Educational Data Mining with R

Okan Bulut

6/25/2019

Packages

```
# Packages to be used
packages <- c("data.table", "tidyverse","e1071", "caret", "mlr",
              "modelr", "randomForest", "rpart", "rpart.plot",
              "GGally", "ggExtra")

install.packages(packages)

# Activate all packages
library("data.table")
library("tidyverse")
library("e1071")
library("caret")
library("mlr")
library("modelr")
library("randomForest")
library("rpart")
library("rpart.plot")
library("GGally")
library("ggExtra")
```

PISA Dataset

```
# fread function from data.table
pisa <- fread("pisa_turkey.csv", na.strings = "")
class(pisa)
```

```
[1] "data.table" "data.frame"
```

```
# Base R function for csv files (DON'T RUN)
pisa <- read.csv("pisa_turkey.csv", header = TRUE)
class(pisa)
```

```
[1] "data.frame"
```

```
# See variable names
names(pisa)
```

```
 [1] "CNTSTUID"    "gender"      "female"      "grade"       "computer"
 [6] "software"    "internet"    "desk"        "own.room"    "quiet.study"
[11] "lit"         "poetry"      "art"         "book.sch"    "tech.book"
[16] "dict"        "art.book"    "ST011Q05TA"  "ST071Q02NA"  "ST071Q01NA"
[21] "ST123Q02NA"  "ST082Q01NA"  "ST119Q01NA"  "ST119Q05NA"  "ANXTEST"
[26] "COOPERATE"   "BELONG"      "EMOSUPS"     "WEALTH"      "PARED"
[31] "TMINS"       "ESCS"        "TEACHSUP"    "TDTEACH"     "IBTEACH"
[36] "SCIEEFF"     "math"        "reading"     "science"
```

```
# Preview the data
head(pisa)
```

```
  CNTSTUID gender female   grade computer software internet desk own.room
1 79200939 Female      1 Grade 9        0        0        0    1        1
2 79206774 Female      1 Grade 9        1        1        1    1        1
3 79204670 Female      1 Grade 9        0        0        1    1        0
4 79201647 Female      1 Grade 9        1        1        0    1        0
5 79203718 Female      1 Grade 9        0        0        0    0        0
6 79204968 Female      1 Grade 9        0        0        0    1        0
  quiet.study lit poetry art book.sch tech.book dict art.book ST011Q05TA
1           1   1      1   0        1         0    1        0         No
2           0   1      0   1        1         0    1        0        Yes
3           0   1      0   0        0         0    1        0         No
4           0   1      1   0        0         0    1        0        Yes
5           0   0      0   0        0         0    1        0         No
6           0   0      1   0        0         0    1        0         No
  ST071Q02NA ST071Q01NA     ST123Q02NA        ST082Q01NA     ST119Q01NA
1         11          7 Strongly agree             Agree Strongly agree
2         NA          5 Strongly agree             Agree Strongly agree
3          6          4          Agree             Agree Strongly agree
4          6          3          Agree          Disagree Strongly agree
5          6          6 Strongly agree          Disagree          Agree
6         12          6       Disagree Strongly disagree Strongly agree
      ST119Q05NA ANXTEST COOPERATE  BELONG EMOSUPS  WEALTH PARED TMINS
1 Strongly agree  1.4394    1.3264 -0.8482  0.5658 -1.7154    12  1560
2          Agree  0.6654    1.1640 -0.7657  1.0991 -1.2426     8   900
3 Strongly agree  1.2704    0.5759 -0.5064 -1.3298 -1.2256    12  1600
4 Strongly agree  2.5493    0.9675  0.3142 -0.3306 -1.8473    16  2280
5 Strongly agree  1.1311   -0.2882 -0.6925 -0.2495 -4.7851     4  1600
6 Strongly agree  1.1311   -1.0629 -0.9932 -1.8280 -2.8012    14  1495
     ESCS TEACHSUP TDTEACH IBTEACH SCIEEFF     math  reading  science
1 -1.7902   0.7900  0.9505  0.8519  1.8526 361.9234 408.0202 394.2937
2 -1.8259   1.4475  0.6580 -1.0496 -0.2154 325.4144 375.4197 350.5305
3 -1.2038  -0.1164  0.4505  0.5453  0.5561 365.5649 381.6692 396.2027
4 -0.9068  -0.4527 -0.0057  1.4812  0.5037 333.6344 345.9447 336.3870
5 -4.1131   0.2725  0.0615  0.9054  0.1091 381.7032 381.6998 403.6358
6 -1.0631  -1.0080 -0.8780  0.0441 -1.4295 352.0996 350.1369 393.2887
```

```
# Dimensions
dim(pisa)
```

```
[1] 5895   39
```

```
# Structure
str(pisa)
```

```
'data.frame':   5895 obs. of  39 variables:
 $ CNTSTUID   : int  79200939 79206774 79204670 79201647 79203718 79204968 79200200 7920
0563 79200129 79205586 ...
 $ gender     : Factor w/ 2 levels "Female","Male": 1 1 1 1 1 1 1 1 1 1 ...
 $ female     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ grade      : Factor w/ 6 levels "Grade 10","Grade 11",..: 6 6 6 6 6 6 6 6 1 1 ...
 $ computer   : int  0 1 0 1 0 0 0 1 0 1 ...
 $ software   : int  0 1 0 1 0 0 1 1 0 NA ...
 $ internet   : int  0 1 1 0 0 0 0 0 0 1 ...
 $ desk       : int  1 1 1 1 0 1 0 0 0 1 ...
 $ own.room   : int  1 1 0 0 0 0 0 0 0 1 ...
 $ quiet.study: int  1 0 0 0 0 0 1 0 1 1 ...
 $ lit        : int  1 1 1 1 0 0 0 0 0 1 ...
 $ poetry     : int  1 0 0 1 0 1 0 0 0 NA ...
 $ art        : int  0 1 0 0 0 0 0 0 0 NA ...
 $ book.sch   : int  1 1 0 0 0 0 0 1 0 1 ...
 $ tech.book  : int  0 0 0 0 0 0 0 0 NA NA ...
 $ dict       : int  1 1 1 1 1 1 1 1 1 1 ...
 $ art.book   : int  0 0 0 0 0 0 0 0 0 1 ...
 $ ST011Q05TA : Factor w/ 2 levels "No","Yes": 1 2 1 2 1 1 2 2 1 NA ...
 $ ST071Q02NA : int  11 NA 6 6 6 12 6 6 NA 3 ...
 $ ST071Q01NA : int  7 5 4 3 6 6 7 4 NA 10 ...
 $ ST123Q02NA : Factor w/ 4 levels "Agree","Disagree",..: 3 3 1 1 3 2 3 3 1 3 ...
 $ ST082Q01NA : Factor w/ 4 levels "Agree","Disagree",..: 1 1 1 2 2 4 2 4 2 3 ...
 $ ST119Q01NA : Factor w/ 4 levels "Agree","Disagree",..: 3 3 3 3 1 3 3 3 3 3 ...
 $ ST119Q05NA : Factor w/ 4 levels "Agree","Disagree",..: 3 1 3 3 3 3 3 3 3 3 ...
 $ ANXTEST    : num  1.439 0.665 1.27 2.549 1.131 ...
 $ COOPERATE  : num  1.326 1.164 0.576 0.968 -0.288 ...
 $ BELONG     : num  -0.848 -0.766 -0.506 0.314 -0.693 ...
 $ EMOSUPS    : num  0.566 1.099 -1.33 -0.331 -0.249 ...
 $ WEALTH     : num  -1.72 -1.24 -1.23 -1.85 -4.79 ...
 $ PARED      : int  12 8 12 16 4 14 8 4 4 8 ...
 $ TMINS      : int  1560 900 1600 2280 1600 1495 1600 1600 NA 1600 ...
 $ ESCS       : num  -1.79 -1.826 -1.204 -0.907 -4.113 ...
 $ TEACHSUP   : num  0.79 1.448 -0.116 -0.453 0.273 ...
 $ TDTEACH    : num  0.9505 0.658 0.4505 -0.0057 0.0615 ...
 $ IBTEACH    : num  0.852 -1.05 0.545 1.481 0.905 ...
 $ SCIEEFF    : num  1.853 -0.215 0.556 0.504 0.109 ...
 $ math       : num  362 325 366 334 382 ...
 $ reading    : num  408 375 382 346 382 ...
 $ science    : num  394 351 396 336 404 ...
```

Variable	Description	Variable	Description
gender	Gender	ST123Q02NA	Whether parents support educational efforts

Variable	Description	Variable	Description
female	Female	ST082Q01NA	Prefering working in a team over working alone
grade	Grade	ST119Q01NA	Wanting top grades in most or all courses
computer	Computer at home?	ST119Q05NA	Wanting to the best student in class
software	Software at home?	ANXTEST	Test anxiety
internet	Internet at home?	COOPERATE	Enjoying cooperation
desk	Desk at home?	BELONG	Sense of belonging to school
own.room	Own a room at home?	EMOSUPS	Parents emotional support
quiet.study	Quiet study area at home?	WEALTH	Family wealth
lit	Interest in literature	PARED	Highest parental education in years of schooling
poetry	Interest in poetry	TMINS	Total learning time per week
art	Interest in art	ESCS	Index of economic, social and cultural status
book.sch	School books	TEACHSUP	Teacher support in science classes
tech.book	Technical books	TDTEACH	Teacher-directed science instruction
dict	Dictionary	IBTEACH	Inquiry based science instruction
art.book	Art book	SCIEEFF	SCIEEFF
ST011Q05TA	Software at home?	math	Students math scores in PISA 2015
ST071Q02NA	Time spent for learning math	reading	Students reading scores in PISA 2015
ST071Q01NA	Time spent for learning science	science	Students science scores in PISA 2015

Data Wrangling and Visualization

Let’s define a few additional variables here using the `dplyr` package.

```
pisa <- mutate(pisa,

               # Reorder the levels of grade
               grade = factor(grade,
                              levels = c("Grade 7", "Grade 8", "Grade 9", "Grade 10",
                                         "Grade 11", "Grade 12", "Grade 13",
                                         "Ungraded")),

               # Define a numerical grade variable
               grade1 = (as.numeric(sapply(grade, function(x) {
                 if(x=="Grade 7") "7"
                 else if (x=="Grade 8") "8"
                 else if (x=="Grade 9") "9"
                 else if (x=="Grade 10") "10"
                 else if (x=="Grade 11") "11"
                 else if (x=="Grade 12") "12"
                 else if (x=="Grade 13") NA_character_
                 else if (x=="Ungraded") NA_character_}))),

               # Total learning time as hours
               learning = round(TMINS/60, 0),

              # Science performance based on OECD average
              science_oecd = as.factor(ifelse(science >= 493, "High", "Low")),

              # Science performance based on Turkey's average
              science_tr = as.factor(ifelse(science >= 422, "High", "Low")))
```

