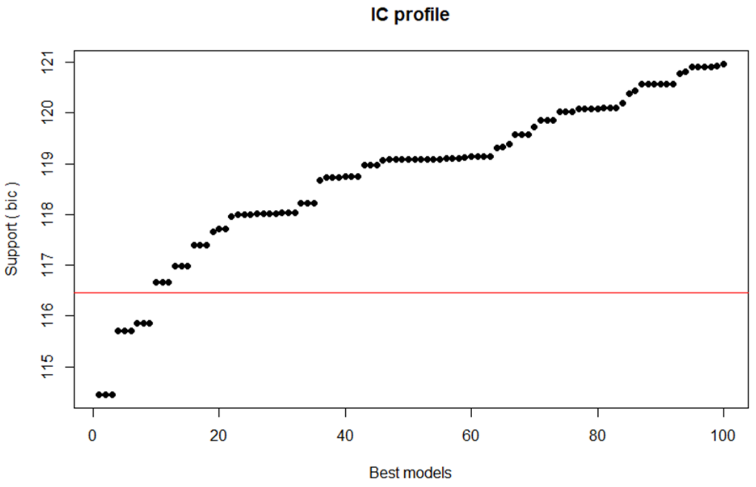
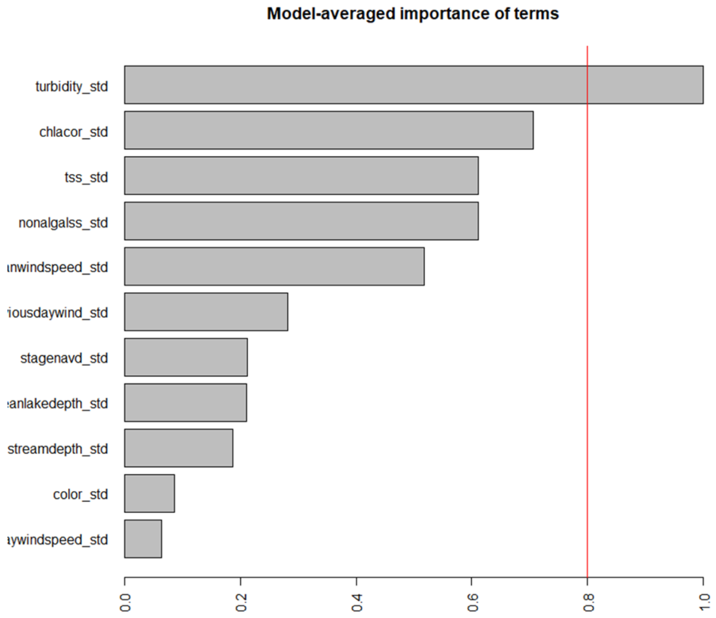
Advanced R: Statistical Machine Learning

Dan Schmutz, MS
Chief Environmental Scientist

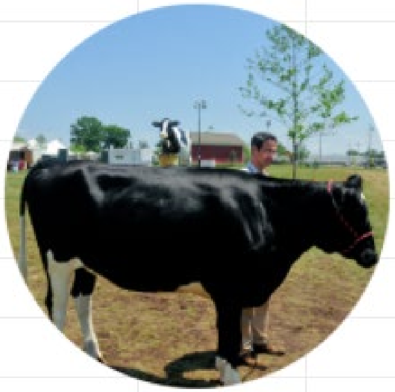
Zoom Workshop for SJRWMD
September 24, 2020





Ensembles and Random Forest

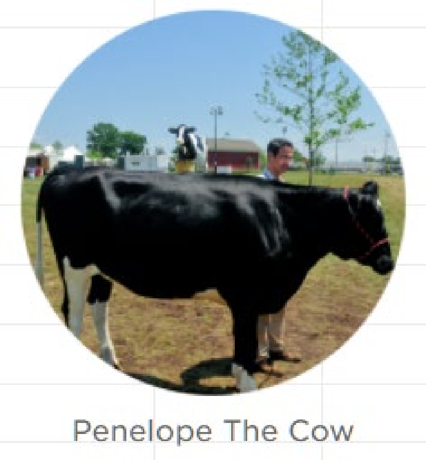
How much does this cow weigh?



Penelope The Cow

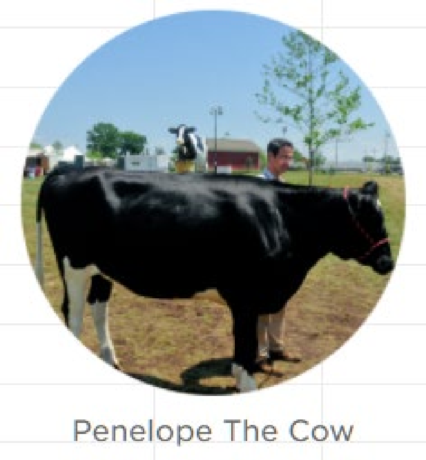
How much does this cow weigh?

