============================                                        |  43%
|
|==============================                                        |  44%
|
|===============================                                       |  44%
|
|===============================                                       |  45%
|
|================================                                      |  45%
|
|================================                                      |  46%
|
|================================                                      |  47%
|
|=================================                                     |  48%
|
|=================================                                     |  49%
|
|==================================                                    |  49%
|
|==================================                                    |  50%
|
|==================================                                    |  51%
|
|===================================                                   |  51%
|
|===================================                                   |  52%
|
|====================================                                  |  53%
|
|====================================                                  |  54%
|
|=====================================                                 |  55%
|
|=====================================                                 |  55%
|
|======================================                                |  56%
|
|======================================                                |  56%
|
|=======================================                               |  57%
|
```


| 58%
| 58%
| 59%
| 60%
| 61%
| 62%
| 62%
| 63%
| 64%
| 65%
| 66%
| 67%
| 68%
| 68%
| 69%
| 69%
| 70%
| 71%
| 72%
| 73%
| 74%
| 75%
| 75%
| 76%
| 77%

```
|==================================================                |  78%
|
|===================================================               |  79%
|
|====================================================              |  80%
|
|=====================================================             |  81%
|
|======================================================            |  81%
|
|=======================================================           |  82%
|
|========================================================          |  82%
|
|=========================================================         |  83%
|
|==========================================================        |  84%
|
|===========================================================       |  85%
|
|============================================================      |  86%
|
|=============================================================     |  87%
|
|==============================================================    |  88%
|
|===============================================================   |  88%
|
|================================================================  |  89%
|
|================================================================= |  90%
|
|==================================================================|  91%
|
|==================================================================|  92%
|
|==================================================================|  92%
|
|==================================================================|  93%
|
|==================================================================|  94%
|
|==================================================================|  94%
|
|==================================================================|  95%
|
|==================================================================|  95%
|
|==================================================================|  96%
|
```

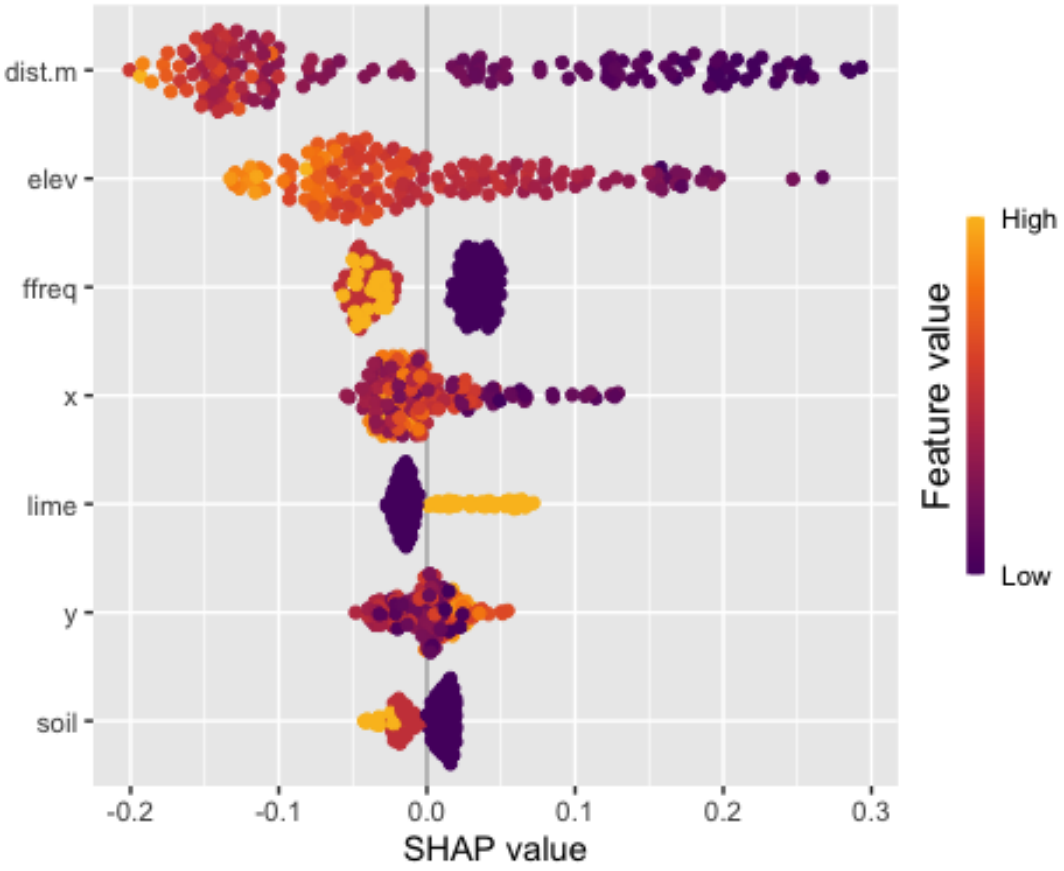
```
|=======================================================================  |  97%
|
|======================================================================== |  98%
|
|========================================================================= |  99%
|
|=========================================================================|  99%
|
|=========================================================================| 100%
```