We can summarize the target variable (science) by categorical independent variables and see if there is any relationship.

```
# By grade
pisa %>%
  group_by(grade) %>%
  summarise(Count = n(),
            science = mean(science, na.rm = TRUE),
            computer = mean(computer, na.rm = TRUE),
            software = mean(software, na.rm = TRUE),
            internet = mean(internet, na.rm = TRUE),
            own.room = mean(own.room, na.rm = TRUE)
            )
```

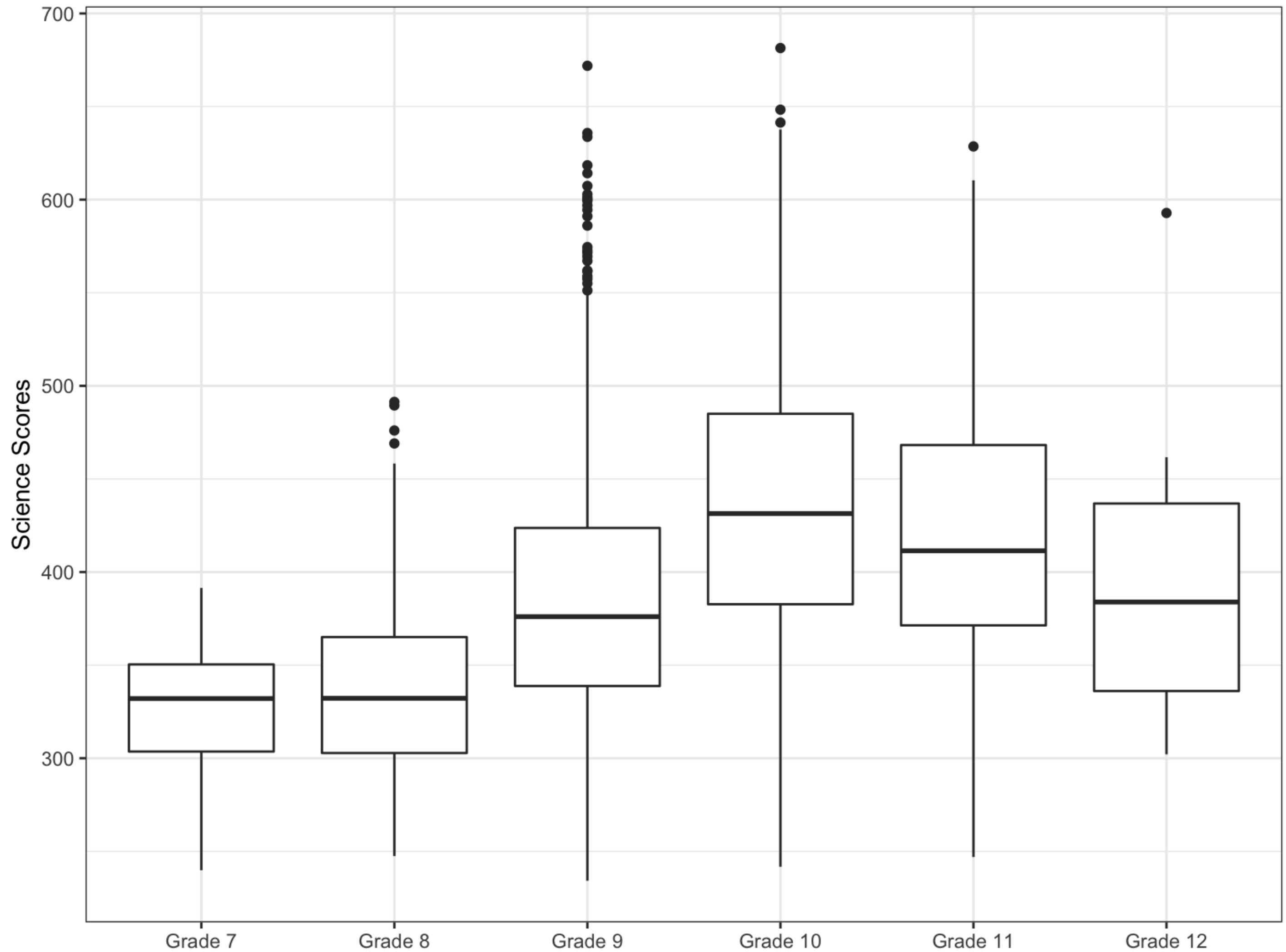
```
# A tibble: 6 x 7
  grade    Count science computer software internet own.room
  <fct>    <int>   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
1 Grade 7     16    328.   0.0625    0.312   0.0625    0.438
2 Grade 8    105    338.   0.303     0.385   0.26      0.324
3 Grade 9   1273    386.   0.609     0.427   0.571     0.657
4 Grade 10  4308    435.   0.710     0.422   0.666     0.740
5 Grade 11   186    418.   0.514     0.324   0.455     0.626
6 Grade 12     7    404.   0.571     0.429   0.286     0.714
```

```
# By grade and gender
pisa %>%
  group_by(grade, gender) %>%
  summarise(Count = n(),
            science = mean(science, na.rm = TRUE),
            computer = mean(computer, na.rm = TRUE),
            software = mean(software, na.rm = TRUE),
            internet = mean(internet, na.rm = TRUE),
            own.room = mean(own.room, na.rm = TRUE)
            )
```

```
# A tibble: 12 x 8
# Groups:   grade [6]
   grade    gender Count science computer software internet own.room
   <fct>    <fct>  <int>   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
 1 Grade 7  Female     6    339.    0        0.333    0        0.333
 2 Grade 7  Male      10    322.    0.1      0.3      0.1      0.5
 3 Grade 8  Female    42    338.    0.275    0.436    0.220    0.268
 4 Grade 8  Male      63    338.    0.322    0.351    0.288    0.361
 5 Grade 9  Female   494    388.    0.602    0.435    0.551    0.651

 6 Grade 9  Male     779    385.    0.613    0.422    0.584    0.660
 7 Grade 10 Female  2272    434.    0.710    0.459    0.671    0.750
 8 Grade 10 Male    2036    437.    0.711    0.382    0.660    0.729
 9 Grade 11 Female   120    421.    0.504    0.342    0.439    0.633
10 Grade 11 Male      66    412.    0.532    0.288    0.484    0.613
11 Grade 12 Female     4    408.    0.75     0.5      0.5      1
12 Grade 12 Male       3    397.    0.333    0.333    0        0.333
```

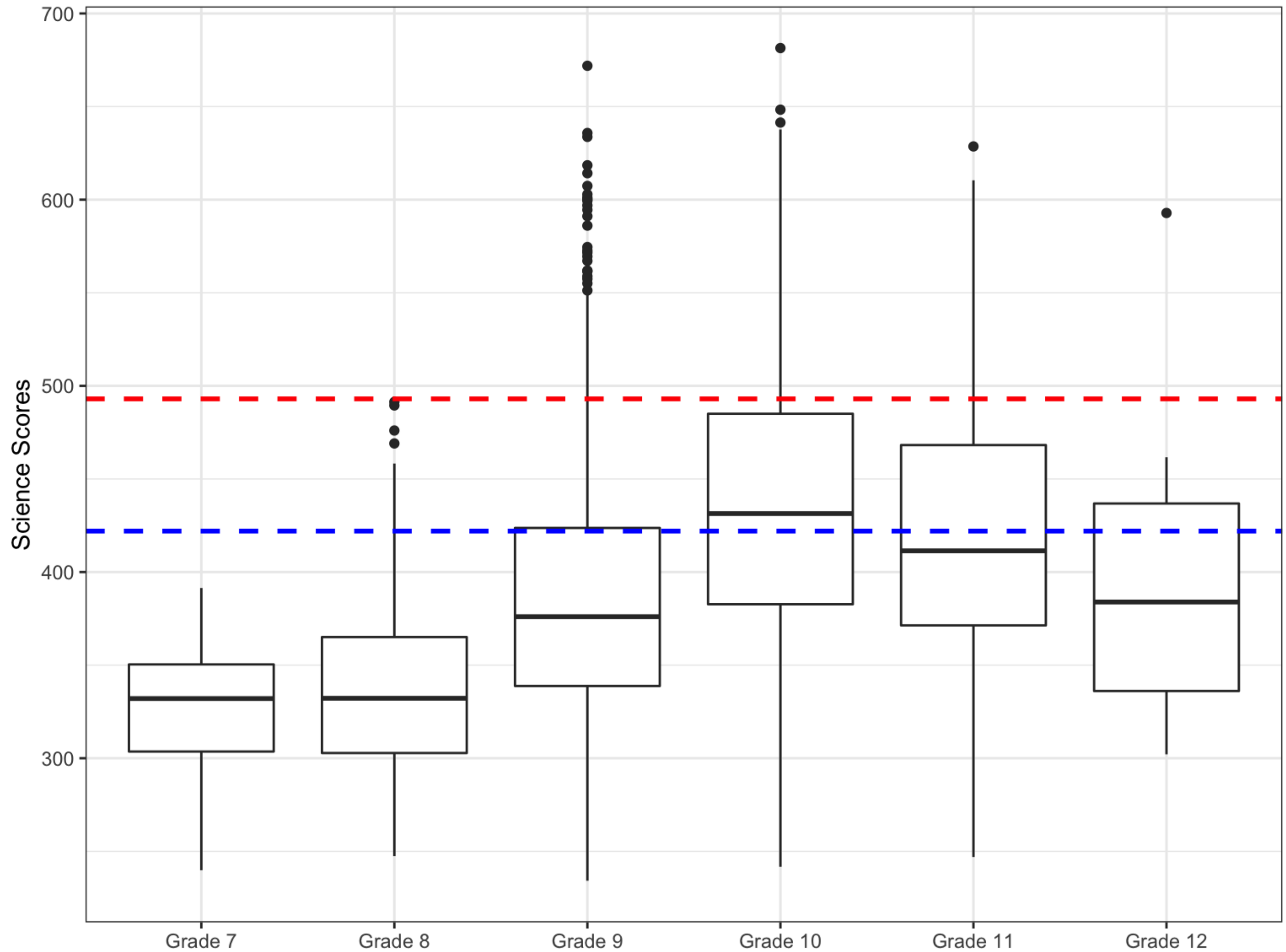
Now we can visualize some of the variables in the data. Let’s view science performance by grade.

```
ggplot(data = pisa,
       mapping = aes(x = grade, y = science)) +
  geom_boxplot() +
  labs(x=NULL, y="Science Scores") +
  theme_bw()
```



