Regression Tree Interpolation

Alaina Stockdill

07/1/2021

Import libraries

```r
library(tidyverse)
library(lubridate)
library(stringr)
library(caret)
library(rattle)
library(rpart)
library(kableExtra)
library(dplyr)
```

Predicting missing values in the LTRM data with regression trees

In order to get the LTRM data ready to use TDAmapper, we need to interpolate the missing values. While many methods have been considered, including linear/polynomial/multivariate regressions and inverse distance interpolation, we will also explore interpolation using regression trees. The benefit of the CART regression tree is that they will automatically choose the most important variables in predicting the target variables and are able to handle missing data very easily. With a data set that contains a large amount of incomplete rows, it seems that this method will be helpful in getting around this issue.

The data set that we are using has been cleaned to:

1. Replace values with bad QF codes with NA
2. Replace negative values for TP, TN, SS, TEMP, and with the minimum recorded value
3. Replace negatives WDP values with 0
4. Filtered for surface samples (excluded mid and bottom samples)
5. Filtered for random samples (excluded fixed sites in river tributaries)
6. Replaced TN outliers (TN > 25) with `NA`
7. Combined rows that had the sample SHEETBAR values

```r
setwd("/Users/alainastockdill/UMR-TDA-2021/InterpolationCode/CART interpolation")
water_data <- read.csv(file = "../../LTRM data/water_data_qfneg.csv")
```

Date and season are added in so that the model can use these values as predictors. Because these variables are included as predictors, we will not need to make separate trees from data that is filtered by these different values. Field number, stratum, and season are all made as categorical variables.

```r
# Add in the date, year, and season
water_data <- water_data %>%
  mutate(nice_date = mdy(DATE),
         year = year(nice_date),
         season = quarter(nice_date, fiscal_start = 3)) %>%
  select(-SHEETBAR, -nice_date, -DATE, -LOCATCD)
```

```r
# Make the FLDNUM and STRATUM categorical variables
water_data$FLDNUM <- as.character(water_data$FLDNUM)
water_data$STRATUM <- as.character(water_data$STRATUM)
water_data$season <- as.character(water_data$season)

water_vars <- list("TP", "TN", "VEL") #, "TURB", "COND","WDP", "SECCHI", "CHLcal", "SS", "TEMP", "DO")

predictor_vars <- c("TP", "TN", "TURB", "COND", "VEL", "WDP", "season", "STRATUM", "FLDNUM", "CHLcal",

rt_data <- water_data[predictor_vars] # regression tree data
```

Assumptions

While there are missing values in all continuous variables, we will be begin by interpolating for only TP, TN, and VEL, the variables with the most missingness. We will be using TP, TN, TURB, COND, VEL, WDP, season, year, STRATUM, FLDNUM, CHLcal, SS, TEMP, SECCHI, and DO as our predictor variables. We will use an 80/20 train-test split to create the model and test for its accuracy. The train and test data for the tree for each interpolated variable will be made from a subset of the entire cleaned UMR data set that has no missing values for the current target variable.

After having created this models, we used the 1-SE rule to set the CP control method. Since TN is so poorly predicted, we will be only taking into account the CP value suggested for TP and VEL. The CP value for TP needed to be between .015 and .01 and for VEL it needed to be between .01 and .03. We will use CP = .011.

The MAE, RMSE, and summary stats for the regression tree models for interpolating TP, TN, VEL will be calculated using the test data.

Creating the model

The creation of the model for each target variable (the variable we are attempting to interpolate) will involve the following process:

1. Create a subset of the entire data set that has a complete column of the variable we are currently estimating
2. Create a 80/20 train/test split of the data
3. Train a regression tree model to predict a target variable with all the other predictors
4. Test the regression model with the test data
5. Calculate the RMSE, MAE, and summary stats for the actual versus predicted values
6. Plot the actual vs. predicted scatter plots, boxplots, and histograms

We will run the model first using surrogates splits and then a second time using median imputation to compare the two methods.

Data imputation functions

Written by Amber

```r
# which predictor variables have no missingness?
# names(water20)[sapply(water20, function(x) sum(!is.na(x)) == 82481)]
# "FLDNUM"  "STRATUM" "year"    "quarter"


replace_median_mode <- function(var_str, dataset, response_var){
  # for imputing test data with median/mode

  # iterate through each variable name var_str with sapply to
  # replace every variable except for response_var (no need to impute)

  # column of the dataset
  data_col <- eval(parse(text = paste("dataset$", var_str, sep = "")))

  if (var_str %in% c(response_var, "FLDNUM",  "STRATUM", "year",  "quarter")){
    # these other variables have no missingness
    return(data_col) # don't impute the response variable
  }

  if (class(data_col) == "factor"){
    # categorical imputation
    # take the mode

    counts_table <- table(data_col)

    # retrieve mode
    imputation <- names(counts_table)[counts_table == max(counts_table)]

  } else if (class(data_col) %in% c("numeric", "integer")){
    # continuous imputation
    # take the median

    imputation <- median(data_col, na.rm = TRUE)

  } else (return("error in class of variable"))

  data_col[is.na(data_col)] <- imputation

  #   print("running replace_median_mode")
  print(paste(var_str, "imputation is", as.character(imputation)))

  if (sum(!is.na(data_col)) == length(data_col)){

    return(data_col)

  } else return("error in imputing NAs")

}

impute_data <- function(df, response_var, df_train){

  df <- data.frame(sapply(names(df), replace_median_mode, df, response_var))
  # trick to fix incompatible types for random forest
  # source: https://stackoverflow.com/questions/24829674/r-random-forest-error-type-of-predictors-in-ne

  df <- rbind(df_train[1, ] , df)
  df <- df[-1,]

  return(df)

}
```

