ECON 187: Project 2

Shannan Liu (305172952), Christina Zhang(605325840), Austin Pham (905318112), Zachary Wrubel (20

5/17/2022

Contents

Acquiring Data & Data Preprocessing

```
df = read_csv('https://raw.githubusercontent.com/onlypham/econ-187/main/proj2/grants.csv')
```

```
## New names:
## * `` -> `...46`
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## Rows: 58984 Columns: 46
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (37): Grantee Address, Grantee City, Grantee County Description, Grante...
## dbl   (5): Award Year, Financial Assistance, HHS Region Number, Geocoding Ar...
## lgl   (1): ...46
## date  (3): Project Period Start Date Text String, Grant Project Period End D...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# initially we have 46 variables
print(length(names(df)))
```

```
## [1] 46
```

```
names(df)
```

```
##  [1] "Award Year"
##  [2] "Financial Assistance"
##  [3] "Grantee Address"
##  [4] "Grantee City"
##  [5] "Grantee County Description"
##  [6] "Grantee County Name"
##  [7] "Grantee Name"
##  [8] "Grantee Region Code"
##  [9] "HRSA Region"
## [10] "Grantee State Abbreviation"
## [11] "State Name"
## [12] "Grantee ZIP Code"
## [13] "Grant Activity Code"
## [14] "Grant Number"
## [15] "Grant Serial Number"
## [16] "Project Period Start Date"
## [17] "Project Period Start Date Text String"
## [18] "Grant Project Period End Date"
## [19] "Grant Project Period End Date Text"
## [20] "HRSA Program Area Code"
## [21] "HRSA Program Area Name"
## [22] "Complete County Name"
## [23] "Grant Program Director E-mail"
```

```
## [24] "Grant Program Director Name"
## [25] "Grant Program Director Phone Number"
## [26] "Grant Program Name"
## [27] "Congressional District Name"
## [28] "Congressional District Number"
## [29] "HHS Region Number"
## [30] "U.S. Congressional Representative Name"
## [31] "State and County Federal Information Processing Standard Code"
## [32] "State FIPS Code"
## [33] "U.S. - Mexico Border 100 Kilometer Indicator"
## [34] "U.S. - Mexico Border County Indicator"
## [35] "Name of U.S. Senator Number One"
## [36] "Name of U.S. Senator Number Two"
## [37] "Data Warehouse Record Create Date"
## [38] "Data Warehouse Record Create Date Text"
## [39] "Uniform Data System Grant Program Description"
## [40] "Abstract"
## [41] "Grantee Type Description"
## [42] "DUNS Number"
## [43] "Unique Entity Identifier"
## [44] "Geocoding Artifact Address Primary X Coordinate"
## [45] "Geocoding Artifact Address Primary Y Coordinate"
## [46] "...46"
```

- Our target variable is `Financial Assistance`
- A lot of the variables are permutations of each other or not useful. After inspecting the data, we've identified that the following variables can be removed:
  - 'Grantee Address', Grantee County Name', 'Grantee City', 'Grant Number', Grantee Name', 'Grant Serial Number', 'Project Period Start Date', 'Project Period Start Date Text String', 'Grant Project Period End Date', 'Grantee State Abbreviation', 'Grant Project Period End Date Text', 'Complete County Name','Grant Program Director Name', 'Grant Program Director Phone Number', 'Congressional District Name', 'State and County Federal Information Processing Standard Code', 'Data Warehouse Record Create Date', 'Data Warehouse Record Create Date Text', 'Uniform Data System Grant Program Description', 'Abstract','DUNS Number','Name of U.S. Senator Number One', 'Name of U.S. Senator Number Two', 'HHS Region Number','U.S. Congressional Representative Name', 'Grantee ZIP Code','Unique Entity Identifier', 'State FIPS Code', 'Grant Program Director E-mail', 'Grant Activity Code', 'Grant Program Name'

```
# Drop the variables listed above
drops <- c('Grantee Address', 'Grantee County Name',
'Grantee City', 'Grant Number', 'Grantee Name',
'Grant Serial Number', 'Project Period Start Date',
'Project Period Start Date Text String',
'Grant Project Period End Date', 'Grantee State Abbreviation',
'Grant Project Period End Date Text',
'Complete County Name','Grant Program Director Name',
'Grant Program Director Phone Number', 'Congressional District Name',
'State and County Federal Information Processing Standard Code',
'Data Warehouse Record Create Date', 'Data Warehouse Record Create Date Text',
'Uniform Data System Grant Program Description', 'Abstract',
'DUNS Number','Name of U.S. Senator Number One',
'Name of U.S. Senator Number Two', 'HHS Region Number',
'U.S. Congressional Representative Name', 'Grantee ZIP Code',
```

```
'Unique Entity Identifier','State FIPS Code',
'Grant Program Director E-mail','Grant Activity Code',
'Grant Program Name')
df <- df[ , !(names(df) %in% drops)]

#  now we have 15 variables
print(length(names(df)))
```

```
## [1] 15
```

```
names(df)
```

```
##  [1] "Award Year"
##  [2] "Financial Assistance"
##  [3] "Grantee County Description"
##  [4] "Grantee Region Code"
##  [5] "HRSA Region"
##  [6] "State Name"
##  [7] "HRSA Program Area Code"
##  [8] "HRSA Program Area Name"
##  [9] "Congressional District Number"
## [10] "U.S. - Mexico Border 100 Kilometer Indicator"
## [11] "U.S. - Mexico Border County Indicator"
## [12] "Grantee Type Description"
## [13] "Geocoding Artifact Address Primary X Coordinate"
## [14] "Geocoding Artifact Address Primary Y Coordinate"
## [15] "...46"
```

Now, we can deal with any missing values in our target variable, if any.

```
# check length of our data
cat("Length of data:",length(df$`Financial Assistance`),'\n')
```

```
## Length of data: 58984
```

```
# check if our target variable has any missing values
cat("Missing values in our target var:",sum(is.na(df['Financial Assistance'])))
```

```
## Missing values in our target var: 0
```

Great, there are no missing values in our target. The next step is to split our data into a training and testing set. Then we'll deal with any general NA values in our dataset.

```
# great, now we can train-test-split
set.seed(42)
train_index = createDataPartition(df$`Financial Assistance`,
                                  p = .7, list = FALSE)
train <- df[train_index,]
test  <- df[-train_index,]
```

```r
# drop columns with high number of NA values
# define "high" as >20% null values
drops <- c()
# find the columns with a high number of NAs
for (col in names(train)){
  if (sum(is.na(df[col])) > 0.2*nrow(df[col])){
    drops <- c(drops,col)
  }
}
# drop those columns from our data
train <- train[ , !(names(train) %in% drops)]
test <- test[ , !(names(test) %in% drops)]

# find if there are any variables that don't
# give any information i.e. 0 variance
# this would be the case if a column only
# has 1 unique value
cat("In the training set these variables have 0 variance:",names(sapply(lapply(train, unique), length)[
```

```
## In the training set these variables have 0 variance:
```

None of our variables have 0 variance, so we can move on to processing our data. In other words, scaling our numeric data and encoding the categorical data.

We'll also impute any missing values as needed.

```r
# data preprocessing
# split into train-test sets
drops <- c('Financial Assistance')
X_train <-train[ , !(names(train) %in% drops)]
y_train <- train$`Financial Assistance`
X_test <- test[ , !(names(test) %in% drops)]
y_test <- test$`Financial Assistance`

# handling numeric data
# (1) impute > median
# (2) scale
X_train_scaled <- X_train %>%
  mutate_if(is.numeric,
            function(x) ifelse(is.na(x),
                               median(x,na.rm=T),x)) %>%
  mutate_if(is.numeric, function(x) scale(x))

X_test_scaled <- X_test %>%
  mutate_if(is.numeric,
            function(x) ifelse(is.na(x),
                               median(x,na.rm=T),x)) %>%
  mutate_if(is.numeric, function(x) scale(x))

# handling categorical data
# (1) impute with mode
X_train_scaled <- X_train_scaled %>%
  mutate_if(is.character,
```

```
            function(x) ifelse(is.na(x),
                               mode(x),x))
X_test_scaled <- X_test_scaled %>%
  mutate_if(is.character,
            function(x) ifelse(is.na(x),
                               mode(x),x))

# dummy vars
dummy <- dummyVars(" ~ .",
                   data = X_train_scaled)
X_train_scaled <- data.frame(predict(dummy,
                                     newdata = X_train_scaled))
X_test_scaled <- data.frame(predict(dummy,
                                    newdata = X_test_scaled))

# putting all the data back together for easier modelling
train_scaled <- X_train_scaled
train_scaled['Financial Assistance'] <- y_train

test_scaled <- X_test_scaled
test_scaled['Financial Assistance'] <- y_test

# check to see if the dummy variable creation process created
# any discrepancies in our training and testing data
cat(length(names(train_scaled)),
    length(names(test_scaled)))
```

```
## 182 181
```

There is an extra variable in our training set that isn't in our testing set.

```
cat("Missing Column(s):",names(train_scaled)[
  !names(train_scaled) %in% names(test_scaled)])
```

```
## Missing Column(s): X.Congressional.District.Number.42
```

Since it's a dummy variable column, we can just create a new zero-column in our test set.

```
test_scaled['X.Congressional.District.Number.42'] = 0

# check again just in case
cat("Missing Column(s):",names(train_scaled)[
  !names(train_scaled) %in% names(test_scaled)])
```

```
## Missing Column(s):
```

Great, our datasets match.

Now that our data is ready, we can begin fitting our models to predict the amount of financial assistance given for healthcare services in underserved populations of the U.S.

GAMs

```
library(splines)
library(gam)
# get non-categorical columns to make our predictions
n_num <- c()
for (col in names(train_scaled)){
  if (nrow(unique(train_scaled[col])) > 2) {
    n_num <- append(n_num,col)
  }
}

# get list of categorical predictor variables
n_cat <- names(train_scaled[,!(
  names(train_scaled) %in% n_num)])

# remove "Financial Assistance", our target,
# from this list of non-categorical
# predictor variables
n_num <- n_num[-length(n_num)]

# Create a GAM formula using natural splines
# on our non-categorical variables
form1 <- as.formula(
  paste("`Financial Assistance`~",
        paste(paste0("ns(",
                     paste0(n_num),
                     sep=", 4)"),
              collapse = " + ")))

# check what our formula looks like
form1
```

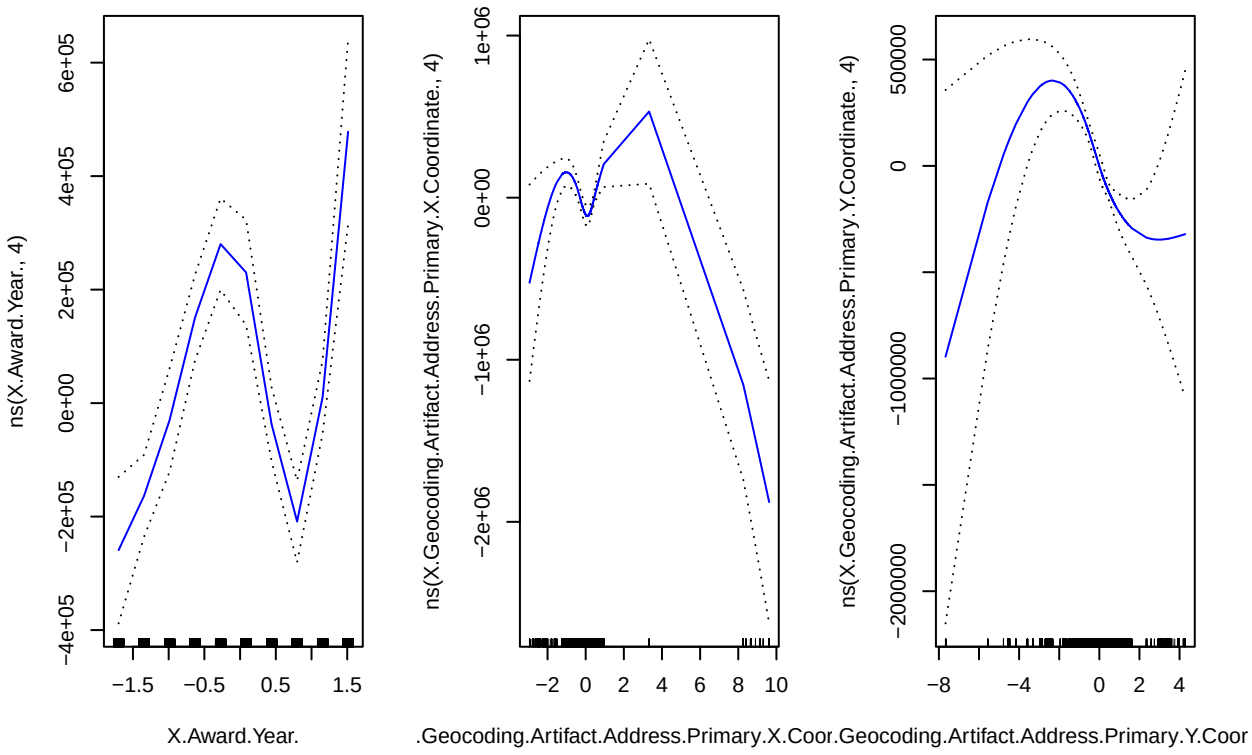
```
## `Financial Assistance` ~ ns(X.Award.Year., 4) + ns(X.Geocoding.Artifact.Address.Primary.X.Coordinate
##     4) + ns(X.Geocoding.Artifact.Address.Primary.Y.Coordinate.,
##     4)
```

From this, we can see that there are only 3 non-categorical variables in this dataset. We can use there 3 variables to create our first GAM model. Our second GAM model will then incorporate all the other categorical variables as well.