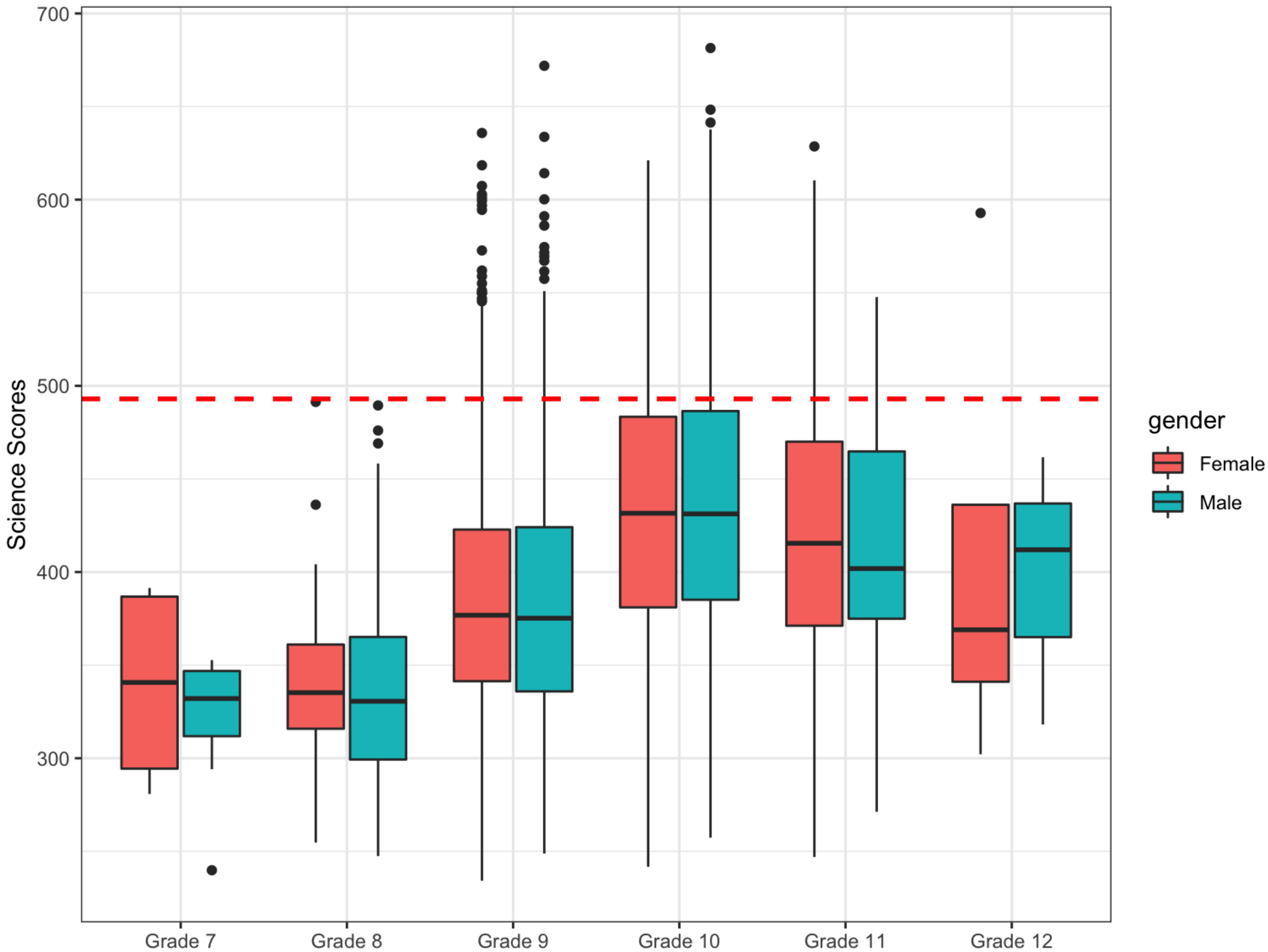
Let’s add horizontal line representing the OECD (red) and Turkey (blue) averages.

```
ggplot(data = pisa,
       mapping = aes(x = grade, y = science)) +
  geom_boxplot() +
  labs(x=NULL, y="Science Scores") +
  geom_hline(yintercept = 493, linetype="dashed", color = "red", size = 1) +
  geom_hline(yintercept = 422, linetype="dashed", color = "blue", size = 1) +
  theme_bw()
```



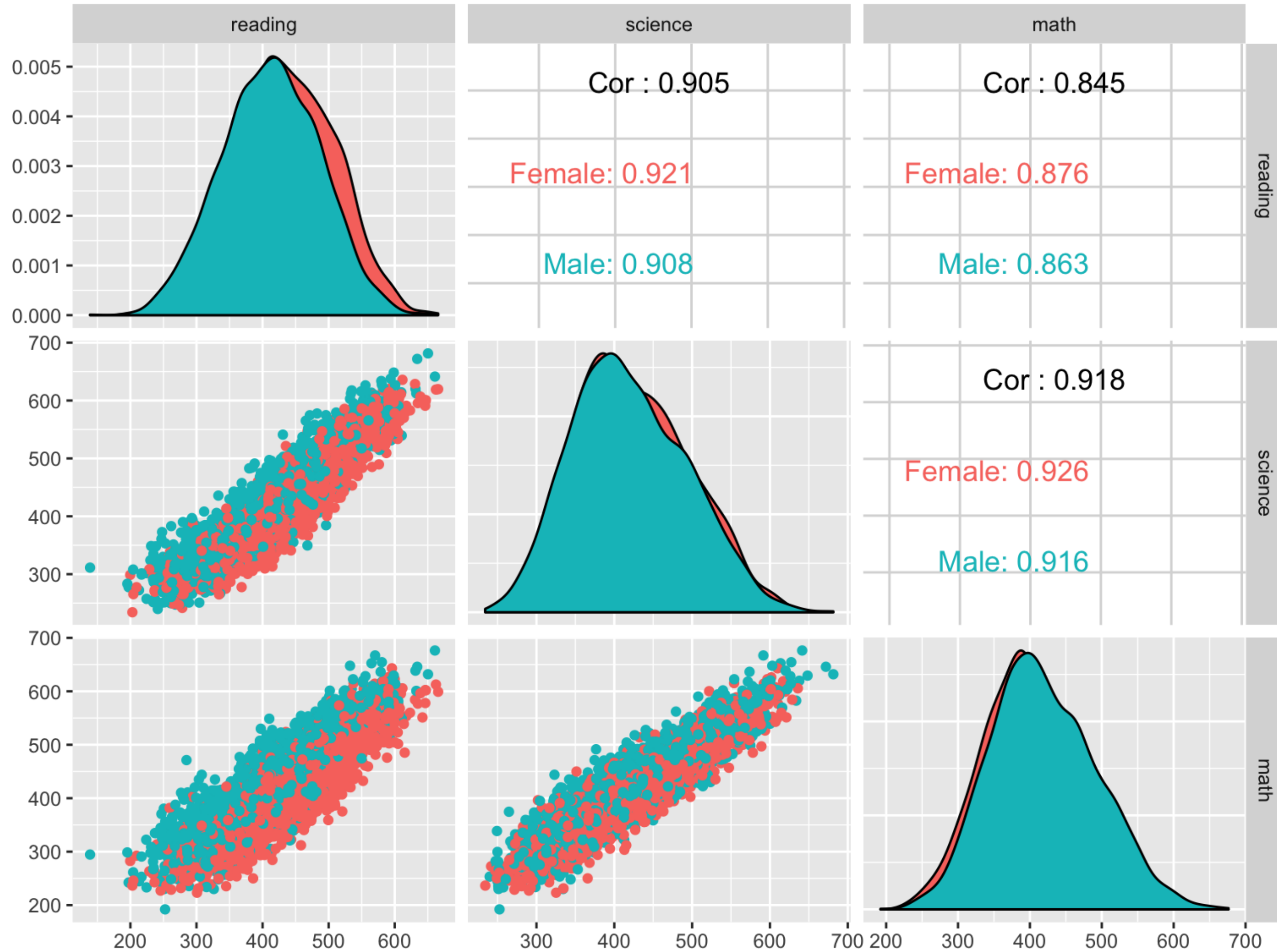
Add gender as a color layer using `fill = gender`:

```
ggplot(data = pisa,
       mapping = aes(x = grade, y = science, fill = gender)) +
  geom_boxplot() +
  labs(x=NULL, y="Science Scores") +
  geom_hline(yintercept = 493, linetype="dashed", color = "red", size = 1) +
  theme_bw()
```