- Google says depends on gender…

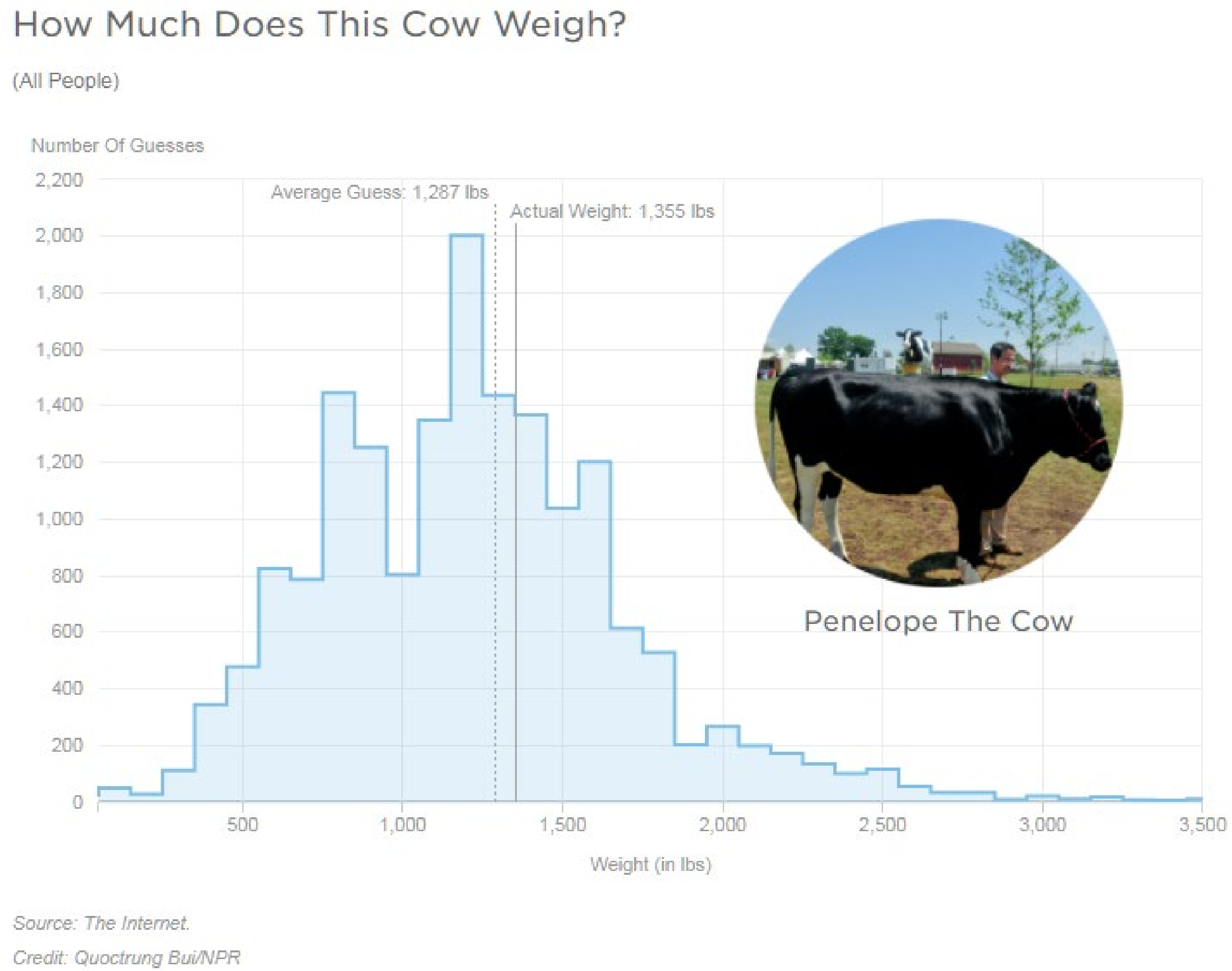


How much does this cow weigh?

- Google says depends on gender…sounds like Google has been doing some machine learning…



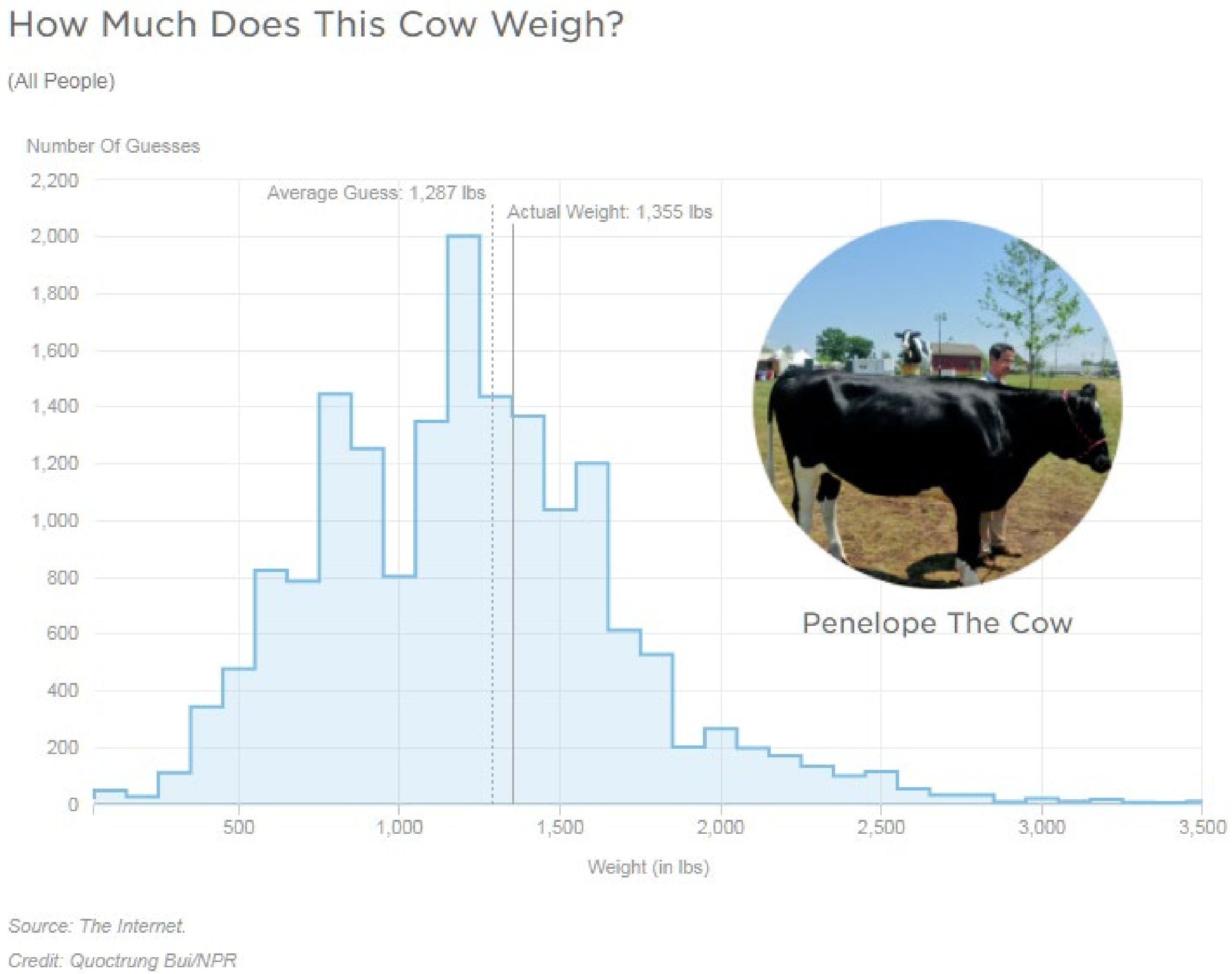
The wisdom of crowds in machine learning: ensemble techniques (bagging and boosting)



https://www.npr.org/sections/money/2015/08/07/429720443/17-205-people-guessed-the-weight-of-a-cow-heres-how-they-did

GPI

The wisdom of crowds in machine learning: ensemble techniques (bagging and boosting)



Idea is that a group of "weak learners" will average out to a better answer than a typical expert!