GAM 1

```
# build model
gam1 <- gam(form1, data = train_scaled)

# construct plots of X vs ns(X,4)
par(mfrow = c(1, 3))
plot(gam1, se = TRUE, col = "blue")
```



We observe that there are strong nonlinearities in our numeric variables being captured by the splines, which is great! Our model summary below affirms that all three variables are important. This is especially evidenced in the `Anova for Nonparametric Effects` section of the summary.

```
summary(gam1)
```

```
##
## Call: gam(formula = form1, data = train_scaled)
## Deviance Residuals:
##       Min        1Q    Median        3Q       Max
##  -2695853  -1374898  -1018817    -88084 149385173
##
## (Dispersion Parameter for gaussian family taken to be 1.987983e+13)
##
##     Null Deviance: 8.243949e+17 on 41290 degrees of freedom
## Residual Deviance: 8.205995e+17 on 41278 degrees of freedom
## AIC: 1381554
##
## Number of Local Scoring Iterations: 2
##
## Anova for Parametric Effects
##                                                             Df     Sum Sq
## ns(X.Award.Year., 4)                                         4 1.7129e+15
## ns(X.Geocoding.Artifact.Address.Primary.X.Coordinate., 4)    4 7.1184e+14
## ns(X.Geocoding.Artifact.Address.Primary.Y.Coordinate., 4)    4 1.3706e+15
## Residuals                                                41278 8.2060e+17
```

```
##                                                            Mean Sq F value
## ns(X.Award.Year., 4)                                     4.2822e+14 21.5406
## ns(X.Geocoding.Artifact.Address.Primary.X.Coordinate., 4) 1.7796e+14  8.9517
## ns(X.Geocoding.Artifact.Address.Primary.Y.Coordinate., 4) 3.4266e+14 17.2364
## Residuals                                                1.9880e+13
##                                                             Pr(>F)
## ns(X.Award.Year., 4)                                     < 2.2e-16 ***
## ns(X.Geocoding.Artifact.Address.Primary.X.Coordinate., 4) 3.193e-07 ***
## ns(X.Geocoding.Artifact.Address.Primary.Y.Coordinate., 4) 3.893e-14 ***
## Residuals
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now, let's build our second model

GAM 2

In this model, we're also going to utilise all of our categorical variables.

```
form2 <- as.formula(
  paste("`Financial Assistance`~",
        paste(paste(paste0("ns(",
                           paste0(n_num),
                           sep=", 4)"),
                    collapse = " + "),
              paste0(n_cat,
                     collapse="+"),
              sep = "+")))

gam2 <- gam(form2, data = train_scaled)

# check which model is better
anova(gam1, gam2, test = "F")
```

```
## Analysis of Deviance Table
##
## Model 1: `Financial Assistance` ~ ns(X.Award.Year., 4) + ns(X.Geocoding.Artifact.Address.Primary.X.C
##     4) + ns(X.Geocoding.Artifact.Address.Primary.Y.Coordinate.,
##     4)
## Model 2: `Financial Assistance` ~ ns(X.Award.Year., 4) + ns(X.Geocoding.Artifact.Address.Primary.X.C
##     4) + ns(X.Geocoding.Artifact.Address.Primary.Y.Coordinate.,
##     4) + X.Grantee.County.Description.Borough + X.Grantee.County.Description.Census.Area +
##     X.Grantee.County.Description.City + X.Grantee.County.Description.County +
##     X.Grantee.County.Description.District + X.Grantee.County.Description.Island +
##     X.Grantee.County.Description.Municipality + X.Grantee.County.Description.Municipio +
##     X.Grantee.County.Description.Not.Determined + X.Grantee.County.Description.Parish +
##     X.Grantee.County.Description.State + X.Grantee.Region.Code.01 +
##     X.Grantee.Region.Code.02 + X.Grantee.Region.Code.03 + X.Grantee.Region.Code.04 +
##     X.Grantee.Region.Code.05 + X.Grantee.Region.Code.06 + X.Grantee.Region.Code.07 +
##     X.Grantee.Region.Code.08 + X.Grantee.Region.Code.09 + X.Grantee.Region.Code.10 +
##     X.Grantee.Region.Code.character + X.HRSA.Region.Not.Determined +
##     X.HRSA.Region.Region.1 + X.HRSA.Region.Region.10 + X.HRSA.Region.Region.2 +
```

```
##     X.HRSA.Region.Region.3 + X.HRSA.Region.Region.4 + X.HRSA.Region.Region.5 +
##     X.HRSA.Region.Region.6 + X.HRSA.Region.Region.7 + X.HRSA.Region.Region.8 +
##     X.HRSA.Region.Region.9 + X.State.Name.Alabama + X.State.Name.Alaska +
##     X.State.Name.American.Samoa + X.State.Name.Arizona + X.State.Name.Arkansas +
##     X.State.Name.California + X.State.Name.Colorado + X.State.Name.Connecticut +
##     X.State.Name.Delaware + X.State.Name.District.of.Columbia +
##     X.State.Name.Federated.States.of.Micronesia + X.State.Name.Florida +
##     X.State.Name.Georgia + X.State.Name.Guam + X.State.Name.Hawaii +
##     X.State.Name.Idaho + X.State.Name.Illinois + X.State.Name.Indiana +
##     X.State.Name.Iowa + X.State.Name.Kansas + X.State.Name.Kentucky +
##     X.State.Name.Louisiana + X.State.Name.Maine + X.State.Name.Marshall.Islands +
##     X.State.Name.Maryland + X.State.Name.Massachusetts + X.State.Name.Michigan +
##     X.State.Name.Minnesota + X.State.Name.Mississippi + X.State.Name.Missouri +
##     X.State.Name.Montana + X.State.Name.Nebraska + X.State.Name.Nevada +
##     X.State.Name.New.Hampshire + X.State.Name.New.Jersey + X.State.Name.New.Mexico +
##     X.State.Name.New.York + X.State.Name.North.Carolina + X.State.Name.North.Dakota +
##     X.State.Name.Northern.Mariana.Islands + X.State.Name.Not.Determined +
##     X.State.Name.Ohio + X.State.Name.Oklahoma + X.State.Name.Oregon +
##     X.State.Name.Pennsylvania + X.State.Name.Puerto.Rico + X.State.Name.Republic.of.Palau +
##     X.State.Name.Rhode.Island + X.State.Name.South.Carolina +
##     X.State.Name.South.Dakota + X.State.Name.Tennessee + X.State.Name.Texas +
##     X.State.Name.U.S..Virgin.Islands + X.State.Name.Utah + X.State.Name.Vermont +
##     X.State.Name.Virginia + X.State.Name.Washington + X.State.Name.West.Virginia +
##     X.State.Name.Wisconsin + X.State.Name.Wyoming + X.HRSA.Program.Area.Code.BHW +
##     X.HRSA.Program.Area.Code.BPHC + X.HRSA.Program.Area.Code.FORHP +
##     X.HRSA.Program.Area.Code.HAB + X.HRSA.Program.Area.Code.HSB +
##     X.HRSA.Program.Area.Code.MCHB + X.HRSA.Program.Area.Code.OA +
##     X.HRSA.Program.Area.Name.Health.Workforce + X.HRSA.Program.Area.Name.Healthcare.Systems +
##     X.HRSA.Program.Area.Name.HIV.AIDS + X.HRSA.Program.Area.Name.Maternal.and.Child.Health +
##     X.HRSA.Program.Area.Name.Office.of.the.Administrator + X.HRSA.Program.Area.Name.Primary.Health.Ca
##     X.HRSA.Program.Area.Name.Rural.Health + X.HRSA.Program.Area.Name.Special.Initiatives.and.Other.Pr
##     X.Congressional.District.Number.00 + X.Congressional.District.Number.01 +
##     X.Congressional.District.Number.02 + X.Congressional.District.Number.03 +
##     X.Congressional.District.Number.04 + X.Congressional.District.Number.05 +
##     X.Congressional.District.Number.06 + X.Congressional.District.Number.07 +
##     X.Congressional.District.Number.08 + X.Congressional.District.Number.09 +
##     X.Congressional.District.Number.10 + X.Congressional.District.Number.11 +
##     X.Congressional.District.Number.12 + X.Congressional.District.Number.13 +
##     X.Congressional.District.Number.14 + X.Congressional.District.Number.15 +
##     X.Congressional.District.Number.16 + X.Congressional.District.Number.17 +
##     X.Congressional.District.Number.18 + X.Congressional.District.Number.19 +
##     X.Congressional.District.Number.20 + X.Congressional.District.Number.21 +
##     X.Congressional.District.Number.22 + X.Congressional.District.Number.23 +
##     X.Congressional.District.Number.24 + X.Congressional.District.Number.25 +
##     X.Congressional.District.Number.26 + X.Congressional.District.Number.27 +
##     X.Congressional.District.Number.28 + X.Congressional.District.Number.29 +
##     X.Congressional.District.Number.30 + X.Congressional.District.Number.31 +
##     X.Congressional.District.Number.32 + X.Congressional.District.Number.33 +
##     X.Congressional.District.Number.34 + X.Congressional.District.Number.35 +
##     X.Congressional.District.Number.36 + X.Congressional.District.Number.37 +
##     X.Congressional.District.Number.38 + X.Congressional.District.Number.39 +
##     X.Congressional.District.Number.40 + X.Congressional.District.Number.41 +
##     X.Congressional.District.Number.42 + X.Congressional.District.Number.43 +
##     X.Congressional.District.Number.44 + X.Congressional.District.Number.45 +
```

```
##     X.Congressional.District.Number.46 + X.Congressional.District.Number.47 +
##     X.Congressional.District.Number.48 + X.Congressional.District.Number.49 +
##     X.Congressional.District.Number.50 + X.Congressional.District.Number.51 +
##     X.Congressional.District.Number.52 + X.Congressional.District.Number.53 +
##     X.Congressional.District.Number.98 + X.Congressional.District.Number.99 +
##     X.Congressional.District.Number.XX + X.U.S....Mexico.Border.100.Kilometer.Indicator.N +
##     X.U.S....Mexico.Border.100.Kilometer.Indicator.Y + X.U.S....Mexico.Border.County.Indicator.N +
##     X.U.S....Mexico.Border.County.Indicator.Y + X.Grantee.Type.Description..No.Data. +
##     X.Grantee.Type.Description.Corporate.Entity..Federal.Tax.Exempt +
##     X.Grantee.Type.Description.Corporate.Entity..Not.Federal.Tax.Exempt +
##     X.Grantee.Type.Description.Foreign.Government + X.Grantee.Type.Description.International.Organiza
##     X.Grantee.Type.Description.Other + X.Grantee.Type.Description.Partnership +
##     X.Grantee.Type.Description.Sole.Proprietorship + X.Grantee.Type.Description.U.S..Government.Enti
##   Resid. Df Resid. Dev  Df   Deviance      F    Pr(>F)
## 1     41278 8.2060e+17
## 2     41146 7.7364e+17 132 4.6962e+16 18.922 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

From the results of the Anova test above, we can see that a GAM including categorical variables is better than a GAM without the categorical data. We can test this information below by examining each model's fit vs residuals plot. We can also compare their prediction values on the test set to see if there are any statistically significant differences between each model's predictions.