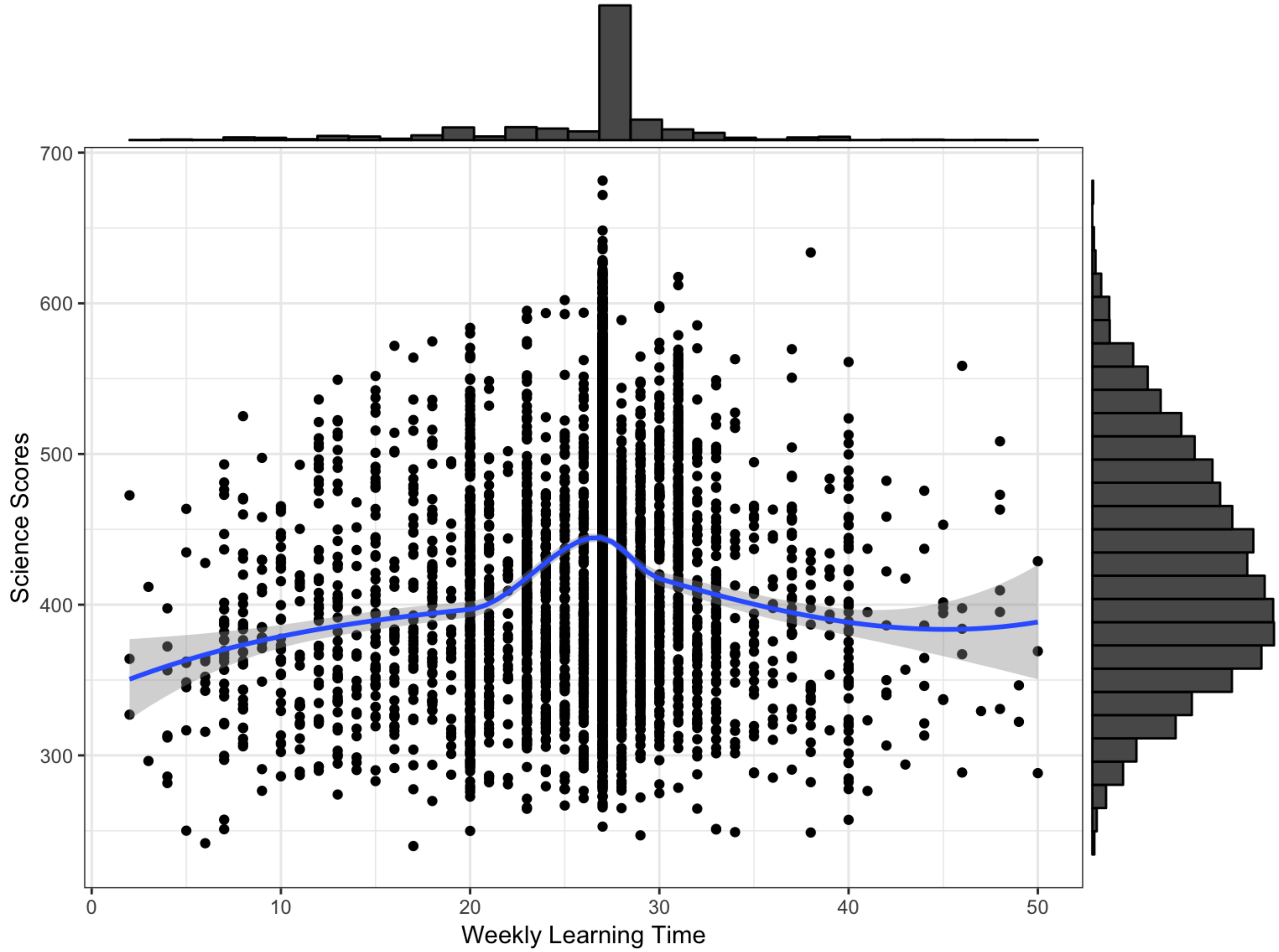
Let's see the impact of gender even further.

```
ggpairs(data = pisa,
        mapping = aes(color = gender),
        columns = c("reading", "science", "math"),
        upper = list(continuous = wrap("cor", size = 4.5))
        )
```



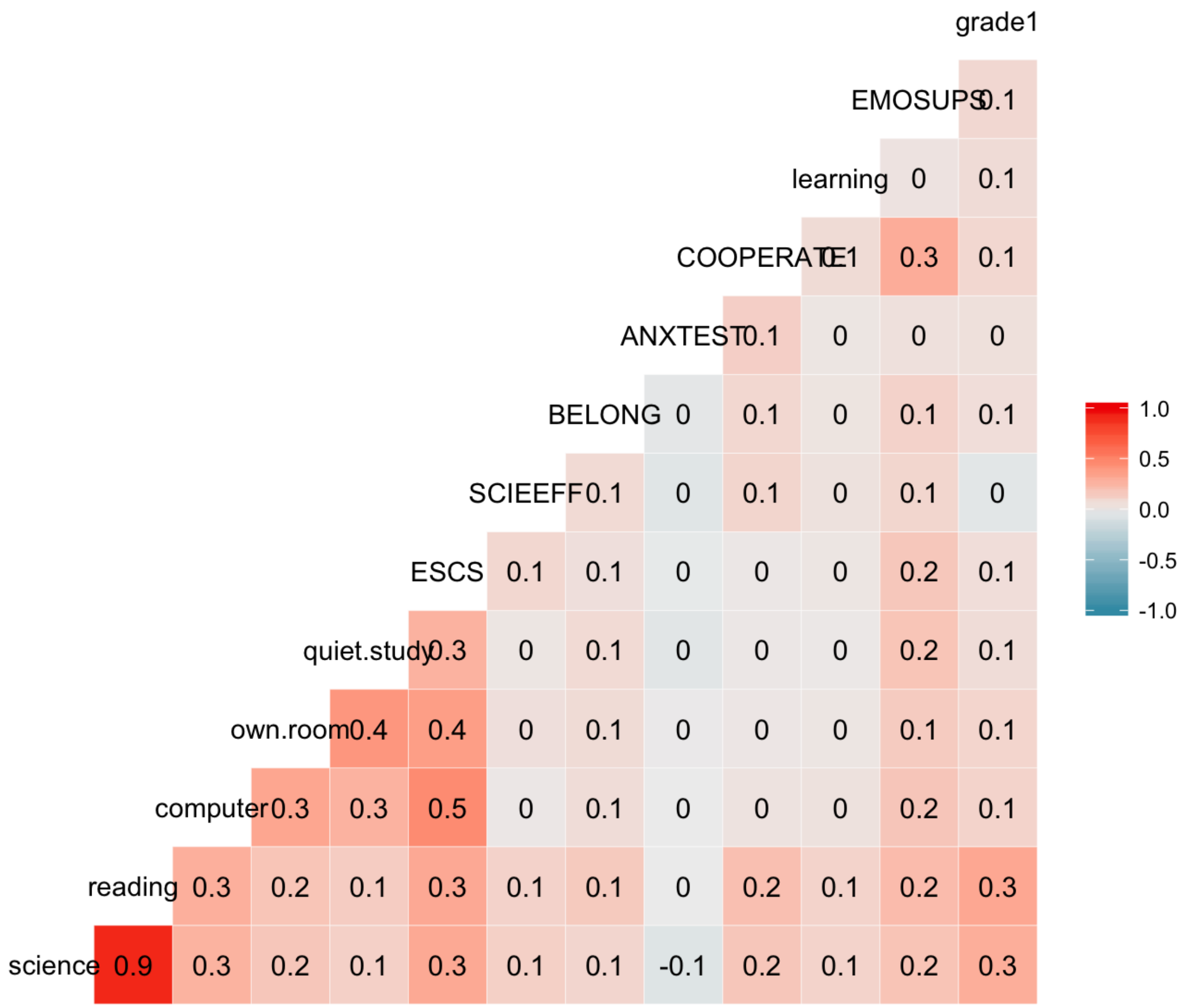
To examine conditional relationships:

```
p1 <- ggplot(data = pisa,
             mapping = aes(x = learning, y = science)) +
  geom_point() +
  geom_smooth(method = "loess") +
  labs(x = "Weekly Learning Time", y = "Science Scores") +
  theme_bw()

# Replace "histogram" with "boxplot" or "density" for other types
ggMarginal(p1, type = "histogram")
```



Most importantly, correlations among the variables should be checked:

```
ggcorr(data = pisa[,c("science", "reading", "computer", "own.room", "quiet.study",
                      "ESCS", "SCIEEFF", "BELONG", "ANXTEST", "COOPERATE", "learning",
                      "EMOSUPS", "grade1")],
       method = c("pairwise.complete.obs", "pearson"),
       label = TRUE, label_size = 4)
```



Decision Trees

We will split our dataset into a training dataset and a test dataset. We will train the decision tree on the training data and check its accuracy using the test data. In order to replicate the results later on, we need to set the seed – which will allow us to fix the randomization. Next, we remove the missing cases, save it as a new dataset, and then use createDataPartition() from the `caret` package to create an index to split the dataset as 70% to 30% using p = 0.7.

```
# Set the seed before splitting the data
set.seed(442019)

# We need to remove missing cases
pisa_nm <- na.omit(pisa)

# Split the data into training and test
index <- createDataPartition(pisa_nm$science_tr, p = 0.7, list = FALSE)
train <- pisa_nm[index, ]
test  <- pisa_nm[-index, ]

nrow(train)
```

```
[1] 2962
```

```
nrow(test)
```

```
[1] 1268
```

To build a decision tree model, we will use the `rpart` function from the `rpart` package. In the function, there are several elements:

- `formula = science_perf ~ ...` defines the dependent variable (i.e., science_perf) and the predictors (and `~` is the separator).
- `data = train` defines the dataset we are using for the analysis.
- `method = "class"` defines what type of decision tree we are building. `method = "class"` defines a classification tree and `method = "anova"` defines a regression tree.
- `control` is a list of control (i.e., tuning) elements for the decision tree algorithm. `minsplit` defines the minimum number of observations that must exist in a node (default = 20); `cp` is the complexity parameter to prune the subtrees that don't improve the model fit (default = 0.01, if `cp` = 0, then no pruning); `xval` is the number of cross-validations (default = 10, if `xval` = 0, then no cross validation).
- `parms` is a list of optional parameters for the splitting function. `anova` splitting (i.e., regression trees) has no parameters. For `class` splitting (i.e., classification tree), the most important option is the split index – which is either `"gini"` for the Gini index or `"information"` for the Entropy index. Splitting based on `information` can be slightly slower compared to the Gini index (see the vignette (https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf) for more information).

We will start building our decision tree model `df_fit1` (standing for decision tree fit for model 1) with no pruning (i.e., `cp = 0`) and no cross-validation as we have a test dataset already (i.e., `xval = 0`). We will use the Gini index for the splitting.