Can think of this as the large positive and negative errors of the "uneducated" canceling out.

https://www.npr.org/sections/money/2015/08/07/429720443/17-205-people-guessed-the-weight-of-a-cow-heres-how-they-did

GPI

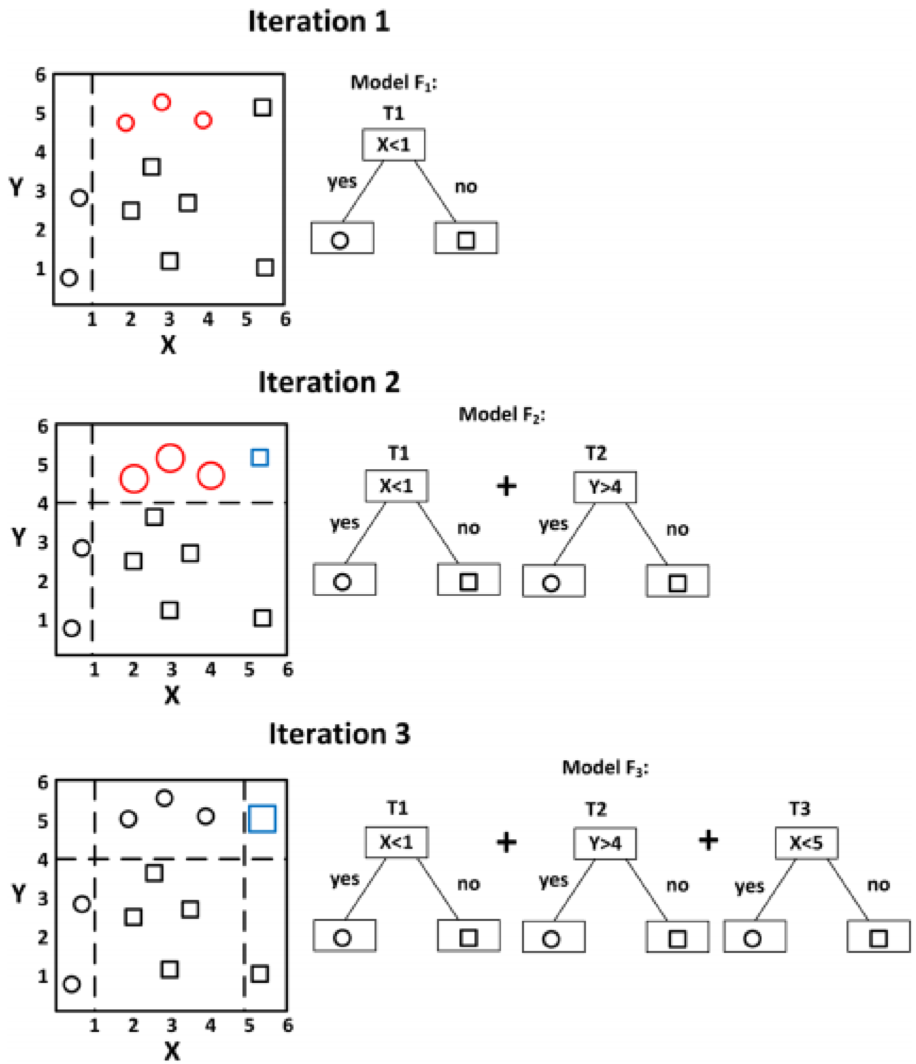
Types of ensembles: bagging

- Bagging – bootstrap aggregation. Subsample rows of a dataset (bootstrapping) repeatedly and then average responses (regression) or majority vote (classification).
  - Decreases the variance in the prediction by generating additional data from training dataset (sampling with replacement)
  - Interestingly, by chance if you sample the same number of rows with replacement on average each bootstrap sample contains 63.2% of the original observations and omits 36.8%
  - The omitted observations from each sample are called out-of-bag observations and can be used as a kind of hold-out sample

Types of ensembles: boosting

XGBoost is a superstar in the machine learning world!

- Boosting – fitting successive models which address residual variation by increasing the weight of poorly fit observations in successive models. This is a sequential method.



https://www.researchgate.net/publication/326379229_Exploring_the_clinical_features_of_narcolepsy_type_1_versus_narcolepsy_type_2_from_European_Narcolepsy_Network_database_with_machine_learning

GPI

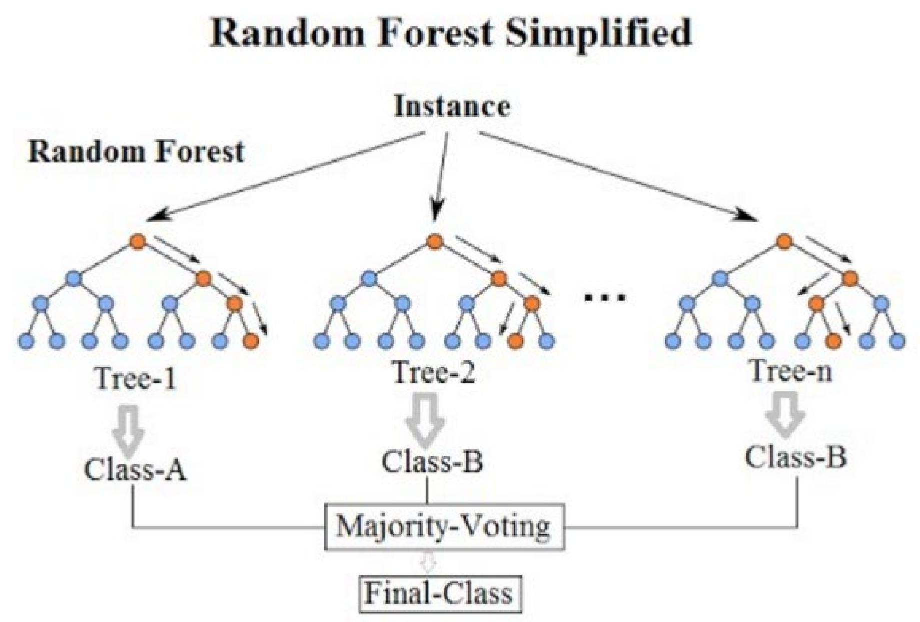
Types of ensembles: stacking

- Stacking – a method of combining heterogenous models. Train several models, then train a metamodel on the outputs of the group of models.