Fit vs Residuals Plots

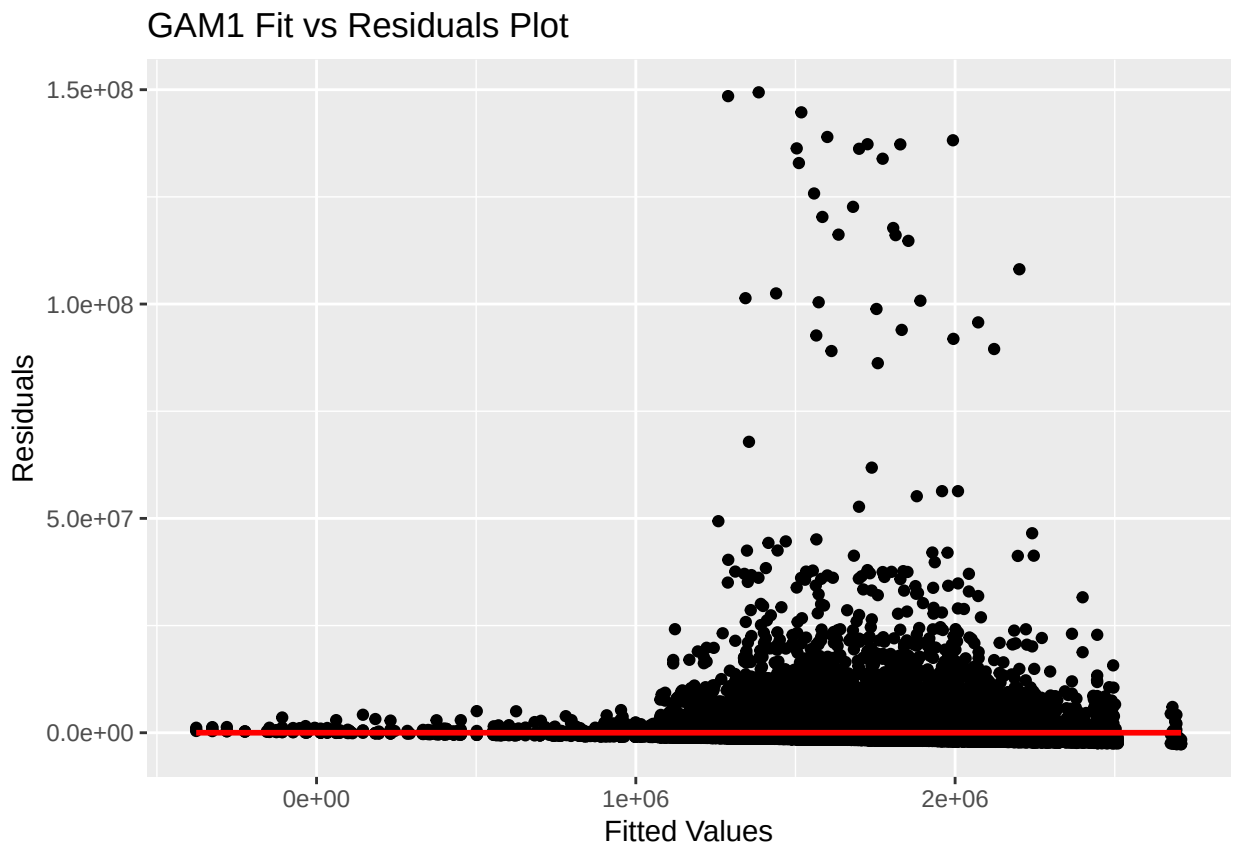
From the plots below, we can see that our second GAM model has better performance when fitting larger values while our first GAM model performs better when predicting smaller fitted values. Still, the second GAM model appears to fit better overall.

```
library(broom)
df <- augment(gam1)
```

```
## Warning: The 'augment()' method for objects of class Gam is not maintained by
## the broom team, and is only supported through the glm tidier method. Please be
## cautious in interpreting and reporting broom output.
```

```
p1 <- ggplot(df, aes(x = .fitted,
                     y = .resid)) +
  geom_point() +
  geom_smooth(method=lm ,
              color="red",
              fill="#0000FF", se=TRUE)
p1 + ggtitle("GAM1 Fit vs Residuals Plot") +
  xlab("Fitted Values") +
  ylab("Residuals")
```

```
## 'geom_smooth()' using formula 'y ~ x'
```

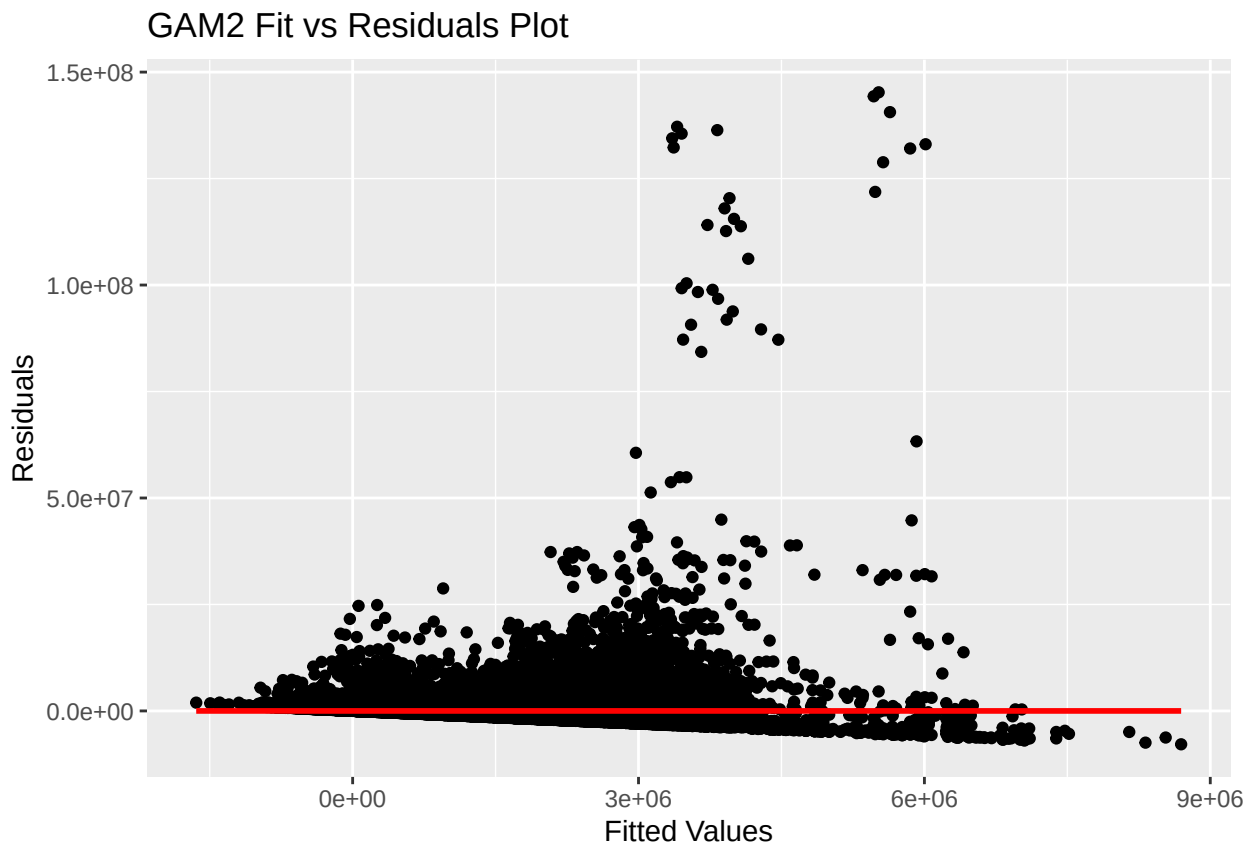



```
df <- augment(gam2)
```

```
## Warning: The 'augment()' method for objects of class Gam is not maintained by
## the broom team, and is only supported through the glm tidier method. Please be
## cautious in interpreting and reporting broom output.
```

```
p1 <- ggplot(df, aes(x = .fitted,
                     y = .resid)) +
  geom_point() +
  geom_smooth(method=lm ,
              color="red",
              fill="#0000FF", se=TRUE)
p1 + ggtitle("GAM2 Fit vs Residuals Plot") +
  xlab("Fitted Values") +
  ylab("Residuals")
```

```
## 'geom_smooth()' using formula 'y ~ x'
```



Model Predictions & Performance

Now we can move onto evaluating the performance of our models using metrics such as RMSE and MAE.

```
preds1 <- predict(gam1,
                  newdata = test_scaled)
preds2 <- predict(gam2,
                  newdata = test_scaled)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
# RMSE
cat("RMSE for GAM1:",sqrt(
  mean((preds1 - y_test)^2)),
  "\n")
```

```
## RMSE for GAM1: 4483210
```

```
cat("RMSE for GAM2:",sqrt(
  mean((preds2 - y_test)^2)),
  "\n")
```

```
## RMSE for GAM2: 4364093
```

```r
# MAE
cat("MAE for GAM1:",mean(
  abs(preds1 - y_test)),
  "\n")
```

```
## MAE for GAM1: 1744391
```

```r
cat("MAE for GAM2:",mean(
  abs(preds2 - y_test)))
```

```
## MAE for GAM2: 1644067
```

We can see that our second GAM model that includes all the categorical variables performs the best on our dataset. Thus, this is the best model, and the one we will be comparing with all of our other models.

Decision Tree

Based on your fits, identify the best model taking into consideration the bias-variance tradeoff. Make sure to discuss your results (including plots and tables), and to use CV and/or bootstrap to evaluate your models' performance.

```r
wine <- read_csv('https://raw.githubusercontent.com/onlypham/econ-187/main/proj2/wine.csv')
```

```
## Rows: 1599 Columns: 12
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (12): fixed acidity, volatile acidity, citric acid, residual sugar, chlo...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
colnames(wine) <- c("fixAcid", "volAcid", "citAcid", "resSugar", "chlorides", "freeSul", "totSul", "den
```

Regression Tree

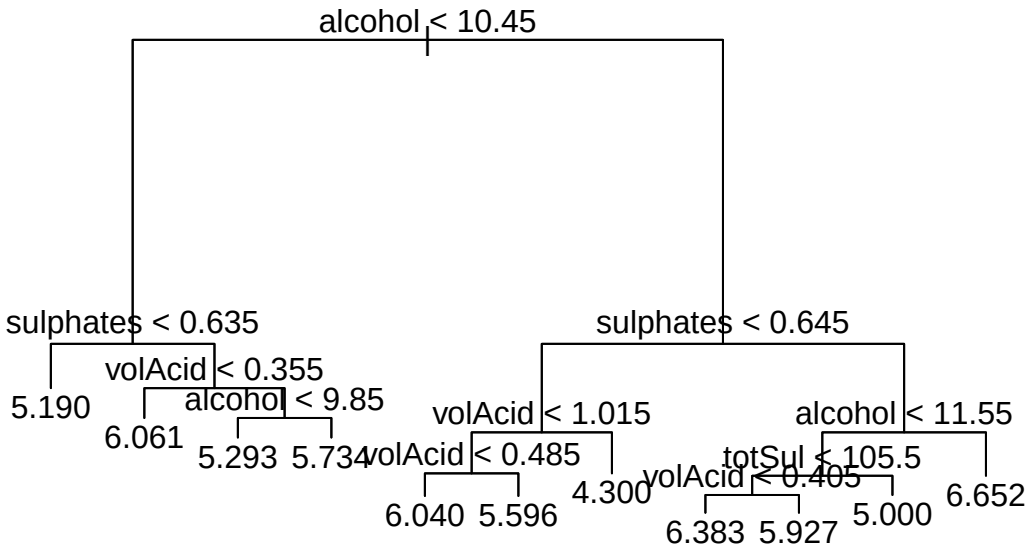
Let's first create a training set and fit a tree. Looking at the summary we notice that only three of the 3 out of the 11 variables are used in constructing the tree: alcohol, sulphates, and volAcid. This also seems to be the relative order of importance as we first split based on alcohol content and then sulphate content.

```r
set.seed(1)
train <- sample(1:nrow(wine), nrow(wine)*0.8)
tree.wine <- tree(quality ~ ., wine, subset = train)
summary(tree.wine)
```

```
##
## Regression tree:
## tree(formula = quality ~ ., data = wine, subset = train)
## Variables actually used in tree construction:
```