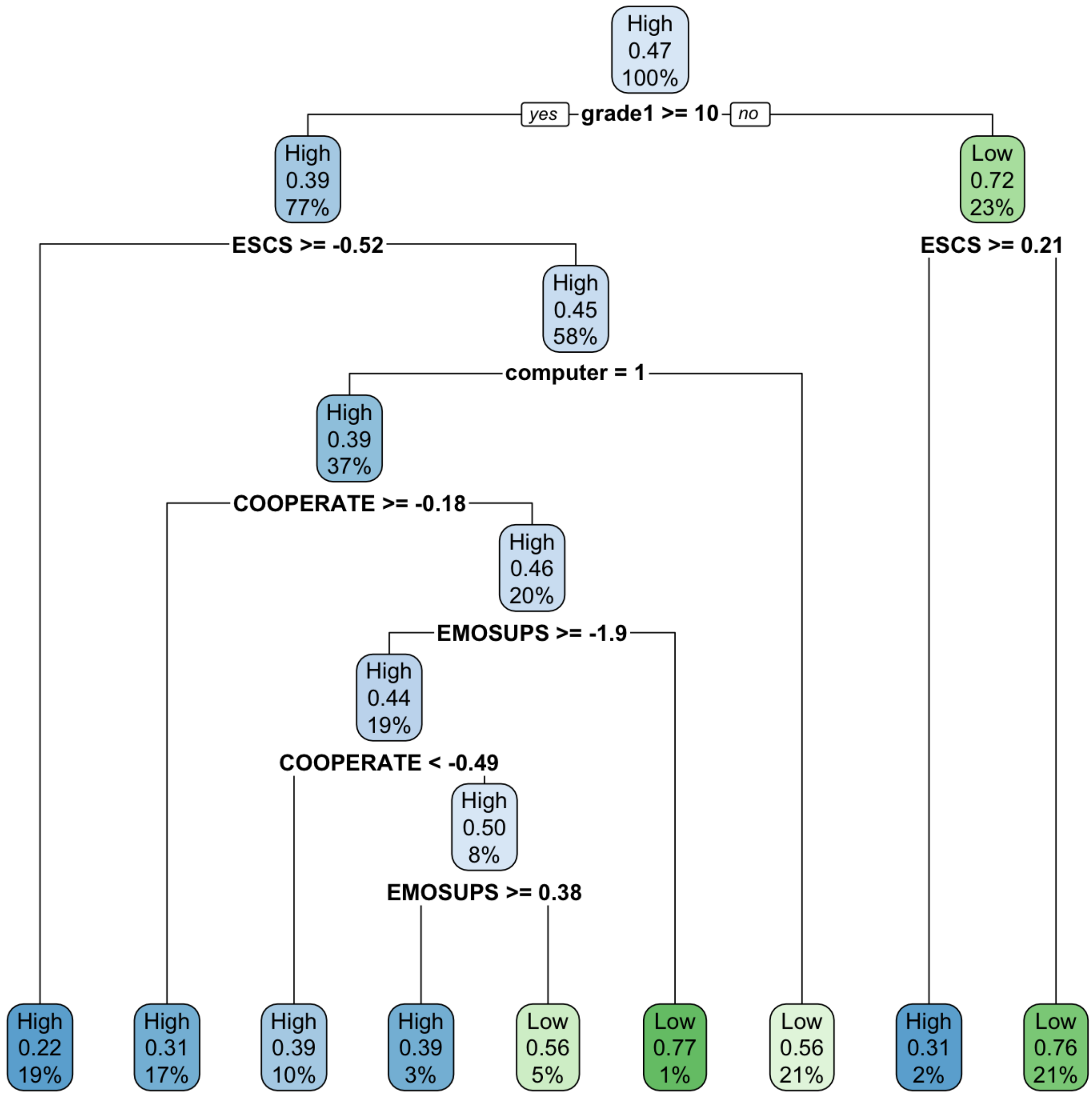
```
dt_fit1 <- rpart(formula = science_tr ~ grade1 + computer + own.room + ESCS +
                   EMOSUPS + COOPERATE,
                 data = train,
                 method = "class",
                 control = rpart.control(minsplit = 20,
                                         cp = 0,
                                         xval = 0),
                parms = list(split = "gini"))
```

To see the results:

```
summary(dt_fit1)
```

Note that the output will be way too long… We definitely need to prune the trees; otherwise the model yields a very complex model with many nodes – which is very likely to overfit the data. In the following model, we use `cp = 0.006`. Remember that as we increase `cp`, the pruning for the model will also increase. The higher the cp value, the shorter the trees with possibly fewer predictors.

```
dt_fit2 <- rpart(formula = science_tr ~ grade1 + computer + own.room + ESCS +
                   EMOSUPS + COOPERATE,
                 data = train,
                 method = "class",
                 control = rpart.control(minsplit = 20,
                                         cp = 0.006,
                                         xval = 0),
                parms = list(split = "gini"))

rpart.plot(dt_fit2)
```



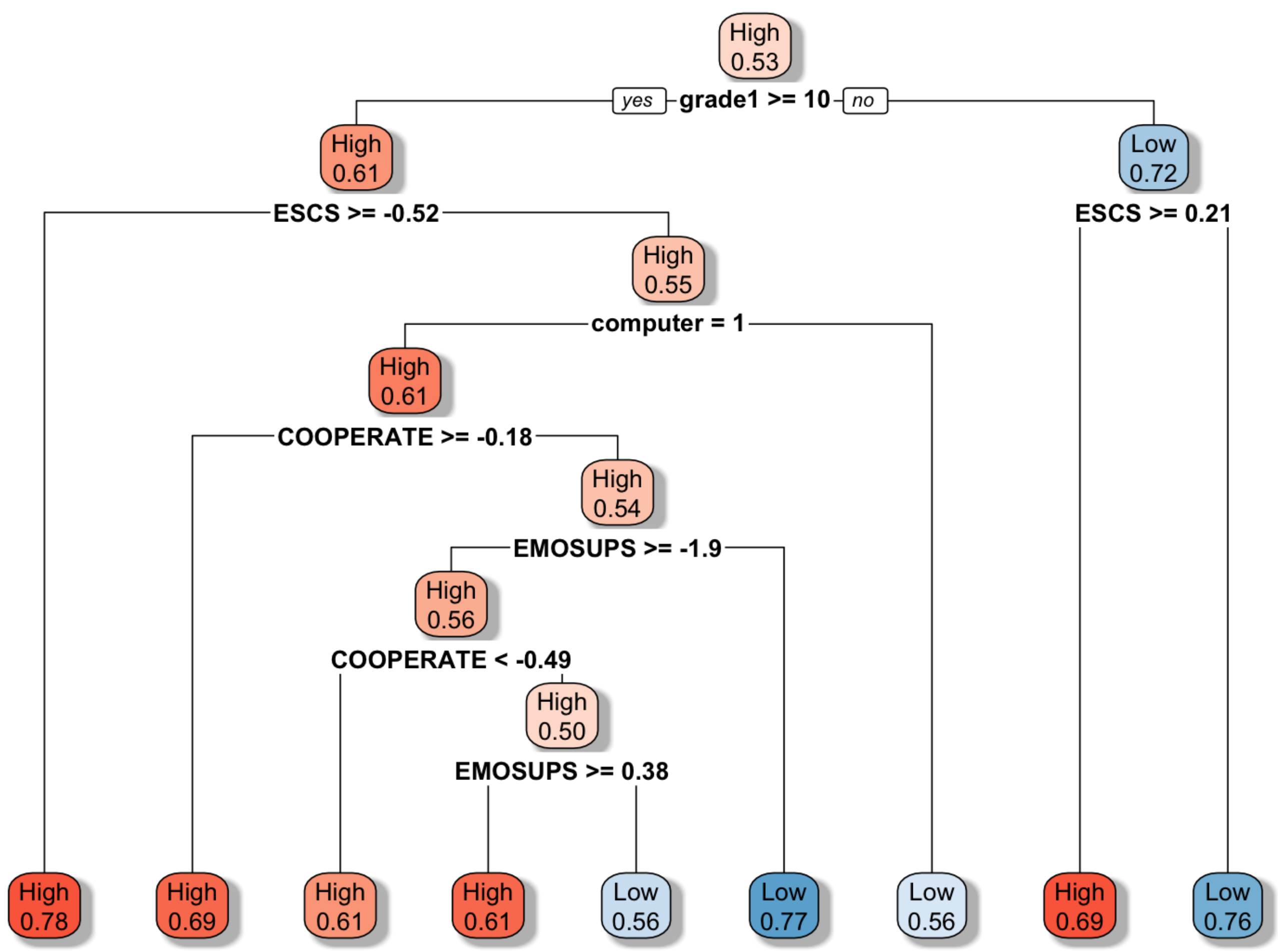
Now our model is less complex compared compared to the previous model. In the above decision tree plot, each node shows:

- the predicted class (High or low)
- the predicted probability of the second class (i.e., “Low”)
- the percentage of observations in the node

Let’s play with the colors to make the trees even more distinct. Also, we will adjust which values should be shown in the nodes, using `extra = 8` (see other possible options HERE (http://www.milbo.org/doc/prp.pdf)). Each node in the new plot shows:

- the predicted class (High or low)
- the predicted probability of the fitted class

```
rpart.plot(dt_fit2, extra = 8, box.palette = "RdBu", shadow.col = "gray")
```



Now let’s print the output of our model using `printcp()` :

```
printcp(dt_fit2)
```

```
Classification tree:
rpart(formula = science_tr ~ grade1 + computer + own.room + ESCS +
    EMOSUPS + COOPERATE, data = train, method = "class", parms = list(split = "gini"),
    control = rpart.control(minsplit = 20, cp = 0.006, xval = 0))

Variables actually used in tree construction:
[1] computer  COOPERATE EMOSUPS   ESCS      grade1

Root node error: 1384/2962 = 0.46725

n= 2962

         CP nsplit rel error
1 0.2124277      0   1.00000
2 0.0274566      1   0.78757
3 0.0166185      3   0.73266
4 0.0068642      4   0.71604
5 0.0060000      8   0.68786
```

In the output, `CP` refers to the complexity parameter, `nsplit` is the number of splits in the decision tree based on the complexity parameter, and `rel error` is the relative error (i.e., $1 - R^2$) of the solution. This is the error for predictions of the data that were used to estimate the model. The section of `Variables actually used in tree construction` shows which variables have been used in the final model.

In addition to `printcp()`, we can use `summary()` to print out more detailed results with all splits.

```
summary(dt_fit2)
```

1varImp() `from the` caret` package tells us which variables are more influential in the analysis:

```
varImp(dt_fit2)
```

```
            Overall
computer  161.73479
COOPERATE  90.74337
EMOSUPS    98.51003
ESCS      175.98497
grade1    114.83902
own.room   19.95518
```

The larger the values are, the more crucial they are for the model. In our example, computer and ESCS seem to be highly important for the decision tree model, whereas own.room is the least important variable.

Furthermore, we need to check the classification accuracy of the estimated decision tree with the **test** data. Otherwise, it is hard to justify whether the estimated decision tree would work accurately for prediction. Below we estimate the predicted classes (either high or low) from the test data by applying the estimated model. First we obtain model predictions using `predict()` and then turn the results into a data frame called `dt_pred`.

```
dt_pred <- predict(dt_fit2, test) %>%
  as.data.frame()

head(dt_pred)
```

```
        High       Low
1  0.2431118 0.7568882
3  0.2431118 0.7568882
4  0.2431118 0.7568882
5  0.2431118 0.7568882
7  0.2431118 0.7568882
13 0.4385113 0.5614887
```

This dataset shows each observation's (i.e., students from the test data) probability of falling into either *high* or *low* categories based on the decision rules that we estimated. We will turn these probabilities into binary classifications, depending on whether they are >= 50%. Then, we will compare these estimates with the actual classes in the test data (i.e., `test$science_tr`) in order to create a confusion matrix.

```
dt_pred <- mutate(dt_pred,
  science_tr = as.factor(ifelse(High >= 0.5, "High", "Low"))
  ) %>%
  select(science_tr)

confusionMatrix(dt_pred$science_tr, test$science_tr)
```

```
Confusion Matrix and Statistics

          Reference
Prediction High Low
      High  448 232
      Low   228 360

               Accuracy : 0.6372
                 95% CI : (0.6101, 0.6637)
    No Information Rate : 0.5331
    P-Value [Acc > NIR] : 0.00000000000004221

                  Kappa : 0.2709

 Mcnemar's Test P-Value : 0.8888

            Sensitivity : 0.6627
            Specificity : 0.6081
         Pos Pred Value : 0.6588
         Neg Pred Value : 0.6122
             Prevalence : 0.5331
         Detection Rate : 0.3533
   Detection Prevalence : 0.5363
      Balanced Accuracy : 0.6354

       'Positive' Class : High

```

The output shows that the overall accuracy is around 64%, sensitivity is 66%, and specificity is 61%. For only a few variable, this is not bad. However, adding more correlated variables can increase the accuracy and sensitivity of the model.