GPI

Random Forest Algorithm: a modified bagged decision tree

- Can be used for classification or regression
- Creates “forest” of decision trees using subset of cases and variables (bootstrap aggregation or “bagging”)
- Predictions are made by running case through all trees (e.g., 500) and taking an average for regression
- Each tree is grown with only about 60% of the data, so the remaining data provide a realistic “out-of-bag” error estimate
- Robust to outliers and noise
- Allows complex interactions
- No statistical assumptions
- Handles datasets >1000 variables
- Avoids overfitting
- Minimal tuning required
- Limitations
  - Can’t extrapolate
  - May overestimate lows/underestimate highs



https://community.tibco.com/wiki/random-forest-template-tibco-spotfirer-wiki-page

GPI

Random Forest Algorithm Pseudocode

```
1.  Given a training data set
2.  Select number of trees to build (n_trees)
3.  for i = 1 to n_trees do
4.  |  Generate a bootstrap sample of the original data
5.  |  Grow a regression/classification tree to the bootstrapped data
6.  |  for each split do
7.  |  | Select m_try variables at random from all p variables
8.  |  | Pick the best variable/split-point among the m_try
9.  |  | Split the node into two child nodes
10. |  end
11. | Use typical tree model stopping criteria to determine when a
    | tree is complete (but do not prune)
12. end
13. Output ensemble of trees
```

Random Forest hyperparameters worth tuning

- mtry – number of variables tried at each split on tree, typical default values are
  - Number of parameters/3 (regression)
  - Number of paramerers^0.5 (classification)
- ntree – number of trees in the forest
  - default value is 500
  - may be useful to tune mtry at 500 trees, then boost number of trees to see if any improvement

GPI

Less frequently tuned hyperparameters

- The complexity of each tree
  - node size, max depth, max number of terminal nodes, or the required node size to allow additional splits
- The sampling scheme
  - Bootstrap with replacement is the norm but if you have many categorical features with a varying number of levels, sampling with replacement can lead to biased variable split selection
- The splitting rule to use during tree construction
  - The splitting is based usually on Gini impurity (in the case of classification) and the SSE (in case of regression), but it can be changed.

GPI

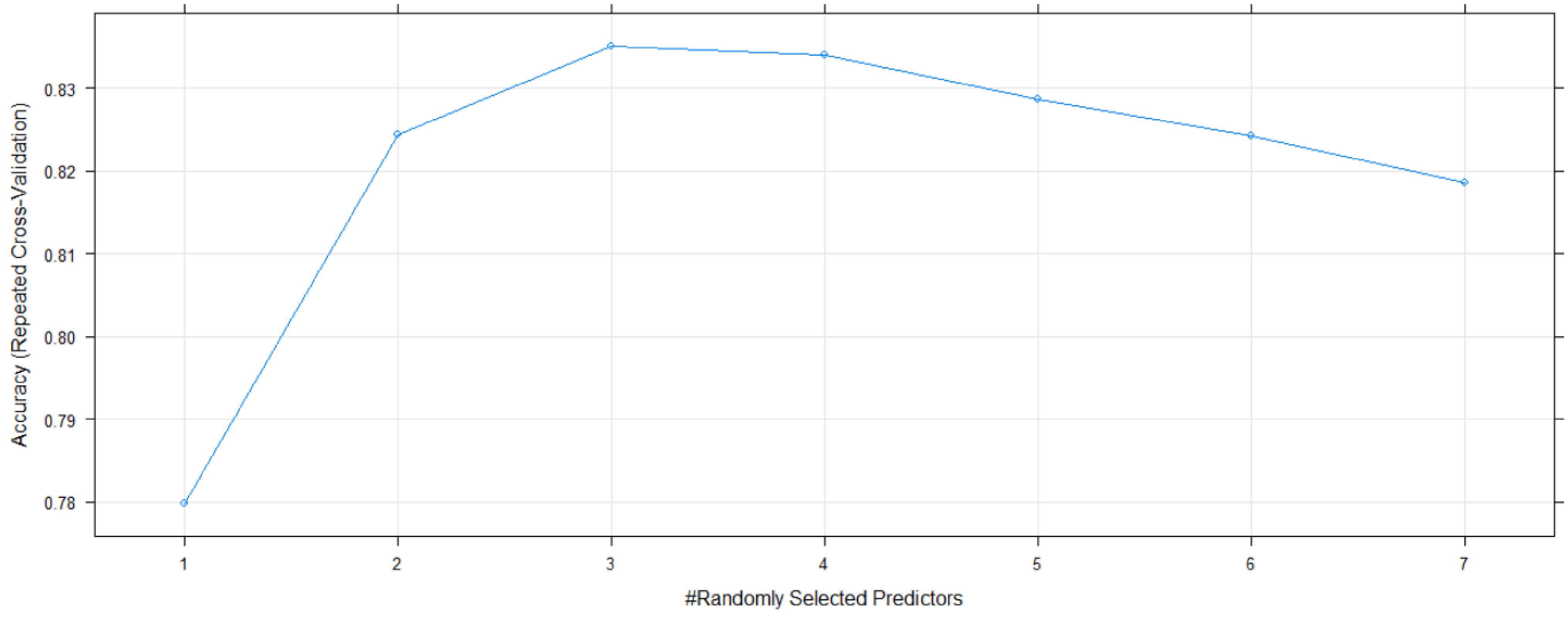
Tuning mtry using caret on ttrain4

```
# code below used to identify optimal mtry hyperparameter for tuning using caret
seed<-42
set.seed(seed)
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid", classProbs = TRUE)
tunegrid <- expand.grid(.mtry=c(1:7))
metric<-"Accuracy"
rf_gridsearch <- train(Survived2~., data= ttrain4rf, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control)

print(rf_gridsearch)
plot(rf_gridsearch)
print(rf_gridsearch)
```

GPI

mtry=3 results in highest accuracy

plot(rf_gridsearch)