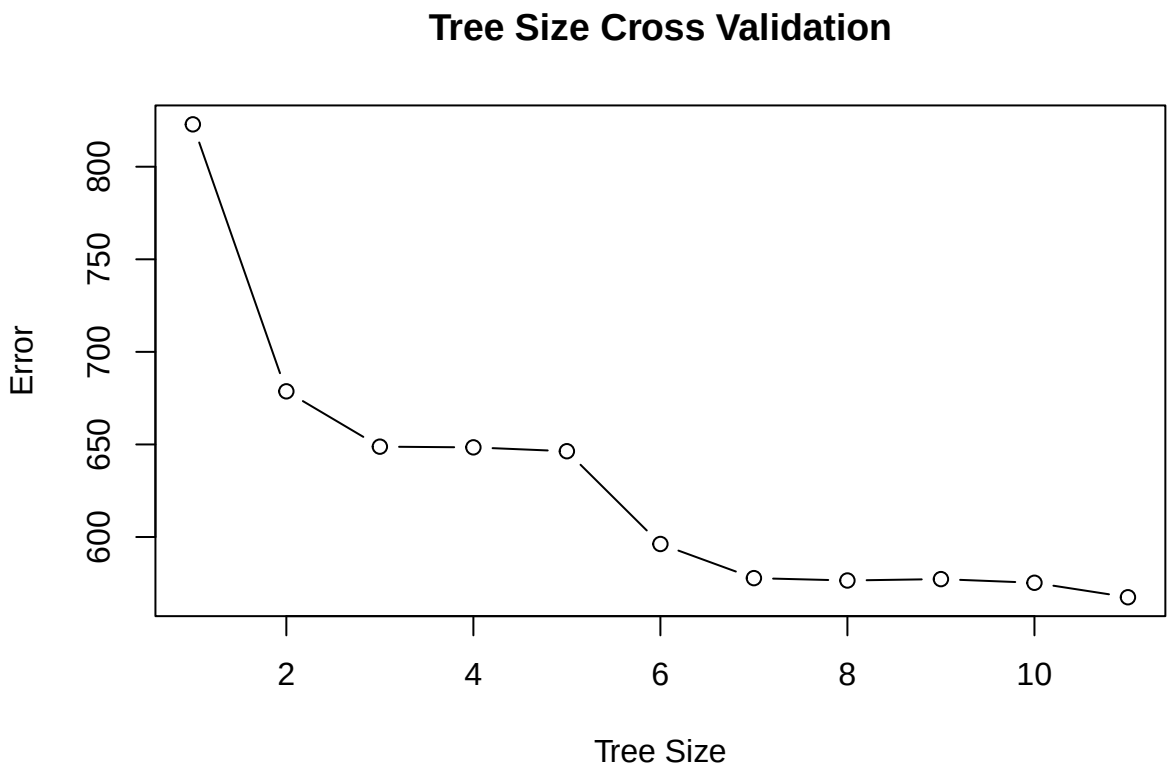
```
## [1] "alcohol"   "sulphates" "volAcid"   "totSul"
## Number of terminal nodes:  11
## Residual mean deviance:  0.3927 = 497.9 / 1268
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.1900 -0.2933 -0.1903  0.0000  0.4043  1.9390
```

```
plot(tree.wine)
text(tree.wine, pretty = 0)
```



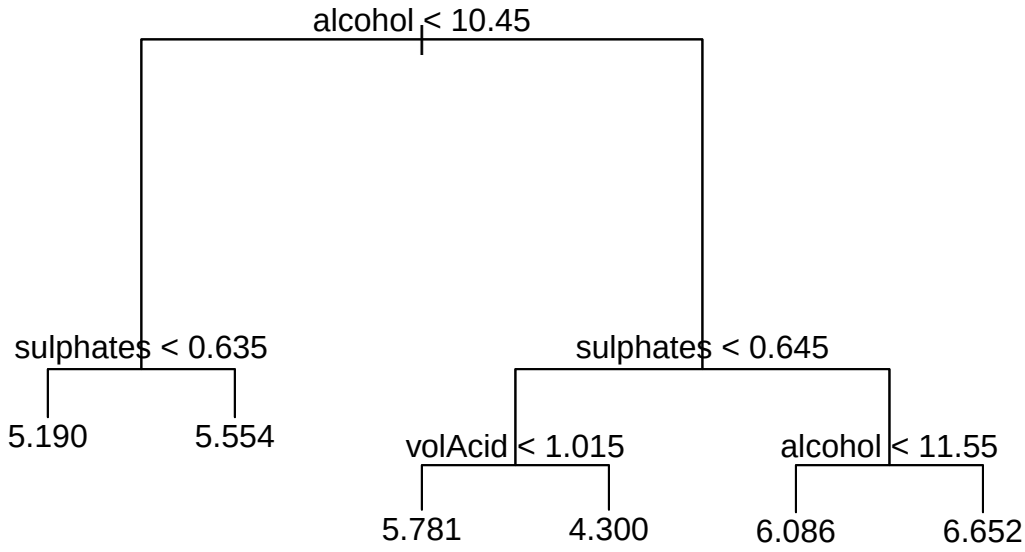
Using cross validation we wish to see if pruning the tree can improve performance.

```
cv.wine <- cv.tree(tree.wine)
plot(cv.wine$size, cv.wine$dev, type = "b", main = "Tree Size Cross Validation", xlab = "Tree Size", yl
```



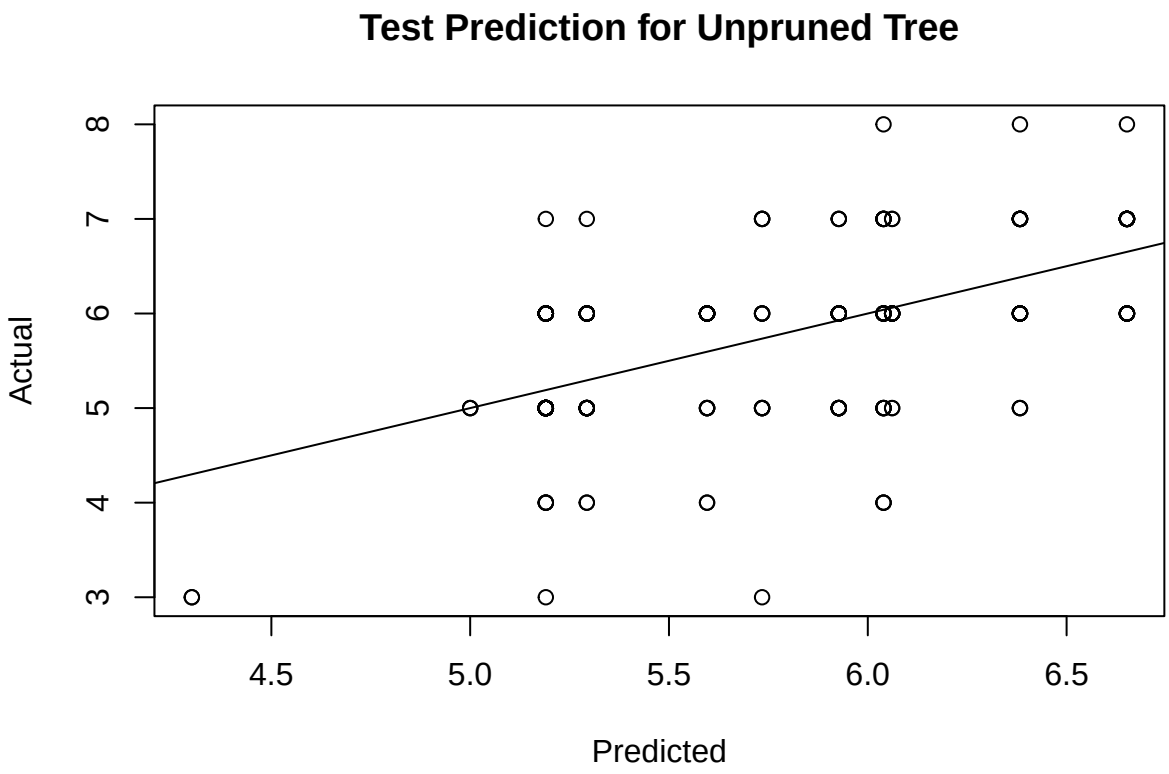
We notice that we get really good performance with even just a tree size of just 5-6. Let's just keep a size of 6 and see what the best pruned tree looks like.

```
prune.wine <- prune.tree(tree.wine, best = 6)
plot(prune.wine)
text(prune.wine, pretty = 0)
```



Using this unpruned tree, we get a MSE of 0.4801827. The RMSE is 0.6929522 which means that our model leads to test predictions that on average are within 0.6929522 of the true wine quality score.

```
yhat <- predict(tree.wine, newdata = wine[-train, ])
wine.test <- wine[-train, ] %>% pull(quality)
plot(yhat, wine.test, xlab = "Predicted", ylab = "Actual", main = "Test Prediction for Unpruned Tree")
abline(0, 1)
```




```
mean((yhat - wine.test)^2)
```

```
## [1] 0.4801827
```

Now, let's use the rpart method! Let's first split up the data.

```
set.seed(42)
inTraining <- createDataPartition(wine$quality, p = 0.8, list = FALSE)
training <- wine[inTraining,]
testing  <- wine[-inTraining,]
```

Let's build a full tree and performa k-fold cross-validation to select the optimal cost complexity (cp). The anova method creates a regression tree. Printing the results, we see that our training set has 1281 observations and at the root node we have an $SSE = 834.47930$ and a predicted quality mean of 5.636222 (this is just the sample mean of our training data).

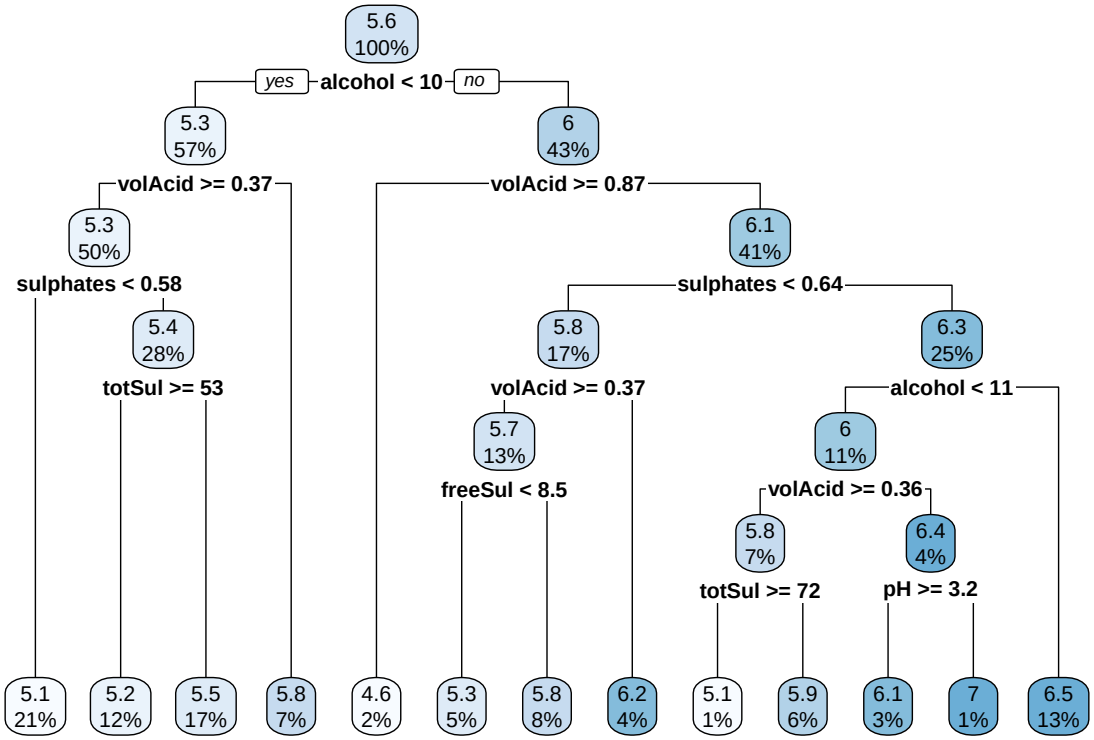
```
set.seed(42)
regression.tree  <- rpart(formula = quality ~ ., data = training, method = "anova", xval = 10, model = 
print(regression.tree)
```

```
## n= 1281
##
## node), split, n, deviance, yval
##       * denotes terminal node
```

```
##
##   1) root 1281 834.47930 5.636222
##     2) alcohol< 10.45 725 301.16690 5.342069
##       4) volAcid>=0.365 635 233.86770 5.272441
##         8) sulphates< 0.575 270  77.88519 5.107407 *
##         9) sulphates>=0.575 365 143.18900 5.394521
##          18) totSul>=52.5 152  35.26316 5.210526 *
##          19) totSul< 52.5 213  99.10798 5.525822 *
##       5) volAcid< 0.365 90  42.50000 5.833333 *
##     3) alcohol>=10.45 556 388.78240 6.019784
##       6) volAcid>=0.87 25  19.76000 4.640000 *
##       7) volAcid< 0.87 531 319.18640 6.084746
##        14) sulphates< 0.635 217 112.82030 5.783410
##          28) volAcid>=0.365 169  78.09467 5.656805
##            56) freeSul< 8.5 61  37.77049 5.344262 *
##            57) freeSul>=8.5 108  31.00000 5.833333 *
##          29) volAcid< 0.365 48  22.47917 6.229167 *
##        15) sulphates>=0.635 314 173.04460 6.292994
##          30) alcohol< 11.15 145  80.93793 6.020690
##            60) volAcid>=0.3625 93  38.51613 5.806452
##             120) totSul>=71.5 16   1.75000 5.125000 *
##             121) totSul< 71.5 77  27.79221 5.948052 *
##            61) volAcid< 0.3625 52  30.51923 6.403846
##             122) pH>=3.215 34  16.73529 6.088235 *
##             123) pH< 3.215 18   4.00000 7.000000 *
##          31) alcohol>=11.15 169  72.13018 6.526627 *
```