```
#
str(s)
```

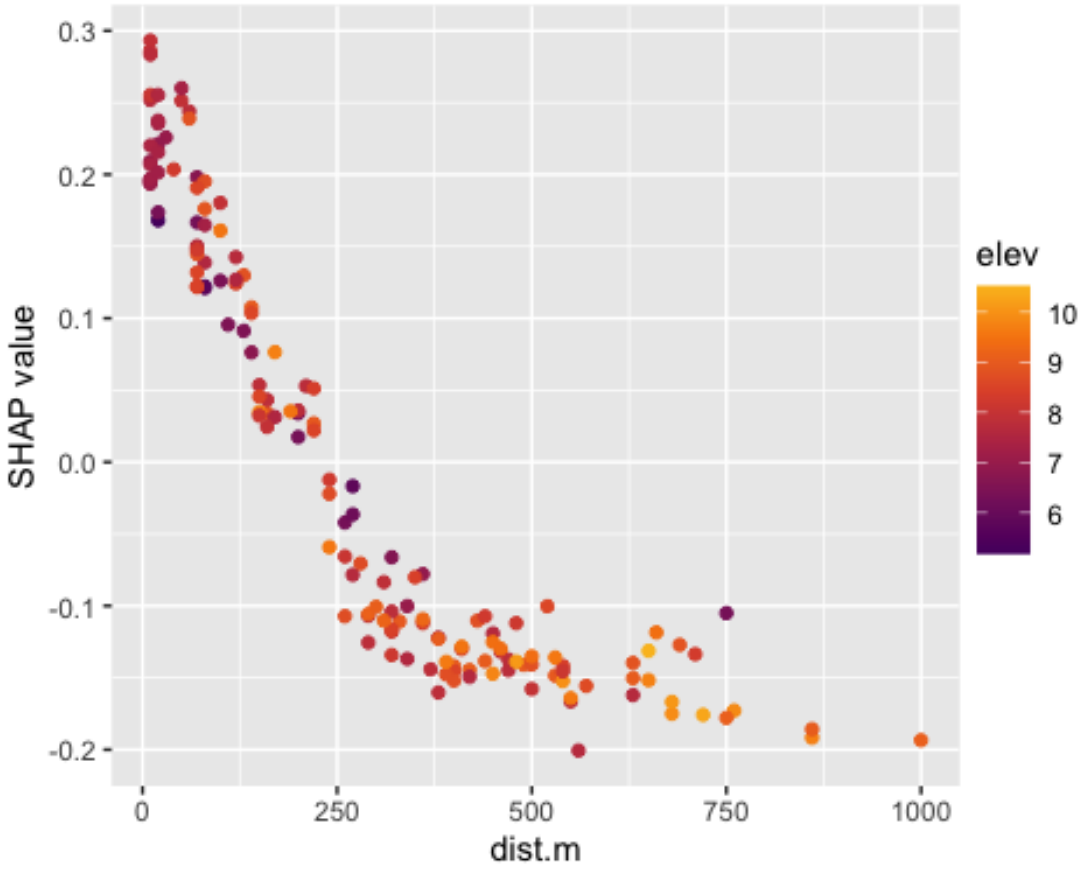
```
## List of 12
##  $ S          : num [1:155, 1:7] 0.0432 0.0413 0.0345 0.0268 0.0275 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:7] "ffreq" "x" "y" "dist.m" ...
##  $ X          :'data.frame': 155 obs. of  7 variables:
##   ..$ ffreq : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ x     : num [1:155] 181072 181025 181165 181298 181307 ...
##   ..$ y     : num [1:155] 333611 333558 333537 333484 333330 ...
##   ..$ dist.m: num [1:155] 50 30 150 270 380 470 240 120 240 420 ...
##   ..$ elev  : num [1:155] 7.91 6.98 7.8 7.66 7.48 ...
##   ..$ soil  : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 1 1 2 ...
##   ..$ lime  : Factor w/ 2 levels "0","1": 2 2 2 1 1 1 1 1 1 1 ...
##  $ baseline   : num 2.56
##  $ SE         : num [1:155, 1:7] 0 0 0 0 0 0 0 0 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:7] "ffreq" "x" "y" "dist.m" ...
##  $ n_iter     : int [1:155] 1 1 1 1 1 1 1 1 1 1 ...
##  $ converged  : logi [1:155] TRUE TRUE TRUE TRUE TRUE TRUE ...
##  $ m          : int 14
##  $ m_exact    : int 126
##  $ prop_exact : num 1
##  $ exact      : logi TRUE
##  $ txt        : chr "Exact Kernel SHAP values"
##  $ predictions: num [1:155, 1] 2.99 3.02 2.75 2.47 2.44 ...
##  - attr(*, "class")= chr "kernelshap"
```

Compute the visualization object and show the same two diagnostic plots. Note that `shapviz` understands `kernelshap` objects, so there is no need to re-format the Shapley values matrix.

```
sv <- shapviz(s)
sv_importance(sv, kind = "bee")
```




```
# auto-select most important interacting feature
sv_dependence(sv, v = "dist.m", color_var = "auto")
```



This is slightly different than the plots from the `fastshap` computation.