Model evalution functions

These functions were written by Amber to compile the RMSE, MAE, and summary stats for each predicted variables. error_df and distribution_df contain the errors and summary stats for the regression trees when the surrogate splits are used. error_df_imp and distribution_df_imp contain the errors and summary stats for regression trees when median imputation is used.

```
error_df <- data.frame("Response" = as.character(),
                       "RSME" = as.numeric(),
                       "MAE" = as.numeric(),
                       `Test data size` = as.numeric())

error_df_imp <- data.frame("Response" = as.character(),
                       "RSME" = as.numeric(),
                       "MAE" = as.numeric(),
                       `Test data size` = as.numeric())

add_errors <- function(error_df, response_var, test_data){
  # the test_data has a column called predicted
  # that has RF predictions without NAs

  test_data <- test_data %>%
    mutate(residual = !!sym(response_var) - predicted,
           residual_sq = residual^2,
           abs_residual = abs(residual))


  n <- round(dim(test_data)[1], 3)
  rsme <- round(sqrt(sum(test_data$residual_sq)/n), 5)
  mae <- round(sum(test_data$abs_residual)/n, 5)


  return(rbind(error_df, data.frame("Response" = response_var,
                                    "RSME" = rsme,
                                    "MAE" = mae,
                                    `Test data size` = n)))

}

distribution_df <- data.frame("Response" = as.character(),
                              "Mean" = as.numeric(),
                              "Minimum" = as.numeric(),
                              "Quartile 1" = as.numeric(),
                              "Median" = as.numeric(),
                              "Quartile 3" = as.numeric(),
                              "Maximum" = as.numeric())

distribution_df_imp <- data.frame("Response" = as.character(),
                              "Mean" = as.numeric(),
                              "Minimum" = as.numeric(),
                              "Quartile 1" = as.numeric(),
                              "Median" = as.numeric(),
                              "Quartile 3" = as.numeric(),
                              "Maximum" = as.numeric())
```

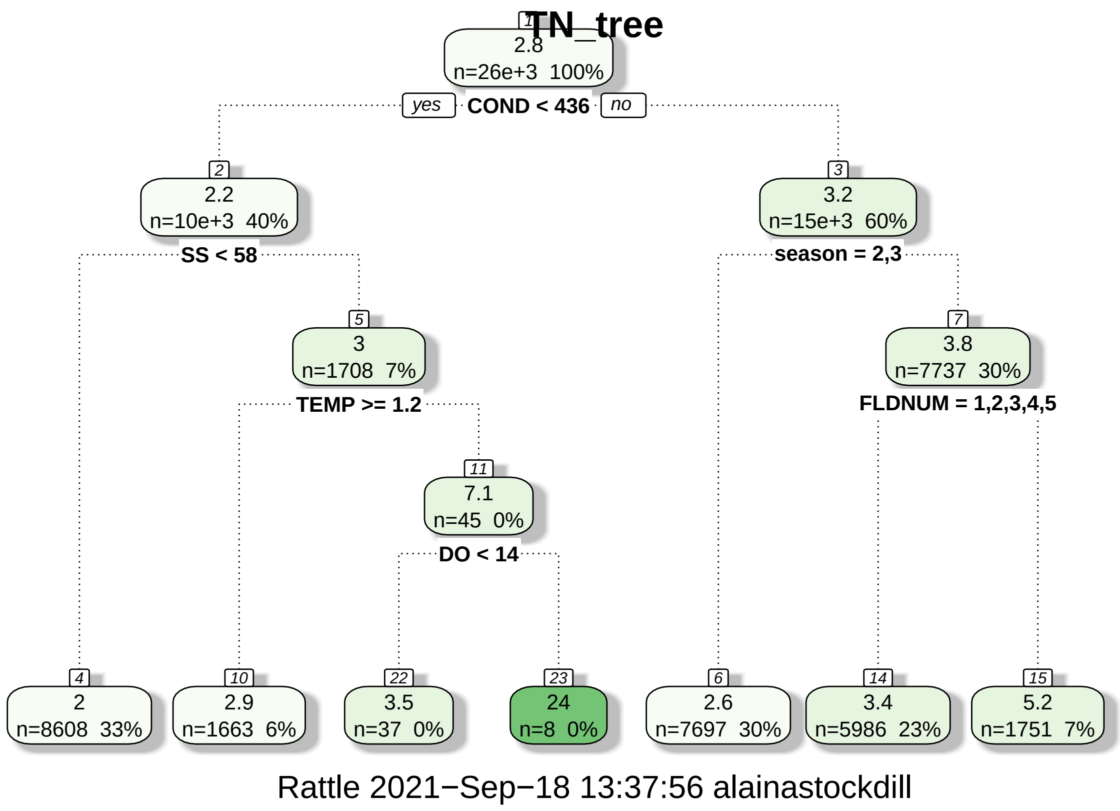
```
add_distribution <- function(distribution_df, response_var, test_data){

  predictions <- test_data$predicted

  return(rbind(distribution_df,
               data.frame("Response" = response_var,
                          "Mean" = round(mean(predictions), 5),
                          "Minimum" = round(min(predictions), 5),
                          "Quartile 1" = round(quantile(predictions, 0.25), 5),
                          "Median" = round(median(predictions), 5),
                          "Quartile 3" = round(quantile(predictions, 0.50), 5),
                          "Maximum" = round(max(predictions), 5))))
}
```