Using rpart() we grew the full tree using cross-validation to test the perofmrance of the possible comlexity hyperparameters.

```
rpart.plot(regression.tree, yesno = TRUE)
```



Let's use printcp to decide how to prune the tree. Looking at the ouptut we notice that an $nsplit = 0$ corresponds to just the root node in which we get a relative error of 1. xerror is the cross-validated SSE and xstd is the standard error.

```
printcp(regression.tree)
```

```
##
## Regression tree:
## rpart(formula = quality ~ ., data = training, method = "anova",
##     model = TRUE, xval = 10)
##
## Variables actually used in tree construction:
## [1] alcohol   freeSul   pH        sulphates totSul    volAcid
##
## Root node error: 834.48/1281 = 0.65143
##
## n= 1281
##
##          CP nsplit rel error  xerror     xstd
## 1  0.173198      0   1.00000 1.00198 0.042724
## 2  0.059721      1   0.82680 0.83797 0.041434
## 3  0.039931      2   0.76708 0.80405 0.037568
## 4  0.029718      3   0.72715 0.78519 0.037071
## 5  0.023939      4   0.69743 0.75920 0.035057
## 6  0.015331      5   0.67349 0.74824 0.034533
## 7  0.014676      6   0.65816 0.72359 0.033168
```

```
## 8  0.014263      7   0.64349 0.72406 0.033397
## 9  0.011725      8   0.62922 0.71017 0.033052
## 10 0.011174      9   0.61750 0.70283 0.032782
## 11 0.010754     10   0.60632 0.70205 0.032909
## 12 0.010567     11   0.59557 0.70205 0.032909
## 13 0.010000     12   0.58500 0.69947 0.032624
```

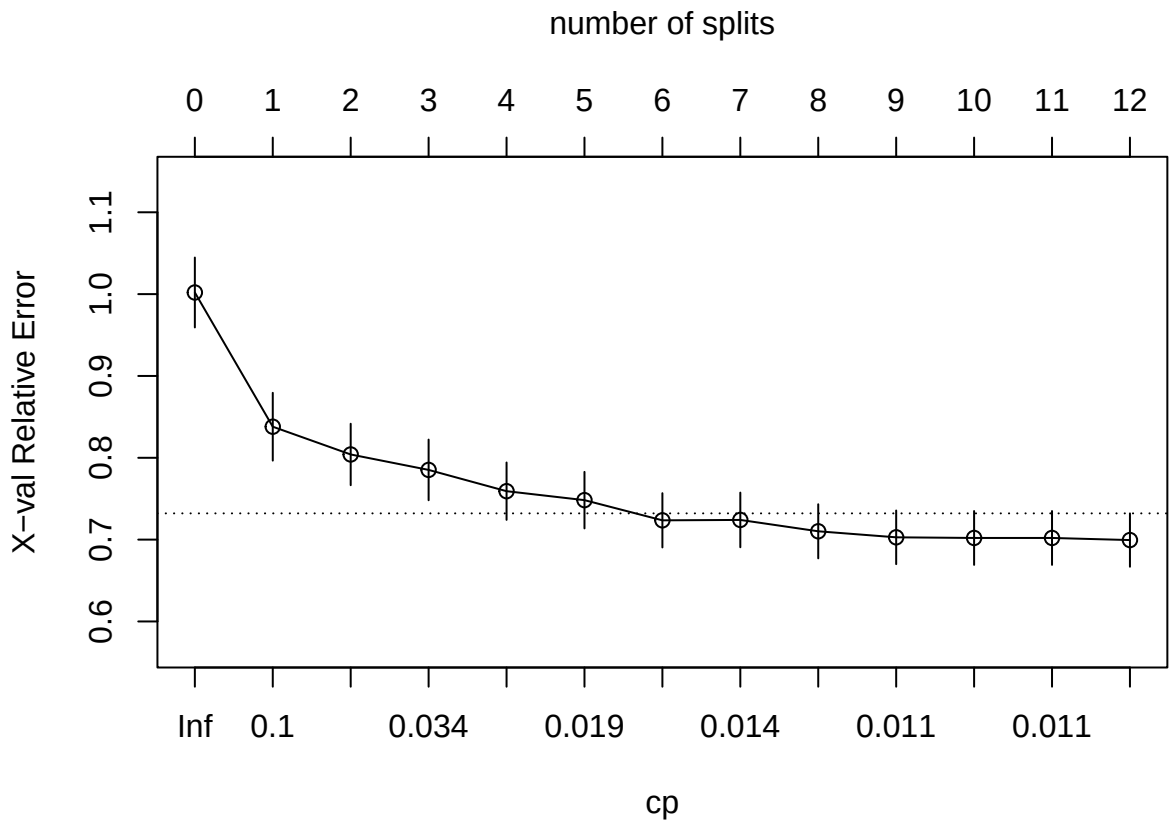
If we want the lowest possible error, we'd prune to the tree with the smallest relative SSE. However, we wish to balance predictive power with simplicty so we will prune to a tree relative the smallest relative SSE. Unsurprisingly, we get the lowest xerror with 11 splits. Let's try to be within two standard deviation of 11 split's xerror. This gives us 4 splits, which is pretty consistent with our other cross validated pruned tree model.

```
index <- which(regression.tree$cptable[,"xerror"] == min(regression.tree$cptable[,"xerror"]))
(threshold <- regression.tree$cptable[index,"xerror"] + 2 * regression.tree$cptable[index,"xstd"])
```

```
## [1] 0.76472
```

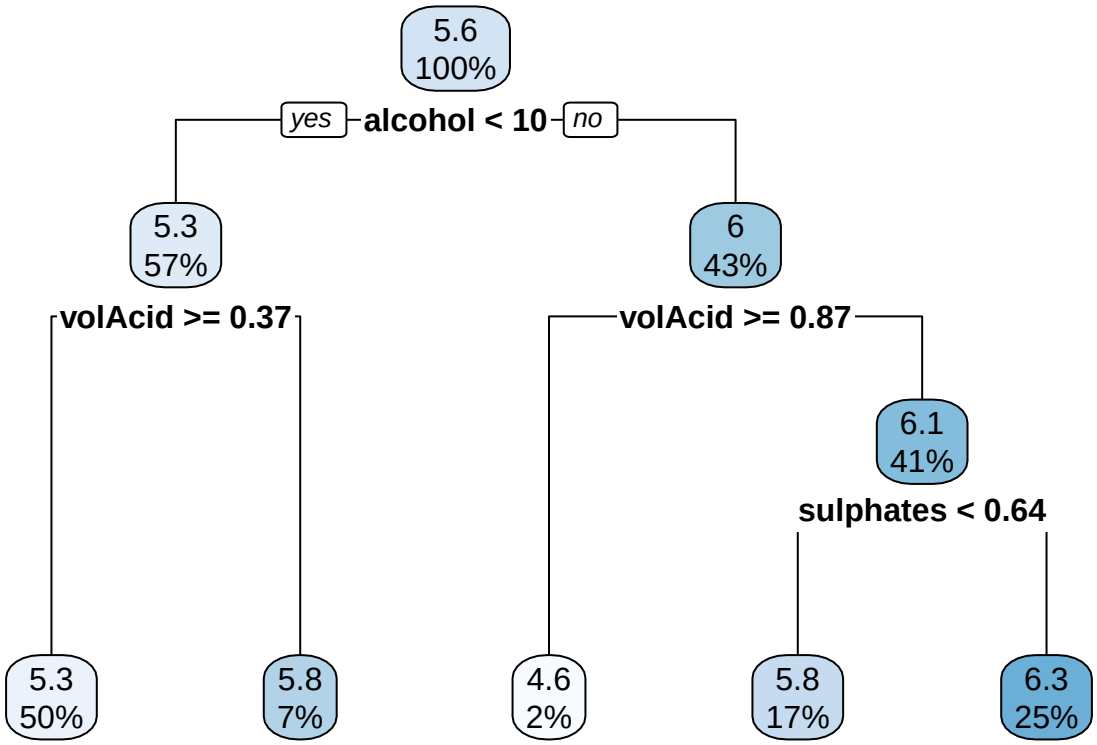
Even the graph shows similiar results.

```
plotcp(regression.tree, upper = "splits")
```



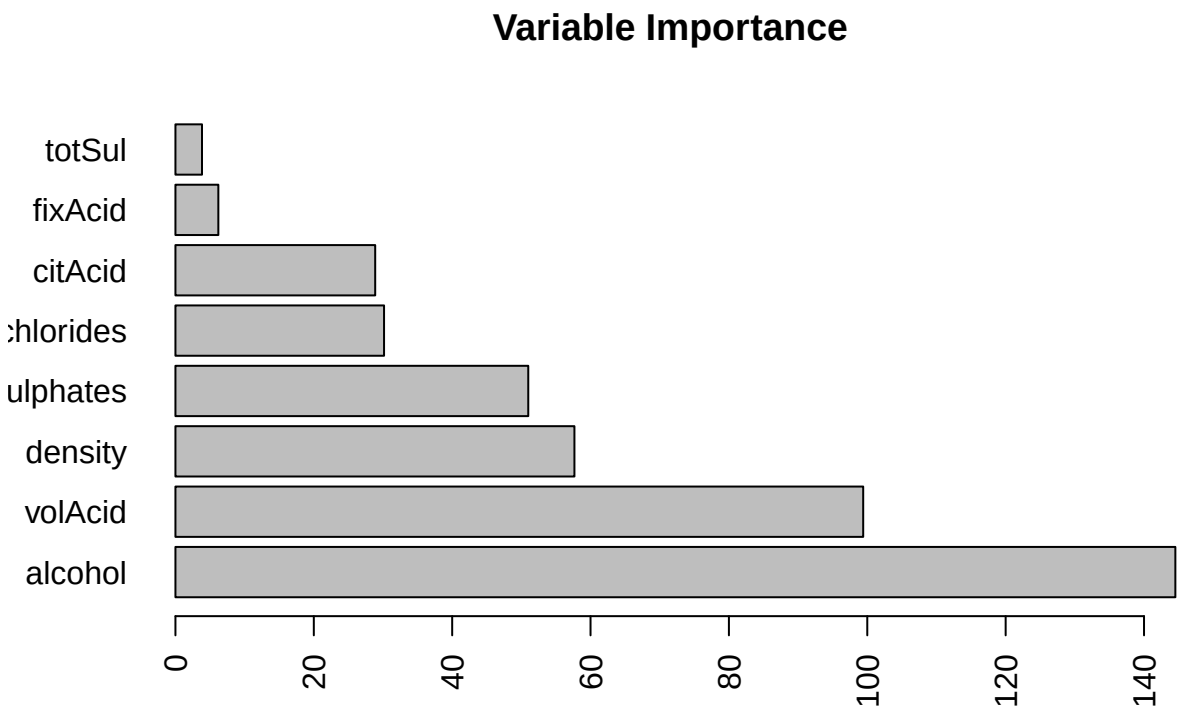
Let's prune the tree real quick.

```
regression.prune <- prune(regression.tree, cp = regression.tree$cptable[regression.tree$cptable[, 2] ==
rpart.plot(regression.prune, yesno = TRUE)
```



Looking at the variable importance, we see Alcohol, Volatile Acid and Acid are most importance.

```
barplot(regression.prune$variable.importance, main = "Variable Importance", horiz = TRUE, las = 2)
```



With this train test split we get a $RMSE = 0.723623$.

```
preds <- predict(regression.prune, testing, type = "vector")
RMSE(pred = preds, obs = testing$quality)
```

```
## [1] 0.723623
```

Now let's rebuild the model using caret(). We'll make a 10-fold cross validation split to optimize the hyper-parameter CP.

```
set.seed(42)
trControl <- trainControl(method = "cv", number = 10, savePredictions = "final")
regression.cv1 <- train(quality ~ ., data = training, method = "rpart", tuneLength = 5,  metric = "RMSE
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

We see we get the best RMSE when $cp = 0.02393885$ We can then refine our search near this location.