Out-of-bag performance on ttrain4, ntree=500

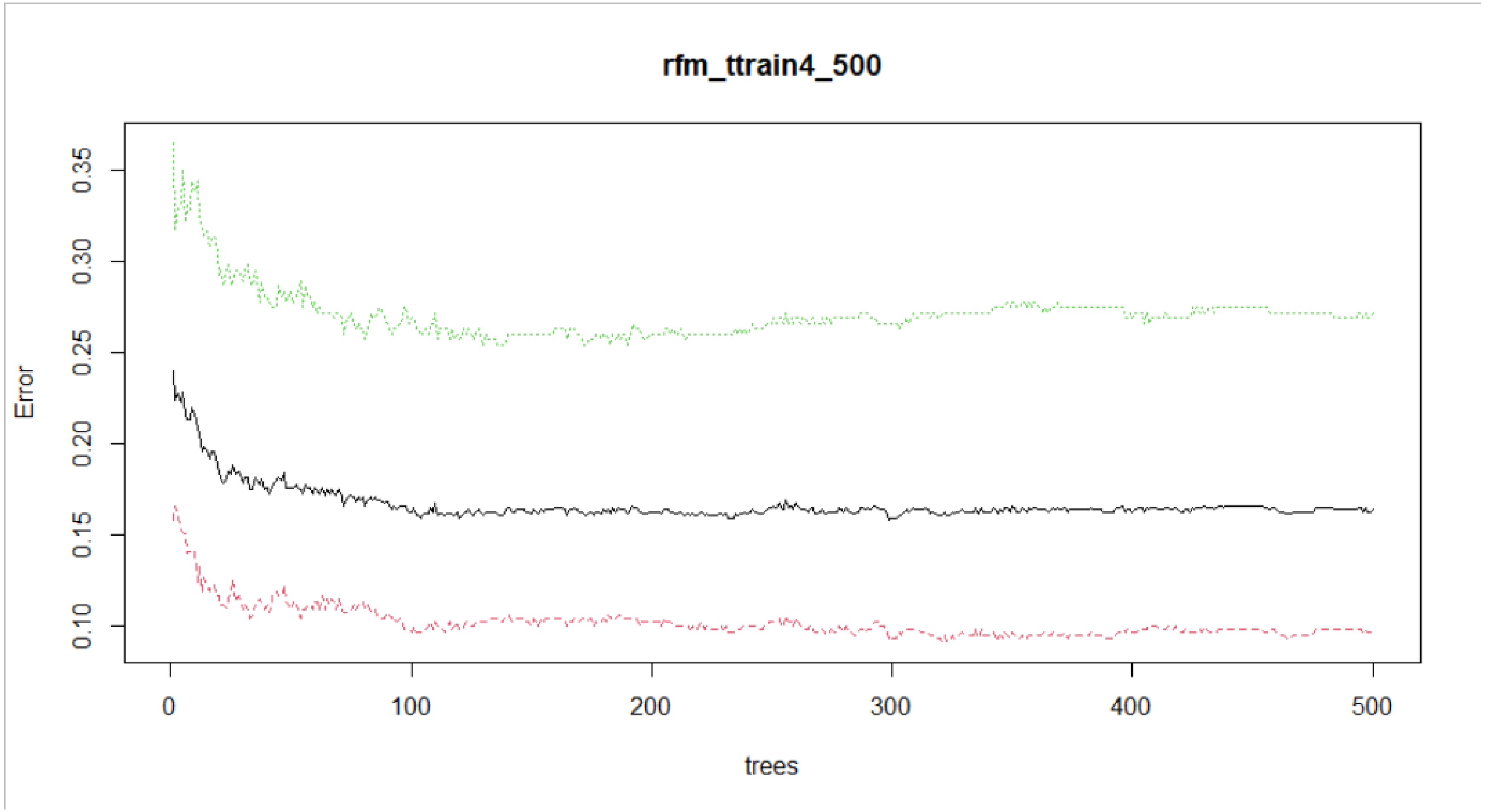
- OOB accuracy of 100-16.39 = 83.61%
- OOB is usually conservative compared to test set results

```
> print(rfm_ttrain4_500) # view OOB estimate of error rate

Call:
 randomForest(formula = Survived2 ~ ., data = ttrain4rf, mtry = 3,      ntree = 500)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 16.39%
Confusion matrix:
    X0  X1 class.error
X0 496  53  0.09653916
X1  93 249  0.27192982
> 
```

GPI

Plotting the randomForest object gives idea of changing error rates with ntree

plot(rfm_ttrain4_500)



Out-of-bag performance on ttrain4 for ntree=10000

- OOB accuracy of 100-16.61 = 83.39%
- No improvement over model with default 500 trees in this case

```
> print(rfm_ttrain4_10000) # view OOB estimate of error rate

Call:
 randomForest(formula = Survived2 ~ ., data = ttrain4rf, mtry = 3,      ntree = 10000)
               Type of random forest: classification
                     Number of trees: 10000
No. of variables tried at each split: 3

        OOB estimate of  error rate: 16.61%
Confusion matrix:
    X0  X1 class.error
X0 494  55   0.1001821
X1  93 249   0.2719298
> plot(rfm_ttrain4_10000)
```

GPI

Test set performance ntree=500

confusionMatrix(pred_rfm_500_test,ttest4rf$Survived2) # evaluating performance on out-of-sample test dataset

- 76% accuracy
- Not as good as knn

```
> confusionMatrix(pred_rfm_500_test,ttest4rf$Survived2) #
 evaluating performance on out-of-sample test dataset
Confusion Matrix and Statistics

          Reference
Prediction  X0  X1
        X0 210  54
        X1  40  91

               Accuracy : 0.762
                 95% CI : (0.7169, 0.8032)
    No Information Rate : 0.6329
    P-Value [Acc > NIR] : 2.664e-08

                  Kappa : 0.4773

 Mcnemar's Test P-Value : 0.18

            Sensitivity : 0.8400
            Specificity : 0.6276
         Pos Pred Value : 0.7955
         Neg Pred Value : 0.6947
             Prevalence : 0.6329
         Detection Rate : 0.5316
   Detection Prevalence : 0.6684
      Balanced Accuracy : 0.7338

       'Positive' Class : X0

>
```

GPI

Test set performance ntree=10000

confusionMatrix(pred_rfm_10000_test,ttest4rf$Survived2)

```
> confusionMatrix(pred_rfm_10000_test,ttest4rf$Survived2)

Confusion Matrix and Statistics

          Reference
Prediction  X0  X1
        X0 212  53
        X1  38  92

               Accuracy : 0.7696
                 95% CI : (0.7249, 0.8103)
    No Information Rate : 0.6329
    P-Value [Acc > NIR] : 3.776e-09

                  Kappa : 0.4932

 Mcnemar's Test P-Value : 0.1422

            Sensitivity : 0.8480
            Specificity : 0.6345
         Pos Pred Value : 0.8000
         Neg Pred Value : 0.7077
             Prevalence : 0.6329
         Detection Rate : 0.5367
   Detection Prevalence : 0.6709
      Balanced Accuracy : 0.7412

       'Positive' Class : X0
```