As you may remember, we set `xval = 0` in our decision tree models because we did not want to run any cross-validation samples. However, cross-validations (e.g., *K*-fold approach) are highly useful when we do not have a test or validation dataset, or our dataset is to small to split into training and test data. A typical way to use cross-validation in decision trees is to not specify a cp (i.e., complexity parameter) and perform cross validation. In the following example, we will assume that our dataset is not too big and thus we want to run 10 cross-validation samples (i.e., splits) as we build our decision tree model. Note that we use `cp = 0` this time.

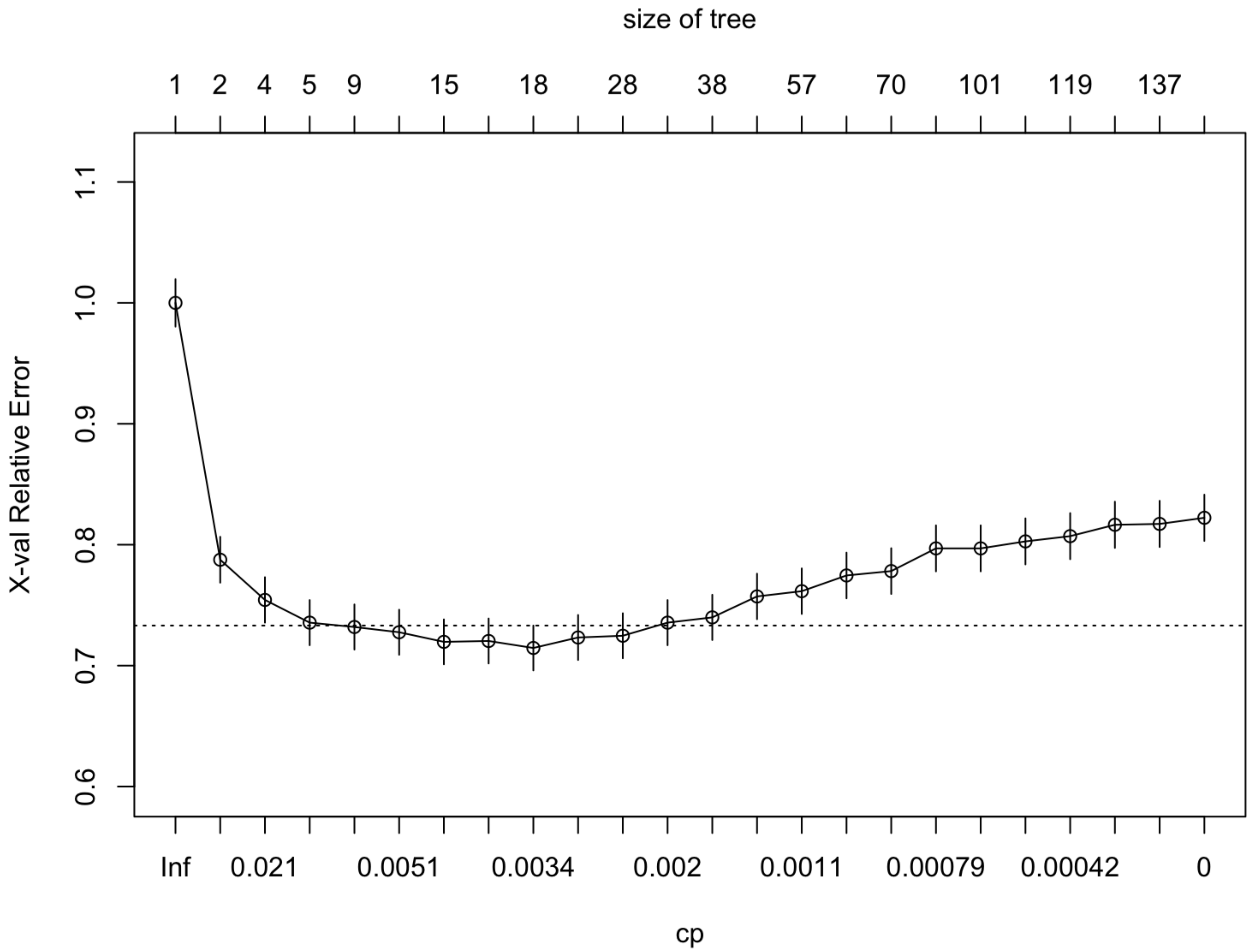
```
dt_fit3 <- rpart(formula = science_tr ~ grade1 + computer + own.room + ESCS +
                   EMOSUPS + COOPERATE,
                 data = train,
                 method = "class",
                 control = rpart.control(minsplit = 20,
                                         cp = 0,
                                         xval = 10),
                parms = list(split = "gini"))
```

In the results, we can evaluate the cross-validated error (i.e., X-val Relative Error) and choose the complexity parameter that would give us an acceptable value. Then, we can use this cp value and prune the trees. We use `plotcp()` function to visualize the cross-validation results.

```
printcp(dt_fit3)
```

```
Classification tree:
rpart(formula = science_tr ~ grade1 + computer + own.room + ESCS +
    EMOSUPS + COOPERATE, data = train, method = "class", parms = list(split = "gini"),
    control = rpart.control(minsplit = 20, cp = 0, xval = 10))

Variables actually used in tree construction:
[1] computer  COOPERATE EMOSUPS   ESCS      grade1    own.room

Root node error: 1384/2962 = 0.46725

n= 2962

           CP nsplit rel error  xerror     xstd
1  0.21242775      0   1.00000 1.00000 0.019620
2  0.02745665      1   0.78757 0.78757 0.018964
3  0.01661850      3   0.73266 0.75434 0.018787
4  0.00686416      4   0.71604 0.73555 0.018676
5  0.00523844      8   0.68786 0.73194 0.018654
6  0.00505780     13   0.66113 0.72760 0.018628
7  0.00433526     14   0.65607 0.71965 0.018578
8  0.00397399     15   0.65173 0.72038 0.018582
9  0.00289017     17   0.64379 0.71460 0.018545
10 0.00252890     21   0.63223 0.72327 0.018601
11 0.00216763     27   0.61272 0.72471 0.018610
12 0.00180636     33   0.59899 0.73555 0.018676
13 0.00144509     37   0.59032 0.73988 0.018702
14 0.00120424     44   0.58020 0.75723 0.018803
15 0.00108382     56   0.56214 0.76156 0.018827
16 0.00088311     58   0.55997 0.77457 0.018897
17 0.00086705     69   0.54986 0.77818 0.018916
18 0.00072254     85   0.53107 0.79697 0.019011
19 0.00065029    100   0.51951 0.79697 0.019011
20 0.00048170    112   0.51156 0.80275 0.019038
21 0.00036127    118   0.50867 0.80708 0.019059
22 0.00018064    132   0.50289 0.81647 0.019102
23 0.00014451    136   0.50217 0.81720 0.019105
24 0.00000000    141   0.50145 0.82225 0.019127
```

```
plotcp(dt_fit3)
```



Based on the results above, we can specify an ideal CP value (e.g., 0.0034) and re-run the model without cross-validation.

Random Forests

In R, `randomForest` and `caret` packages can be used to apply the random forest algorithm to classification and regression problems. The use of the `randomForest()` function is similar to that of `rpart()`. The main elements that we need to define are:

- **formula**: A regression-like formula defining the dependent variable and the predictors – it is the same as the one for `rpart()`.
- **data**: The dataset that we use to train the model.
- **importance**: If TRUE, then importance of the predictors is assessed in the model.
- **ntree**: Number of trees to grow in the model; we often start with a large number and then reduce it as we adjust the model based on the results. A large number for **ntree** can significantly increase the estimation time for the model.

There are also other elements that we can change depending on whether it is a classification or regression model (see `?randomForest` for more details). In the following example, we will focus on the same classification problem that we used before for decision trees. We initially set `ntree = 1000` to get 1000 trees in total but we will evaluate whether we need all of these trees to have an accurate model.