```
print(regression.cv1)
```

```
## CART
##
## 1281 samples
```

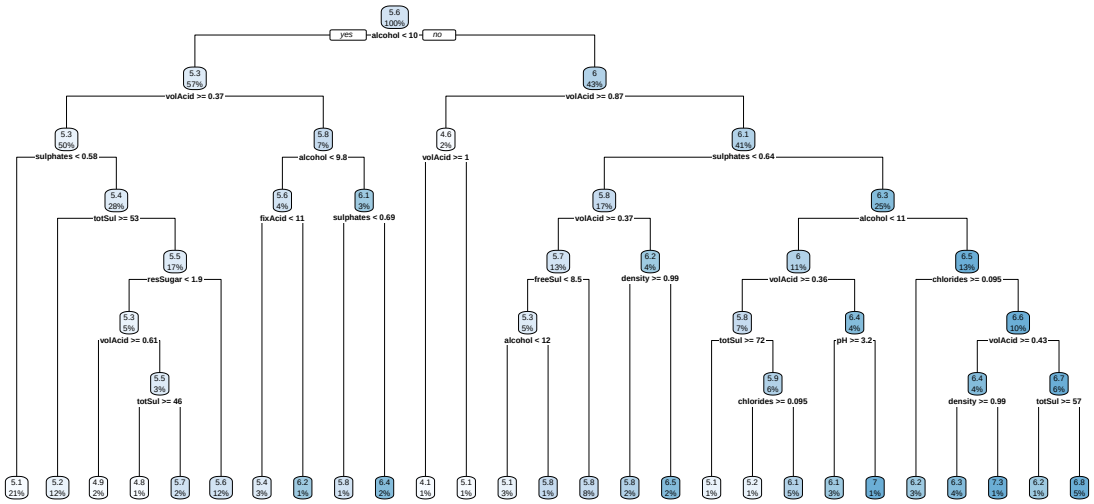
```
##   11 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1152, 1154, 1153, 1153, 1154, 1153, ...
## Resampling results across tuning parameters:
##
##   cp          RMSE       Rsquared   MAE
##   0.02393885  0.7051881  0.2448153  0.5607436
##   0.02971815  0.7110340  0.2307733  0.5757793
##   0.03993098  0.7288515  0.1920910  0.5904613
##   0.05972099  0.7391724  0.1670755  0.5802692
##   0.17319787  0.7619473  0.1382685  0.5986842
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.02393885.
```

We see we get the best RMSE when $cp = 0.005$.

```
searchGrid <- expand.grid(cp = seq(from = 0, to = 0.05, by = 0.005))
regression.cv2 <- train(quality ~ ., data = training, method = "rpart", tuneGrid  = searchGrid,  metric
print(regression.cv2)
```

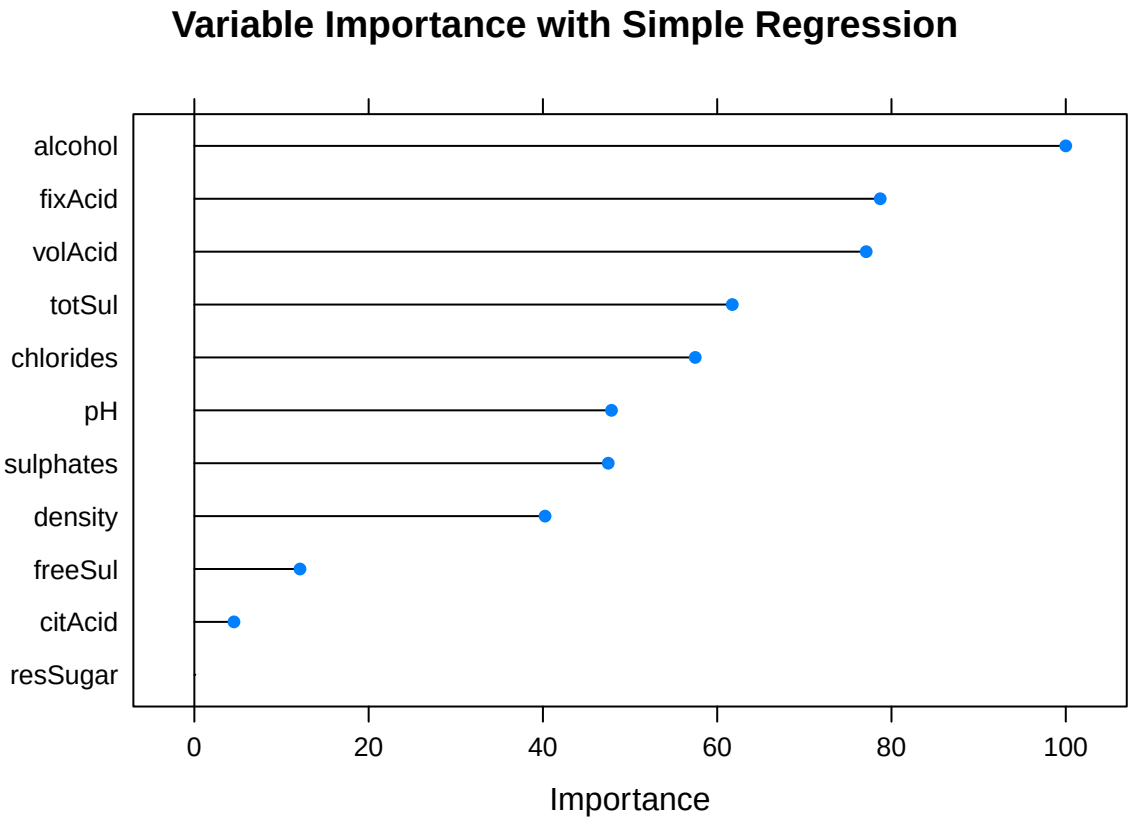
```
## CART
##
## 1281 samples
##   11 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1153, 1154, 1153, 1153, 1153, 1153, ...
## Resampling results across tuning parameters:
##
##   cp     RMSE       Rsquared   MAE
##   0.000  0.7047080  0.3122235  0.5292237
##   0.005  0.6642924  0.3424930  0.5025219
##   0.010  0.6690460  0.3222036  0.5196286
##   0.015  0.6869480  0.2841217  0.5482143
##   0.020  0.6896632  0.2746346  0.5529808
##   0.025  0.6919295  0.2699329  0.5554084
##   0.030  0.6953739  0.2613797  0.5641368
##   0.035  0.7076564  0.2360447  0.5803806
##   0.040  0.7201499  0.2086288  0.5891018
##   0.045  0.7233546  0.2012585  0.5861612
##   0.050  0.7233546  0.2012585  0.5861612
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.005.
```

```
rpart.plot(regression.cv2$finalModel)
```



We get the same approximate variable importance plots.

```
plot(varImp(regression.cv2), main="Variable Importance with Simple Regression")
```



We see using the cross-validated pruned tree, we get a $RMSE = 0.6736749$.

```
preds.cv <- predict(regression.cv2, testing, type = "raw")
RMSE(pred = preds.cv, obs = testing$quality)
```

```
## [1] 0.6736749
```

Classification Tree

Now, let's create a classification tree by turning our quality variable into a binary response variable corresponding to Good and Bad split when $quality = 6$.

```
response <- factor(ifelse(wine$quality <= 6, "Bad", "Good"))
wine.class <- data.frame(wine, response)
wine.class <- wine.class[, -12] # remove quality response
```

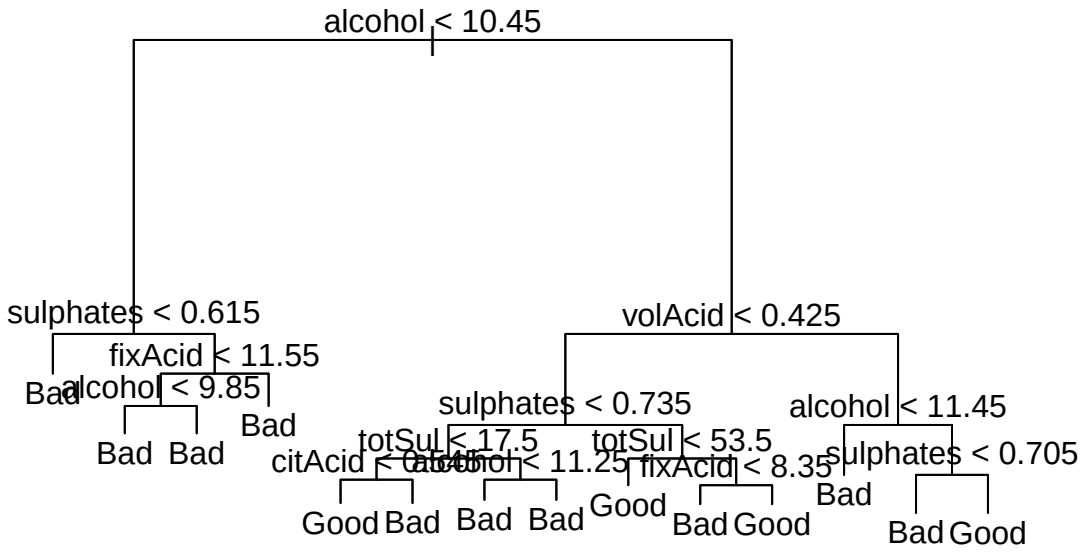
Here is our basic tree. We currently have a training error rate of 0.09944.

```
tree.wine <- tree(response ~ ., wine.class)
summary(tree.wine)
```

```
##
## Classification tree:
## tree(formula = response ~ ., data = wine.class)
```

```
## Variables actually used in tree construction:
## [1] "alcohol"   "sulphates" "fixAcid"   "volAcid"   "totSul"    "citAcid"
## Number of terminal nodes:  14
## Residual mean deviance:  0.4803 = 761.2 / 1585
## Misclassification error rate: 0.09944 = 159 / 1599
```

```
plot(tree.wine)
text(tree.wine, pretty = 0)
```



Let's use a basic split. We get correction test predictions around 0.884375 of the time.