GPI

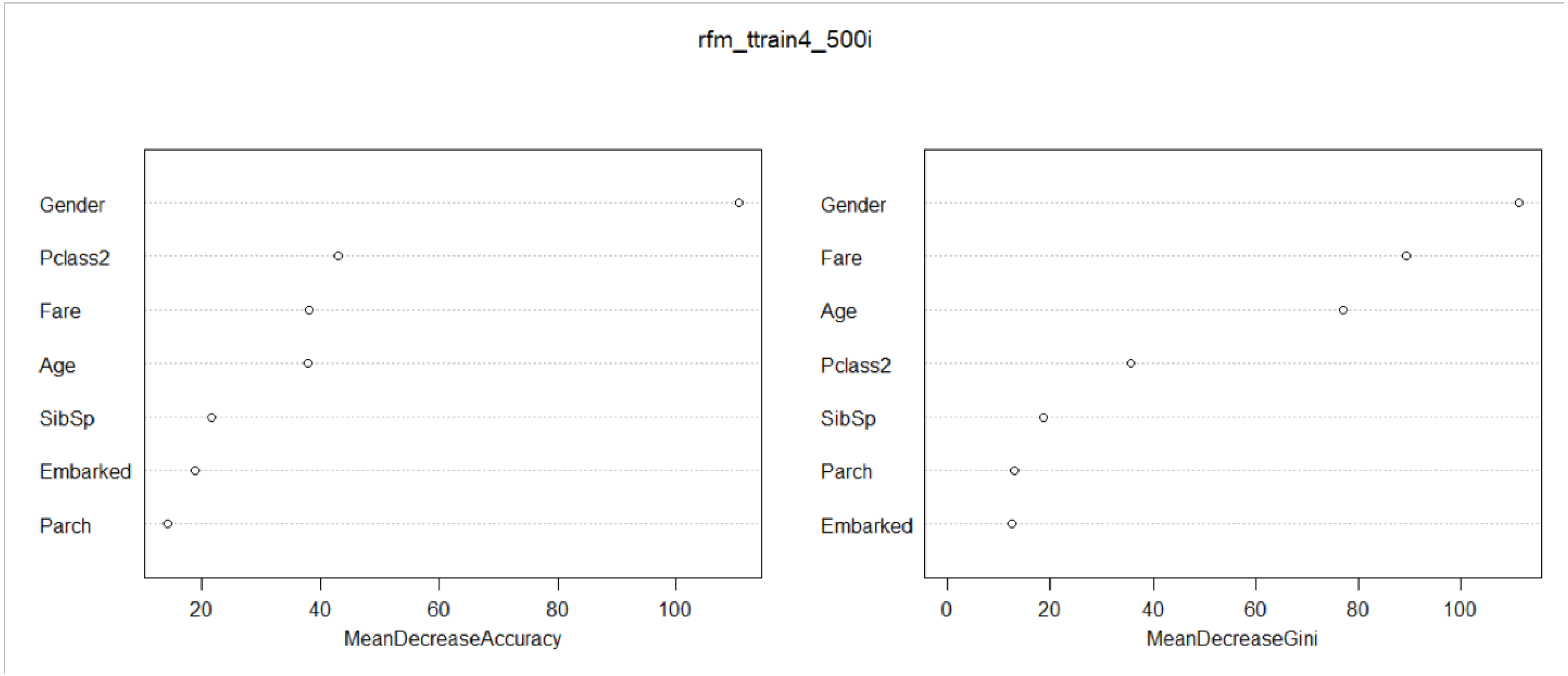
Peeking into the random forest– a “black box” model

GPI

Two types of importance available for randomForest

rfm_ttrain4_500i<-randomForest(Survived2~.,ttrain4rf,mtry=3,ntree=500, importance=T)

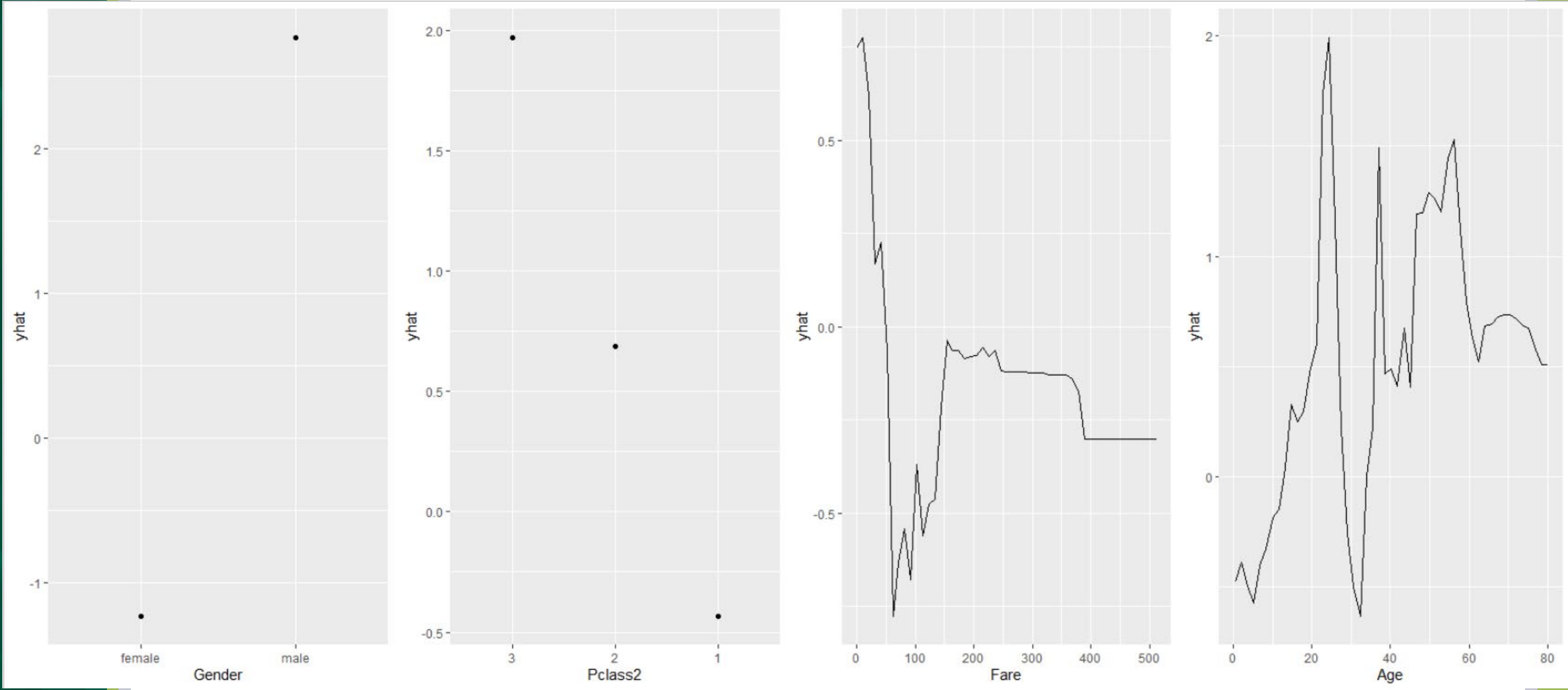
- varImpPlot(rfm_ttrain4_500i)