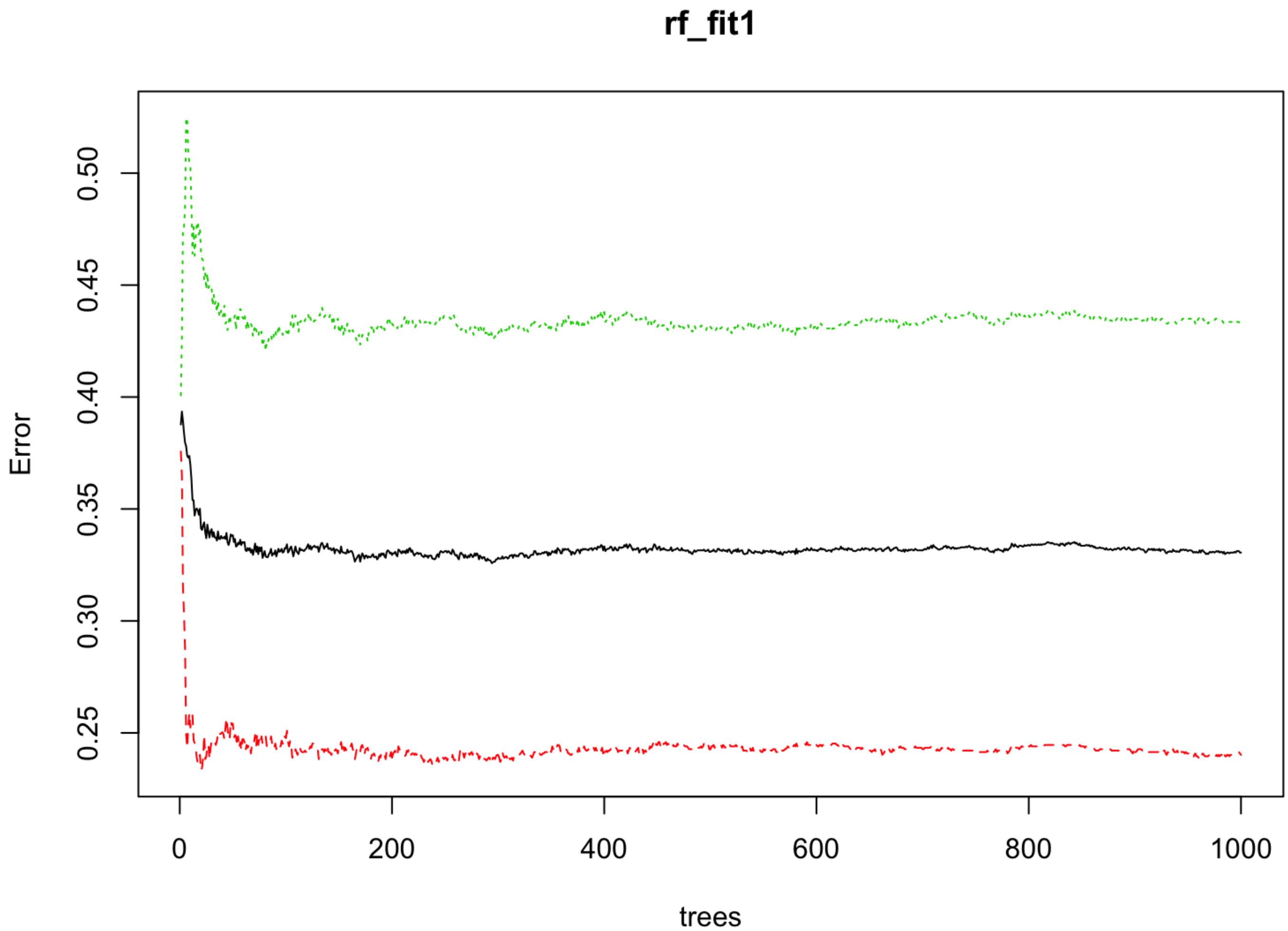
```
rf_fit1 <- randomForest(formula = science_tr ~ grade1 + computer + own.room + ESCS +
                          EMOSUPS + COOPERATE,
                        data = train,
                        importance = TRUE,
                        ntree = 1000)

print(rf_fit1)
```

```

Call:
 randomForest(formula = science_tr ~ grade1 + computer + own.room +      ESCS + EMOSUPS
+ COOPERATE, data = train, importance = TRUE,      ntree = 1000)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 2

        OOB estimate of  error rate: 33.05%
Confusion matrix:
     High Low class.error
High 1199 379   0.2401774
Low   600 784   0.4335260
```

In the output, we see the confusion matrix along with classification error and out-of-bag (OOB) error. OBB is a method of measuring the prediction error of random forests, finding the mean prediction error on each training sample, using only the trees that did not have in their bootstrap sample. The results show that the overall OBB error is around 33%, while the classification error is 24% for the *high* category and around 43% for the *low* category.

Next, by checking the level error across the number of trees, we can determine the ideal number of trees for our model.

```
plot(rf_fit1)
```



The plot shows that the error level does not go down any further after roughly 50 trees. So, we can run our model again by using `ntree = 50` this time.

```
rf_fit2 <- randomForest(formula = science_tr ~ grade1 + computer + own.room + ESCS +
                          EMOSUPS + COOPERATE,
                        data = train,
                        importance = TRUE,
                        ntree = 50)

print(rf_fit2)
```

```
Call:
 randomForest(formula = science_tr ~ grade1 + computer + own.room +      ESCS + EMOSUPS 
+ COOPERATE, data = train, importance = TRUE,      ntree = 50)
               Type of random forest: classification
                     Number of trees: 50
No. of variables tried at each split: 2

        OOB estimate of  error rate: 33.15%
Confusion matrix:
     High Low class.error
High 1203 375   0.2376426
Low   607 777   0.4385838
```

We can see the overall accuracy of model as follows:

```
sum(diag(rf_fit2$confusion)) / nrow(train)
```

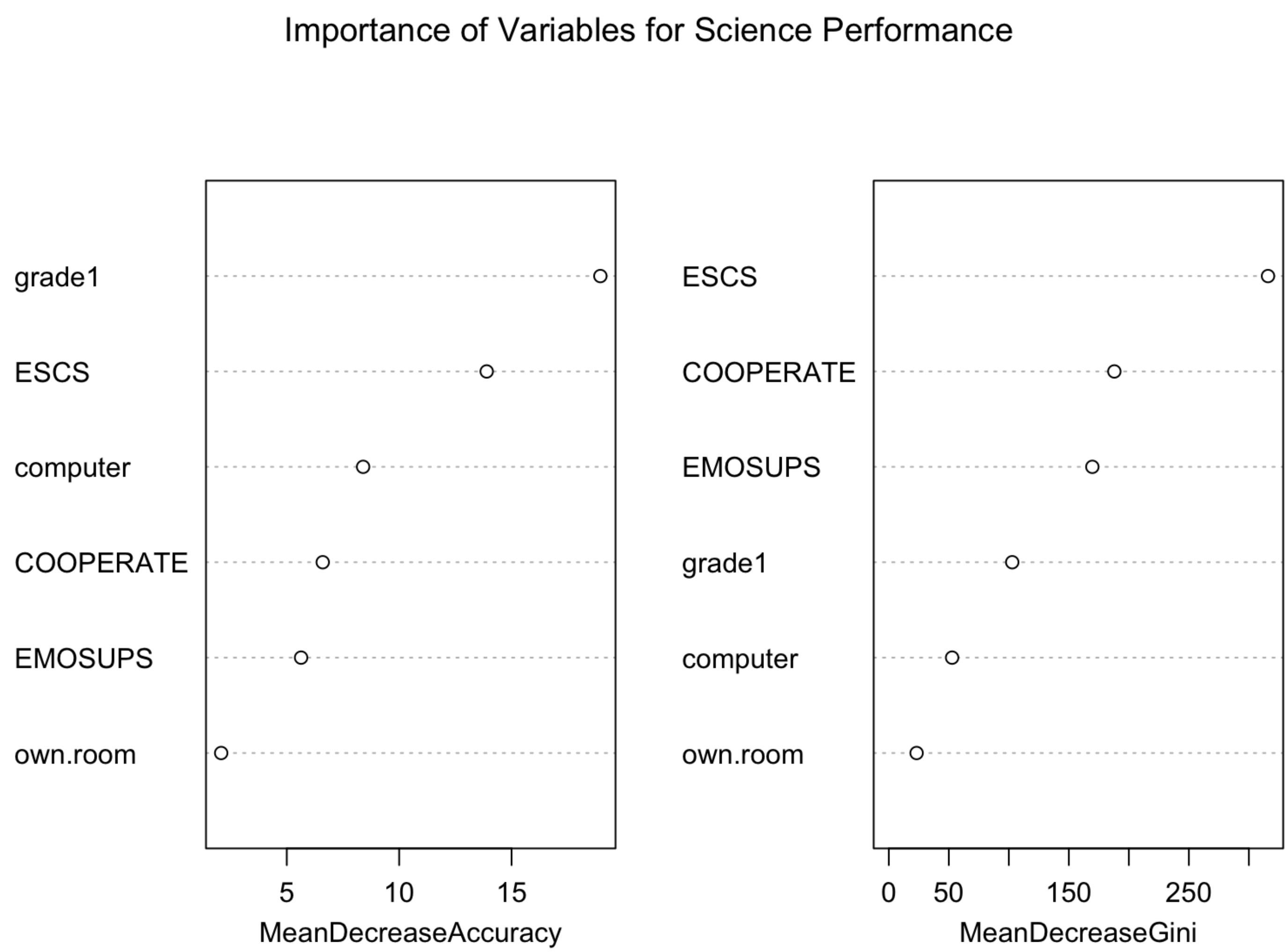
```
[1] 0.6684673
```

As we did for the decision trees, we can check the importance of the predictors in the model, using `importance()` and `varImpPlot()`. With `importance()`, we will first import the importance measures, turn it into a data.frame, save the row names as predictor names, and finally sort the data by MeanDecreaseGini (or, you can also see the basic output using only `importance(rf_fit2)`)

```
importance(rf_fit2) %>%
  as.data.frame() %>%
  mutate(Predictors = row.names(.)) %>%
  arrange(desc(MeanDecreaseGini))
```

```
        High       Low MeanDecreaseAccuracy MeanDecreaseGini Predictors
1 10.3285846  7.255164            13.893529        315.94495       ESCS
2  5.0398120  4.205330             6.599446        187.90433  COOPERATE
3  3.9979602  3.222740             5.636452        169.57163    EMOSUPS
4 17.2757145 14.219412            18.950736        102.81910     grade1
5  4.2833612  7.516286             8.396277         52.71843   computer
6  0.1854807  2.119964             2.081243         23.13221   own.room
```

```
varImpPlot(rf_fit2,
           main = "Importance of Variables for Science Performance")
```



The output shows different importance measures for the predictors that we used in the model. `MeanDecreaseAccuracy` and `MeanDecreaseGini` represent the overall classification error rate (or, mean squared error for regression) and the total decrease in node impurities from splitting on the variable, averaged over all trees. In the output, ESCS and grade are the two predictors that seem to influence the model performance substantially, whereas own.room and EMOSUPS are the least important variables. `varImpPlot()` presents the same information visually.

Next, we check the confusion matrix to see the accuracy, sensitivity, and specificity of our model.

```
rf_pred <- predict(rf_fit2, test) %>%
  as.data.frame() %>%
  mutate(science_tr = as.factor(`.`)) %>%
  select(science_tr)

confusionMatrix(rf_pred$science_tr, test$science_tr)
```

```
Confusion Matrix and Statistics

          Reference
Prediction High Low
      High  522 286
      Low   154 306

               Accuracy : 0.653
                 95% CI : (0.6261, 0.6792)
    No Information Rate : 0.5331
    P-Value [Acc > NIR] : < 0.00000000000000022

                  Kappa : 0.2931

 Mcnemar's Test P-Value : 0.0000000004233

            Sensitivity : 0.7722
            Specificity : 0.5169
         Pos Pred Value : 0.6460
         Neg Pred Value : 0.6652
             Prevalence : 0.5331
         Detection Rate : 0.4117
   Detection Prevalence : 0.6372
      Balanced Accuracy : 0.6445

       'Positive' Class : High

```

Better or worse than decision tree results???

Support Vector Machines

To do support vector classifiers (and SVMs) in R, we'll use the `e1071` package (though the `caret` package could be used, too). The `svm` function in the `e1071` package requires that the outcome variable is a factor – like the `science_tr` variable that we created earlier. If the outcome is not a factor, `svm` will perform regression.