```
set.seed(42)
train <- sample(1:nrow(wine.class), nrow(wine.class)*0.8)
wine.class.test <- wine.class[-train, ]
response.test <- response[-train]
tree.wine <- tree(response ~ ., wine.class, subset = train)
tree.pred <- predict(tree.wine, wine.class.test, type = "class")
table(tree.pred, response.test)
```

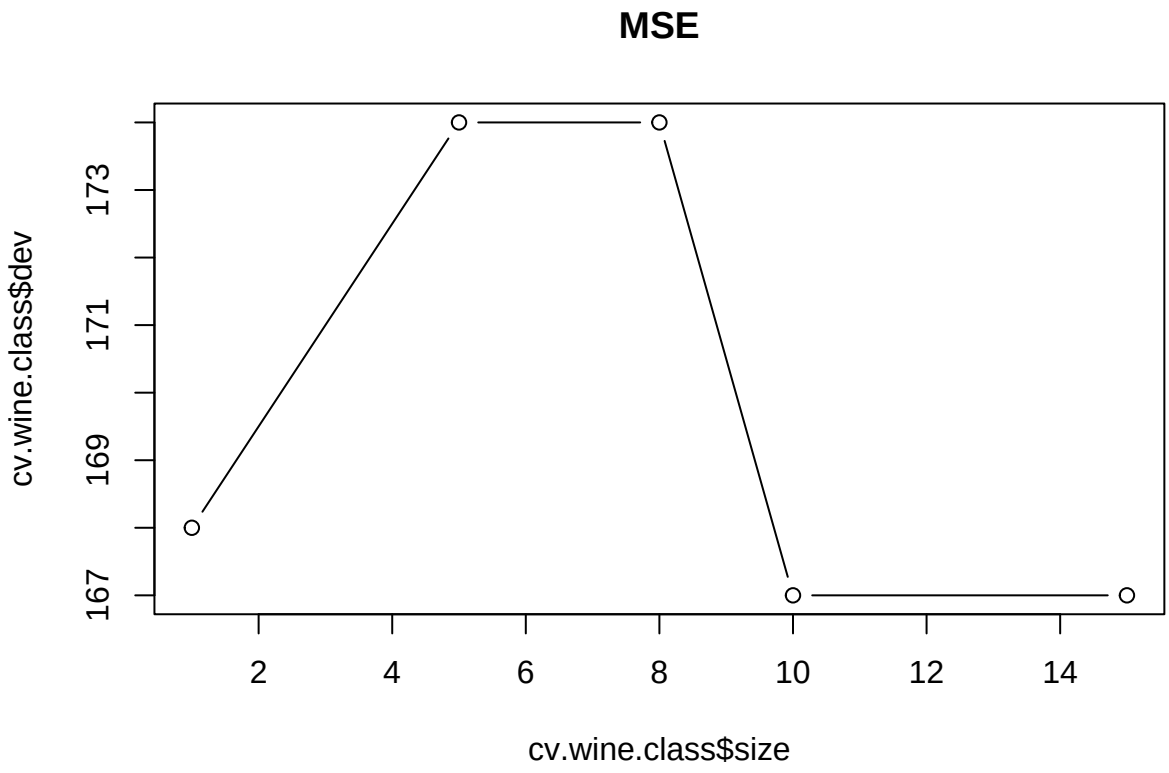
```
##          response.test
## tree.pred Bad Good
##      Bad  258   35
##      Good  11   16
```

```
(267 + 16) / 320
```

```
## [1] 0.884375
```

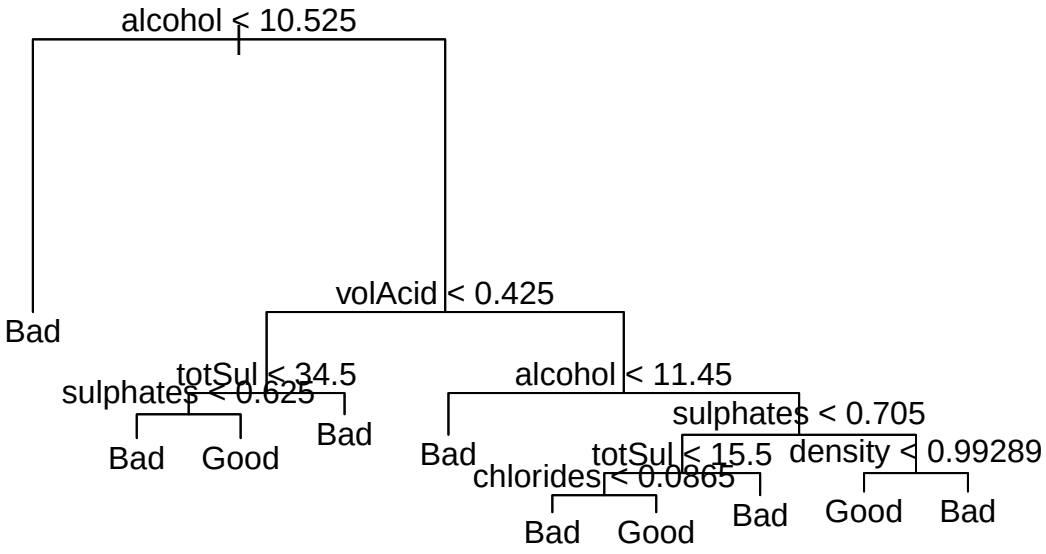
Now, lets perform cross-validation in order to see the best level of tree complexity.

```
set.seed(42)
cv.wine.class <- cv.tree(tree.wine, FUN = prune.misclass)
plot(cv.wine.class$size, cv.wine.class$dev, type = "b", main = "MSE")
```



Lets apply the prune function to obtain a seven-node tree.

```
prune.wine.class <- prune.misclass(tree.wine, best = 9)
plot(prune.wine.class)
text(prune.wine.class, pretty = 0)
```



This time we get a 0.85625 of the test observations correctly classified.

```
tree.pred <- predict(prune.wine.class, wine.class.test, type = "class")
table(tree.pred, response.test)
```

```
##          response.test
## tree.pred Bad Good
##      Bad  258   35
##      Good  11   16
```

```
(258 + 16) / 320
```

```
## [1] 0.85625
```

With our cross-validated classification tree, we get an optimal $cp = 0.02150538$

```
set.seed(42)
trControl <- trainControl(method = "cv", number = 10, savePredictions = "final")
classification.cv1 <- train(response ~ ., data = wine.class, method = "rpart", tuneLength = 5, trControl
print(classification.cv1)
```

```
## CART
##
## 1599 samples
```

```
##   11 predictor
##    2 classes: 'Bad', 'Good'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1439, 1439, 1439, 1439, 1438, 1440, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.02150538  0.8881034  0.4276793
##   0.02304147  0.8824665  0.4017190
##   0.02764977  0.8824665  0.3474272
##   0.03225806  0.8805915  0.3487530
##   0.08294931  0.8667904  0.1152164
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.02150538.
```

With this cross validated classification tree, we get an accuracy of 0.9.

```
preds.cv <- predict(classification.cv1, wine.class.test, type = "raw")
table(preds.cv, response.test)
```

```
##         response.test
## preds.cv Bad Good
##     Bad  265   28
##     Good   4   23
```

```
(265+23)/320
```

```
## [1] 0.9
```

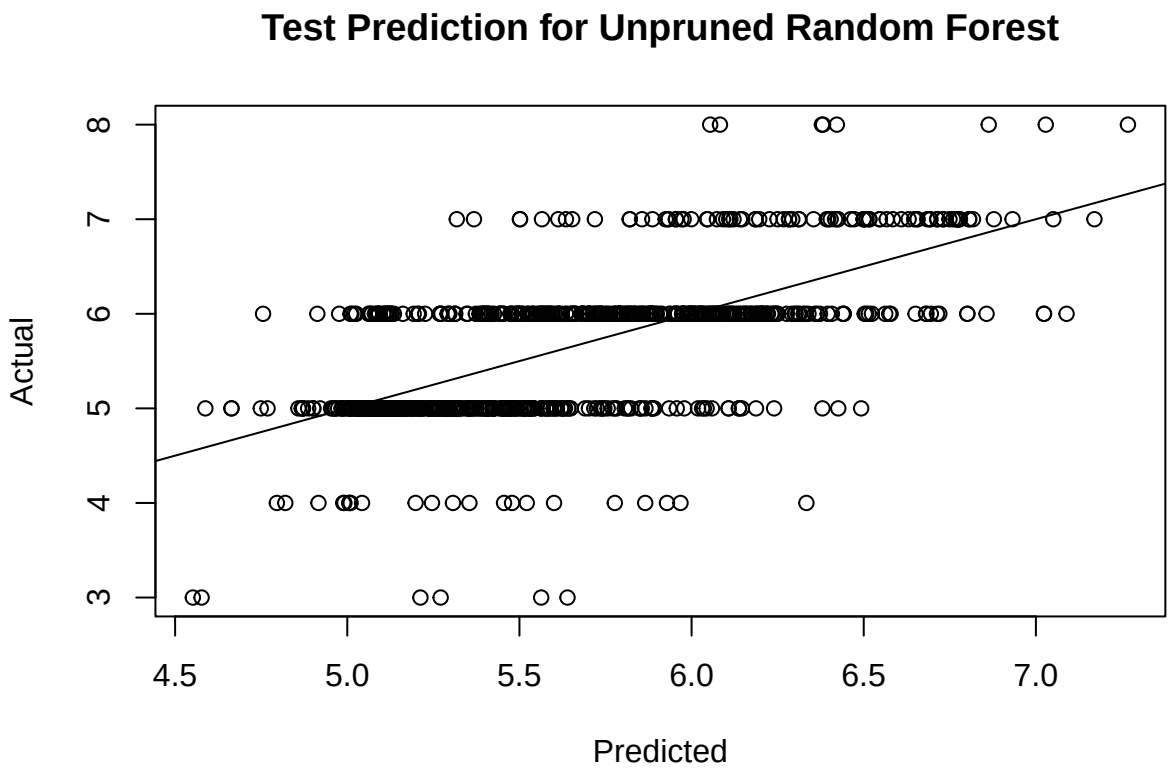
Random Forest

Now let's create a randomForest. The argument mytry=11 indicates all 11 predictors should be considered for each split of the tree

```
set.seed(123)
train <- sample(1:nrow(wine), nrow(wine) / 2)
wine.test <- wine[-train, "quality"]
bag.wine <- randomForest(quality ~ ., data = wine,
    subset = train, mtry = 11, importance = TRUE)
bag.wine
```

```
##
## Call:
##  randomForest(formula = quality ~ ., data = wine, mtry = 11, importance = TRUE,      subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##           Mean of squared residuals: 0.4094131
##                     % Var explained: 40.16
```

```
yhat.bag <- predict(bag.wine, newdata = wine[-train, ])
plot(yhat.bag, wine.test$quality, xlab = "Predicted", ylab = "Actual", main = "Test Prediction for Unpr
abline(0, 1)
```



The test set MSE associated with the bagged regression tree is $MSE = 0.3548118$

```
mean((yhat.bag - wine.test$quality)^2)
```

```
## [1] 0.3548118
```

Let's change up the number of trees.

```
rf.wine <- randomForest(quality ~ ., data =wine,
    subset = train, mtry = 11, ntree=25)
yhat.rf <- predict(rf.wine, newdata = wine[-train, ])
```

The test set MSE associated with the random forest is $MSE = 0.3781385$.

```
mean((yhat.rf - wine.test$quality)^2)
```

```
## [1] 0.3781385
```

Now let's try a smaller value of mtry.

```
rf.wine <- randomForest(quality ~ ., data =wine, subset = train, mtry = 5)
yhat.rf <- predict(rf.wine, newdata = wine[-train, ])
```

The test set MSE associated with random forest is $MSE = 0.347583$. The MSE improved. Random forest yields a better MSE than bagging in this case.

```
mean((yhat.rf - wine.test$quality)^2)
```

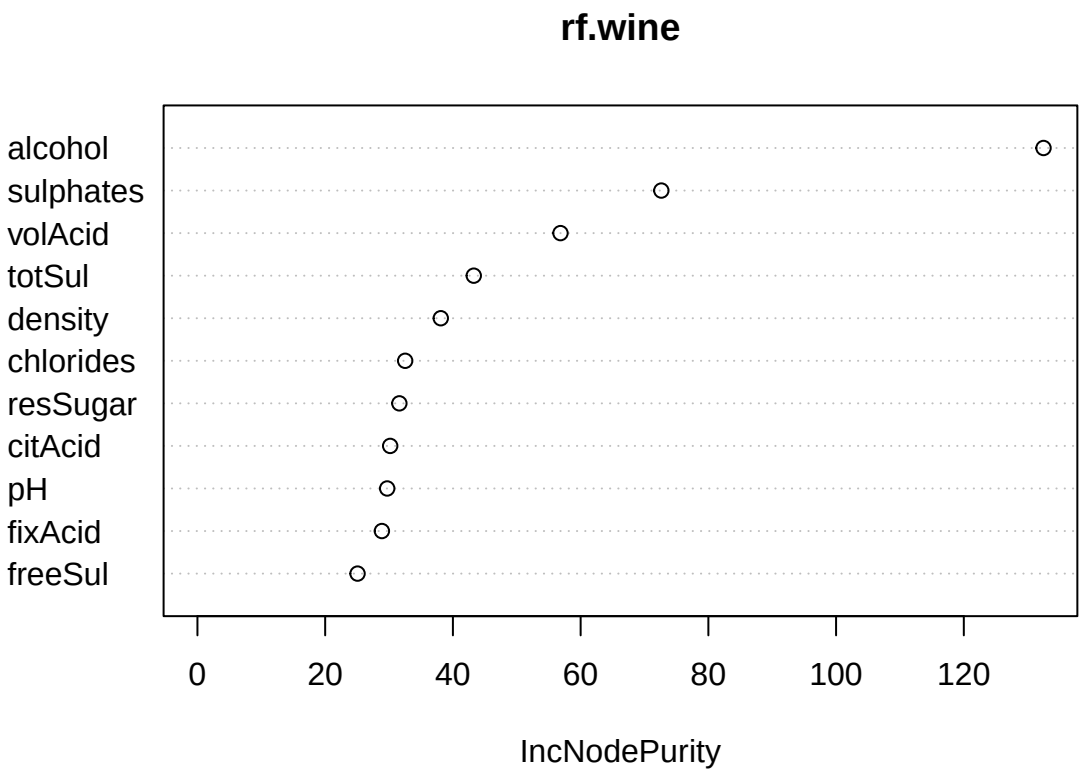
```
## [1] 0.347583
```

As we check the importance of the variables and plot the importance, we can see similiar results as before. The plot indicates that across all tress considered in random forest, the percent alcohol content of the wine (alcohol) and wine additive that contribute to SO2 levels (sulfates) are the two most important variables.