Randomization-based

Importance for tree-splitting

Dependence plots



randomForest on ames train_3

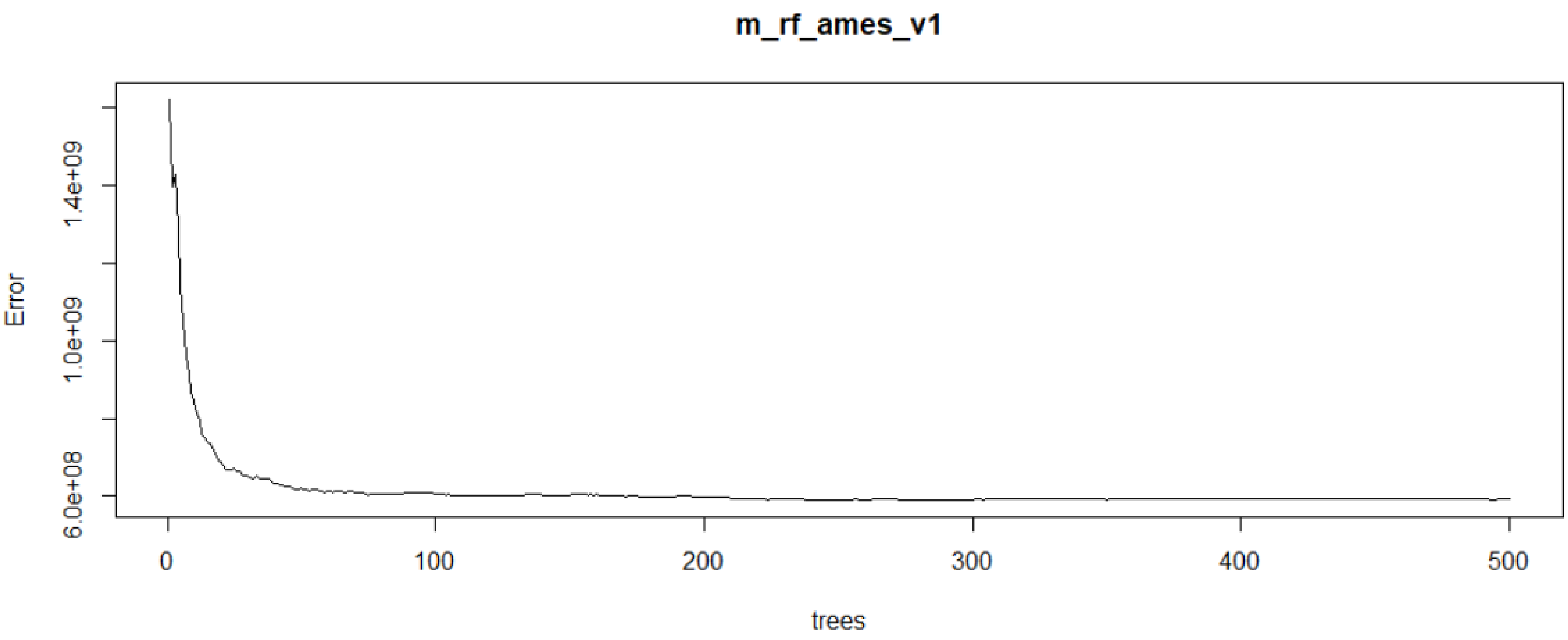
- 90.51% out-of-bag estimate of variance explained

```
> m_rf_ames_v1<-randomForest(Sale_Price~., data=train_3)
> print(m_rf_ames_v1)

Call:
 randomForest(formula = Sale_Price ~ ., data = train_3)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 26

          Mean of squared residuals: 592460699
                    % Var explained: 90.51
> 
```