To perform support vector classification, we use the `svm` function with the `kernel = "linear"` argument. We also need to specify our tolerance, which is represented by the `cost` argument. The `cost` parameter is essentially the inverse of the tolerance parameter, C. When the `cost` value is low, the tolerance is high (i.e., the margin is wide and there are lots of support vectors) and when the `cost` value is high, the tolerance is low (i.e., narrower margin). By default `cost = 1` and we will tune this parameter via cross-validation momentarily. For now, we'll just fit the model.

```
svc_fit <- svm(formula = science_tr ~ reading + math + grade1 + computer + own.room +
                 ESCS + EMOSUPS + COOPERATE,
               data = train,
               kernel = "linear")
```

We can obtain basic information about our model using the `summary` function.

```
summary(svc_fit)
```

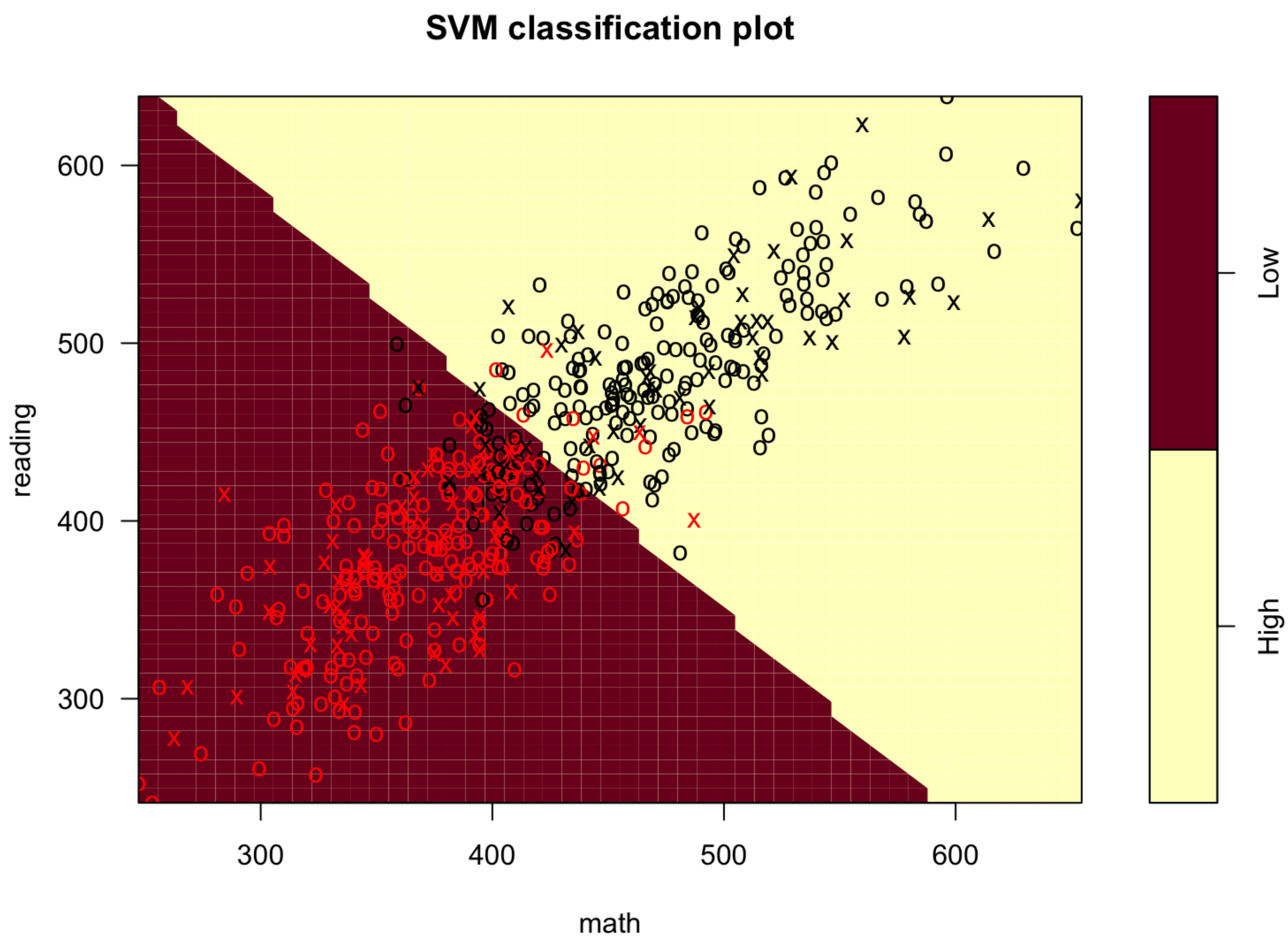
```


Call:
svm(formula = science_tr ~ reading + math + grade1 + computer +
    own.room + ESCS + EMOSUPS + COOPERATE, data = train, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  691

 ( 346 345 )


Number of Classes:  2

Levels:
 High Low
```

We see there are 691 support vectors: 346 in class “High” and 345 in class “Low”. We can also plot our model but we need to specify the two features we want to plot (because our model has six feature). Let’s look at the model with `reading` on the y-axis and `math` on the x-axis.

```
plot(svc_fit, data = train, reading ~ math)
```


In this figure, the Xs are the support vectors, while the Os are the non-support vector observations; the upper triangle are for “High”, while the lower triangle is for “Low”. While the decision boundary looks jagged, it’s just an artifact of the way it’s drawn with this function. We can see that many observations are misclassified (i.e., some red points are in the higher triangle and some black points are in the lower triangle). However, there are a lot of observations shown in this figure and it is difficult to discern the nature of the misclassification. Therefore, let’s take a random sample of 500 observations to get a better sense of our classifier.

```
set.seed(1)
ran_obs <- sample(1:nrow(train), 500)
plot(svc_fit, data = train[ran_obs, ], reading ~ math)
```



Initially when we fit the support vector classifier, we used the default cost parameter, but we really should select this parameter through tuning via cross-validation as we might be able to do an even better job at classifying. The `e1071` package includes a `tune` function which makes this easy and automatic. It performs the tuning via 10-folds cross-validation by default, which is probably a fine tradeoff. We need to provide the `tune` function with a range of cost values (which again corresponds to our tolerance to violate the margin and hyperplane).

```
tune_svc <- tune(svm,
                 science_tr ~ reading + math + grade1 + computer + own.room +
                 ESCS + EMOSUPS + COOPERATE,
                 data = train,
                 kernel="linear",
                 ranges = list(cost = c(.01, .1, 1, 5, 10)))
```

We can view the cross-validation errors by using the `summary` function on this object.

```
summary(tune_svc)
```

```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.0992538

- Detailed performance results:
   cost     error dispersion
1  0.01 0.0992538 0.01742894
2  0.10 0.1026231 0.01752322
3  1.00 0.1026242 0.01470387
4  5.00 0.1029621 0.01488161
5 10.00 0.1029621 0.01488161
```

And then select the best model and view it.

```
best_svc <- tune_svc$best.model
summary(best_svc)
```

```


Call:
best.tune(method = svm, train.x = science_tr ~ reading + math +
    grade1 + computer + own.room + ESCS + EMOSUPS + COOPERATE,
    data = train, ranges = list(cost = c(0.01, 0.1, 1, 5, 10)),
    kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  1016

 ( 510 506 )


Number of Classes:  2

Levels:
 High Low
```

Next, we write a function to evaluate our classifier that has one argument that takes a confusion matrix.

```r
#' Evaluate classifier
#'
#' Evaluates a classifier (e.g. SVM, logistic regression)
#' @param tab a confusion matrix
eval_classifier <- function(tab, print = F){
  n <- sum(tab)
  TN <- tab[2,2]
  FP <- tab[2,1]
  FN <- tab[1,2]
  TP <- tab[1,1]
  classify.rate <- (TP + TN) / n
  TP.rate <- TP / (TP + FN)
  TN.rate <- TN / (TN + FP)
  object <- data.frame(accuracy = classify.rate,
                       sensitivity = TP.rate,
                       specificity = TN.rate)
  return(object)
}
```

A confusion matrix for our `best_svc` can be created by:

```r
# to create a confusion matrix this order is important!
# observed values first and predict values second!
svc_train <- table(train$science_tr, predict(best_svc))
eval_classifier(svc_train)
```

```
   accuracy sensitivity specificity
1 0.9027684    0.900507   0.9053468
```

These statistics are likely overly optimistic as we are evaluating our model using the training data (the same data that we used to build our model). How well does the model perform on the testing data?

```r
svc_test <- table(test$science_tr, predict(best_svc, newdata = test))
eval_classifier(svc_test)
```

```
   accuracy sensitivity specificity
1 0.9290221   0.9289941   0.9290541
```

The statistics are impressively high! This is a very good classifier indeed. This is likely because `math` and `reading` are so highly correlated with `science` scores.

Comparison to logistic regression

Support vector classifiers are quite similar to logistic regression. This has to do with them having similar loss functions (the functions used to estimate the parameters). In situations where the classes are well separated, SVM (more generally), tend to do better than logistic regression and when they are not well separated, logistic regression tends to do better (James et al., 2013).

Let's compare logistic regression to the support vector classier. We'll begin by fitting the model

```
lr_fit <- glm(science_tr ~ reading + math + grade1 + computer + own.room + ESCS +
                EMOSUPS + COOPERATE,
              data = train,
              family = "binomial")
```

and then viewing the coefficients.

```
coef(lr_fit)
```

```
(Intercept)     reading        math      grade1    computer    own.room
34.92263354 -0.03862266 -0.04140528 -0.14934789  0.07882287  0.23195757
       ESCS     EMOSUPS   COOPERATE
-0.10357636 -0.00733228 -0.08985897
```

How does it do relative to our best support vector classifier on the training and the testing data sets? For the training data set:

```
lr_train <- table(train$science_tr,
                  round(predict(lr_fit, type = "response")))
lr_train
```

```


          0    1
  High 1421  157
  Low   133 1251
```

```
eval_classifier(lr_train)
```

```
   accuracy sensitivity specificity
1 0.9020932    0.900507   0.9039017
```

and then for the testing data set:

```
lr_test <- table(test$science_tr,
                 round(predict(lr_fit, newdata = test, type = "response")))
lr_test
```

```


         0   1
  High 625  51
  Low   46 546
```

```
eval_classifier(lr_test)
```

```
   accuracy sensitivity specificity
1 0.9235016   0.9245562   0.9222973
```

Equivalent out to the hundredths place. Either model would be fine here.