```
importance(rf.wine)
```

```
##           IncNodePurity
## fixAcid        28.88084
## volAcid        56.84413
## citAcid        30.17830
## resSugar       31.61817
## chlorides      32.52554
## freeSul        25.06836
## totSul         43.26580
## density        38.10254
## pH             29.71112
## sulphates      72.63673
## alcohol       132.47905
```

```
varImpPlot(rf.wine)
```



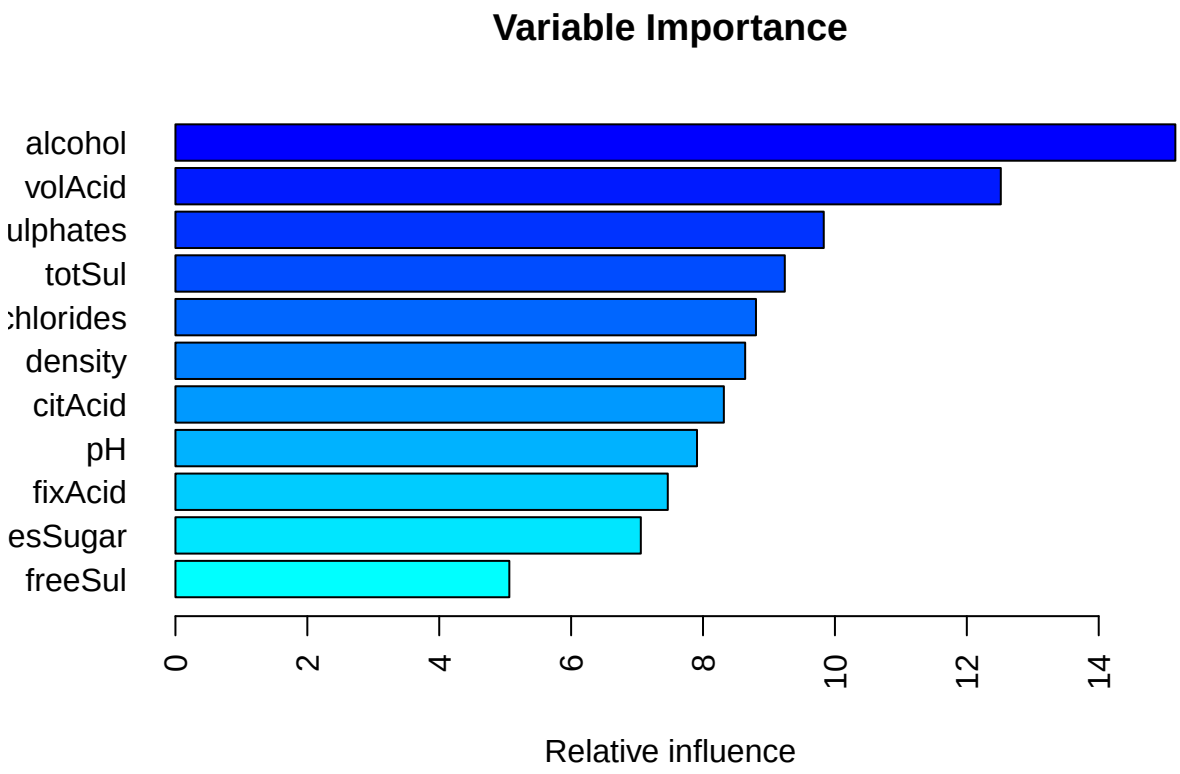
Boosting

Now let's perform boosting. We will use 5000 trees. Setting the interaction.depth to 4 will limit the depth of each tree.

```
set.seed(123)
boost.wine <- gbm(quality ~ ., data = wine[train, ],
    distribution = "gaussian", n.trees = 5000, interaction.depth = 4)
```

As we can from the variable importance, alcohol and volatile acidity are by far the most important variables.

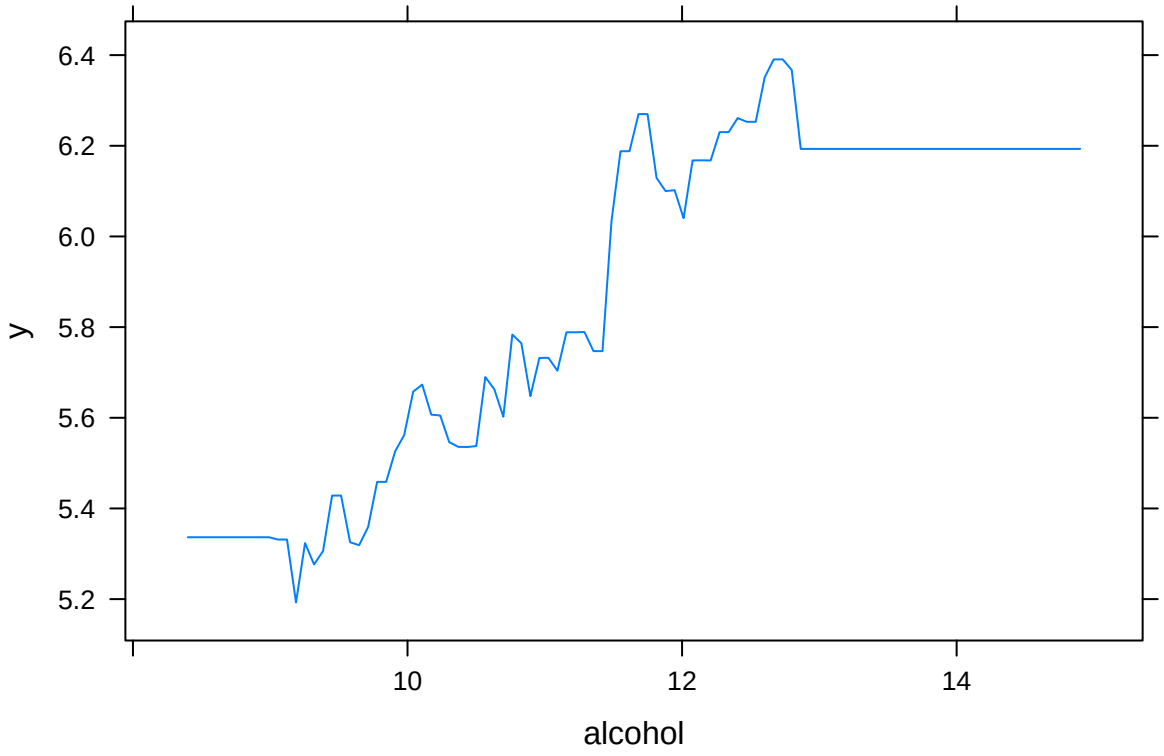
```
summary(boost.wine, las = 2, main = "Variable Importance")
```



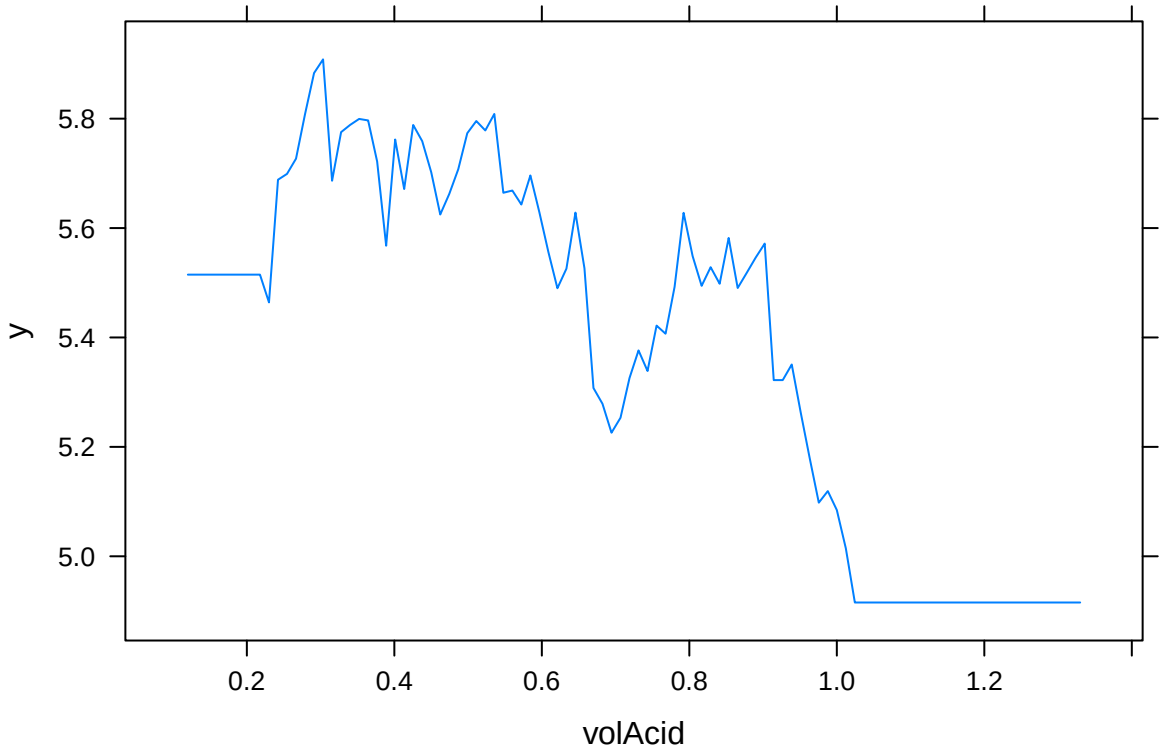

```
##                 var   rel.inf
## alcohol     alcohol 15.162165
## volAcid     volAcid 12.515183
## sulphates sulphates  9.830801
## totSul       totSul  9.239876
## chlorides chlorides  8.802799
## density     density  8.639175
## citAcid     citAcid  8.316073
## pH               pH  7.908865
## fixAcid     fixAcid  7.465531
## resSugar   resSugar  7.056779
## freeSul     freeSul  5.062752
```

Now, let's look at the partial dependence plot.

```
par(mfrow=c(1,2))
plot(boost.wine, i = "alcohol")
```




```
plot(boost.wine, i = "volAcid")
```



Using prediction, we get a $MSE = 0.4732347$

```
yhat.boost <- predict(boost.wine, newdata = wine[-train, ], n.trees = 5000)
mean((yhat.boost - wine.test$quality)^2)
```

```
## [1] 0.4732347
```

Now let's try a different shrinking parameter lambda. With this, we get a $MSE = 0.426222$. Our shrinkage parameter become 0.03.

```
boost.wine <- gbm(quality ~ ., data = wine[train, ],
    distribution = "gaussian", n.trees = 5000,
    interaction.depth = 4, shrinkage = 0.03, verbose = F)
yhat.boost <- predict(boost.wine, newdata = wine[-train, ], n.trees = 5000)
mean((yhat.boost - wine.test$quality)^2)
```

```
## [1] 0.426222
```

```
boost.wine$shrinkage
```

```
## [1] 0.03
```

In summation, we get an classification accuracy of 0.9 with our cross-validated classification tree. For our regression analysis, we get the best results for the following models: $RMSE = 0.6736749$ for regression tree, $RMSE = 0.5895617$ for random forest and $RMSE = 0.6528568$ for boosting.