GPI

Results from ames train

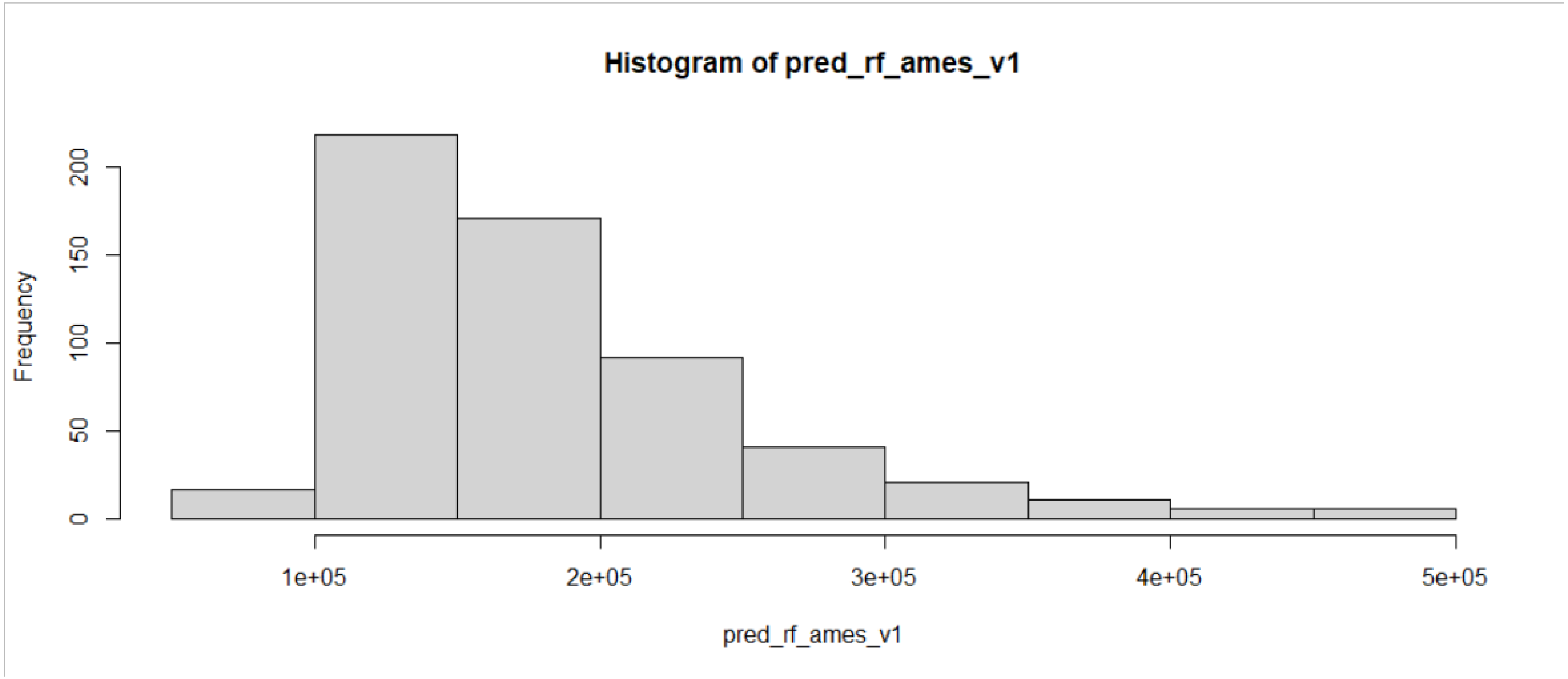
plot(m_rf_ames_v1)



GPI

Predicting test_3

```
pred_rf_ames_v1<-predict(m_rf_ames_v1,newdata=test_3)
hist(pred_rf_ames_v1)
```



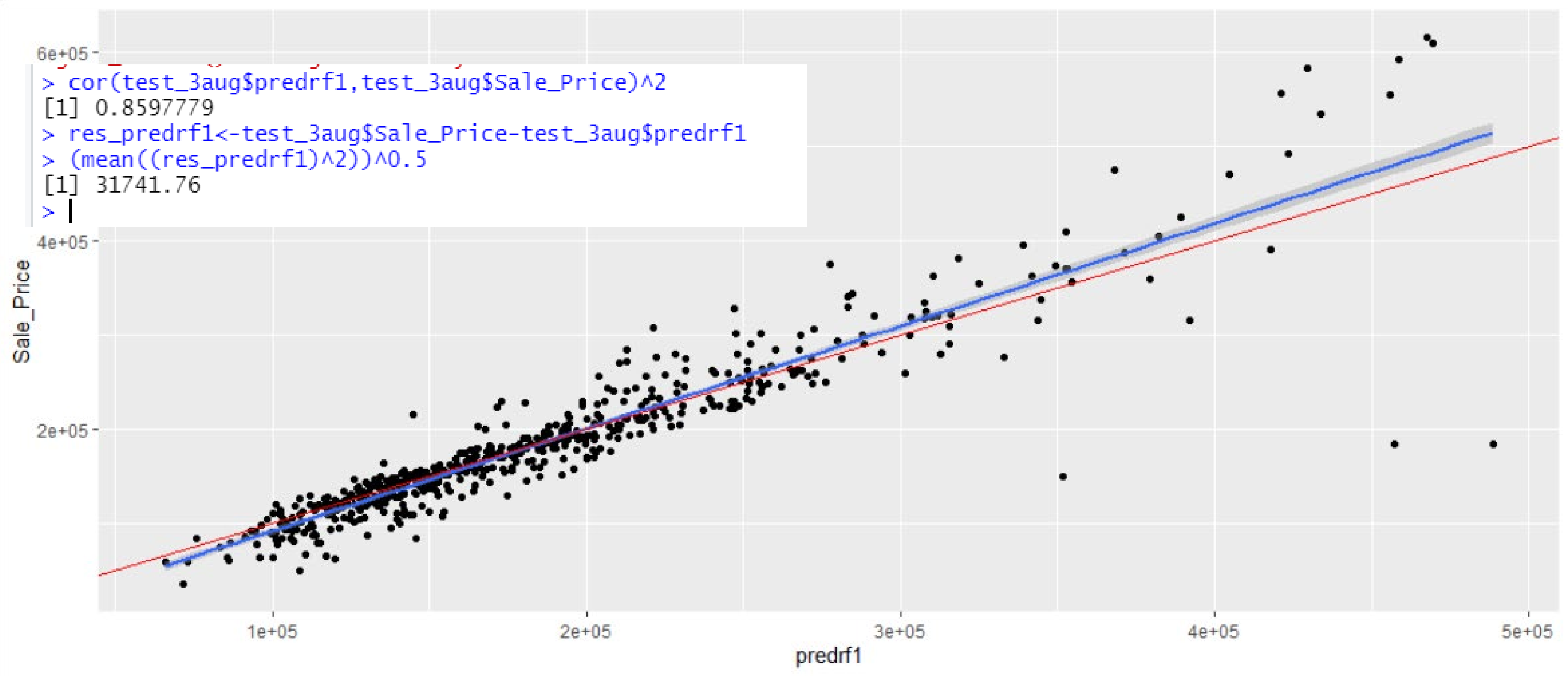
rf train and test RMSE

```
> RMSE(m_rf_ames_v1$predicted, train_3$Sale_Price)
[1] 24340.52
> RMSE(pred_rf_ames_v1, test_3$Sale_Price)
[1] 31741.76
> test_3aug$predrf1<-pred_rf_ames_v1
> ggplot(test_3aug,aes(x=predrf1,y=Sale_Price))+
+   geom_point()+stat_smooth(method=lm)+geom_abline(slope=1, intercept=0, col
='red')
`geom_smooth()` using formula 'y ~ x'
> cor(test_3aug$predrf1,test_3aug$Sale_Price)^2
[1] 0.8597779
> res_predrf1<-test_3aug$Sale_Price-test_3aug$predrf1
> (mean((res_predrf1)^2))^0.5
[1] 31741.76
>
```

GPI

rf excellent performance on test set with RMSE 31741.76 and r^2 of 6%

- randomForest algorithm well suited to this problem



GPI