TN model

```

#Create a new data set that removes only the na TN values
 water_TN = rt_data %>% filter(!is.na(TN))

set.seed(571)
sample_size = 0.80 * nrow(water_TN)
train_indices <- sample(seq_len(nrow(water_TN)), size = sample_size)

tn_train <- water_TN[train_indices, ]
tn_test <- water_TN[-train_indices, ]

tr.TN_rpart <- rpart(TN ~ .,
                     data = tn_train,
                     method = "anova",
                     control = rpart.control(cp = .011) )
```

Predicting with surrogate splits

```
fancyRpartPlot(tr.TN_rpart, main = "TN_tree")
```

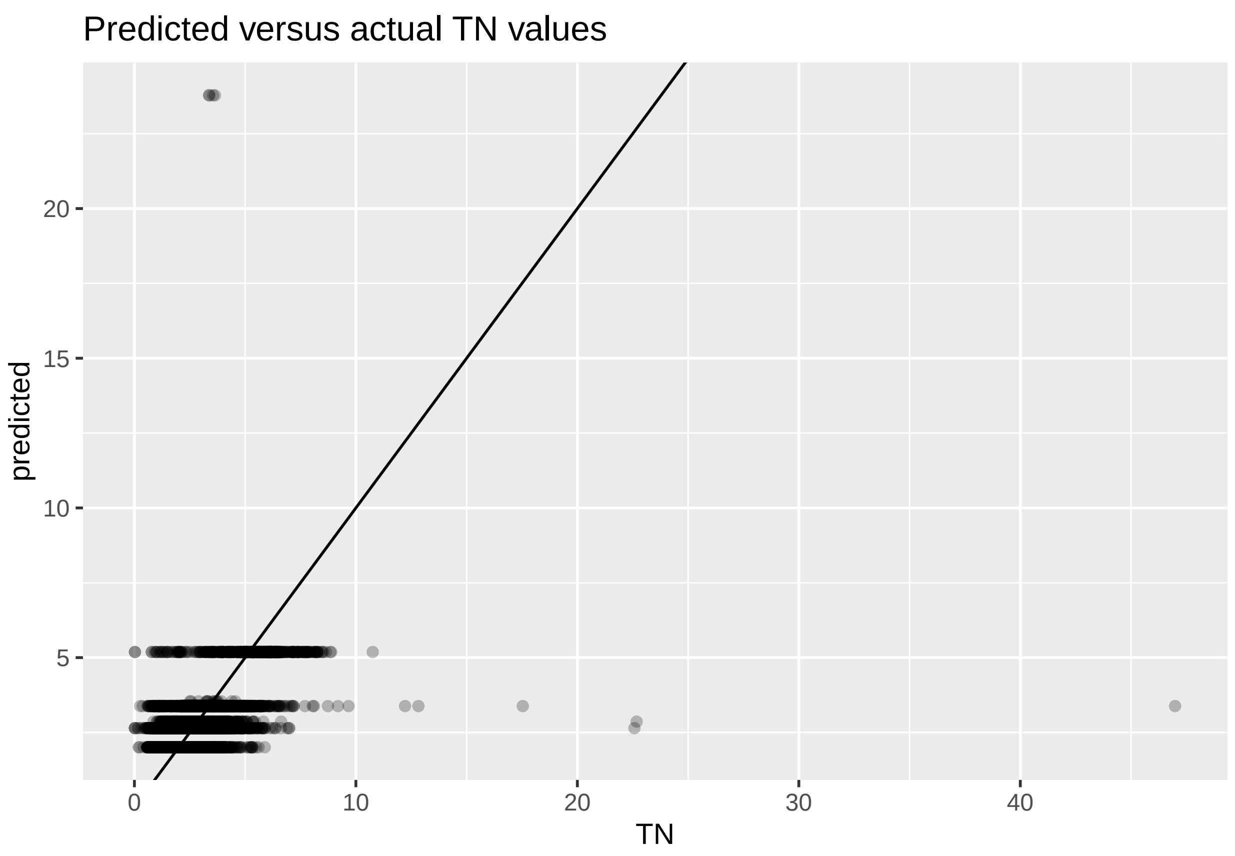


```
printcp(tr.TN_rpart)
```

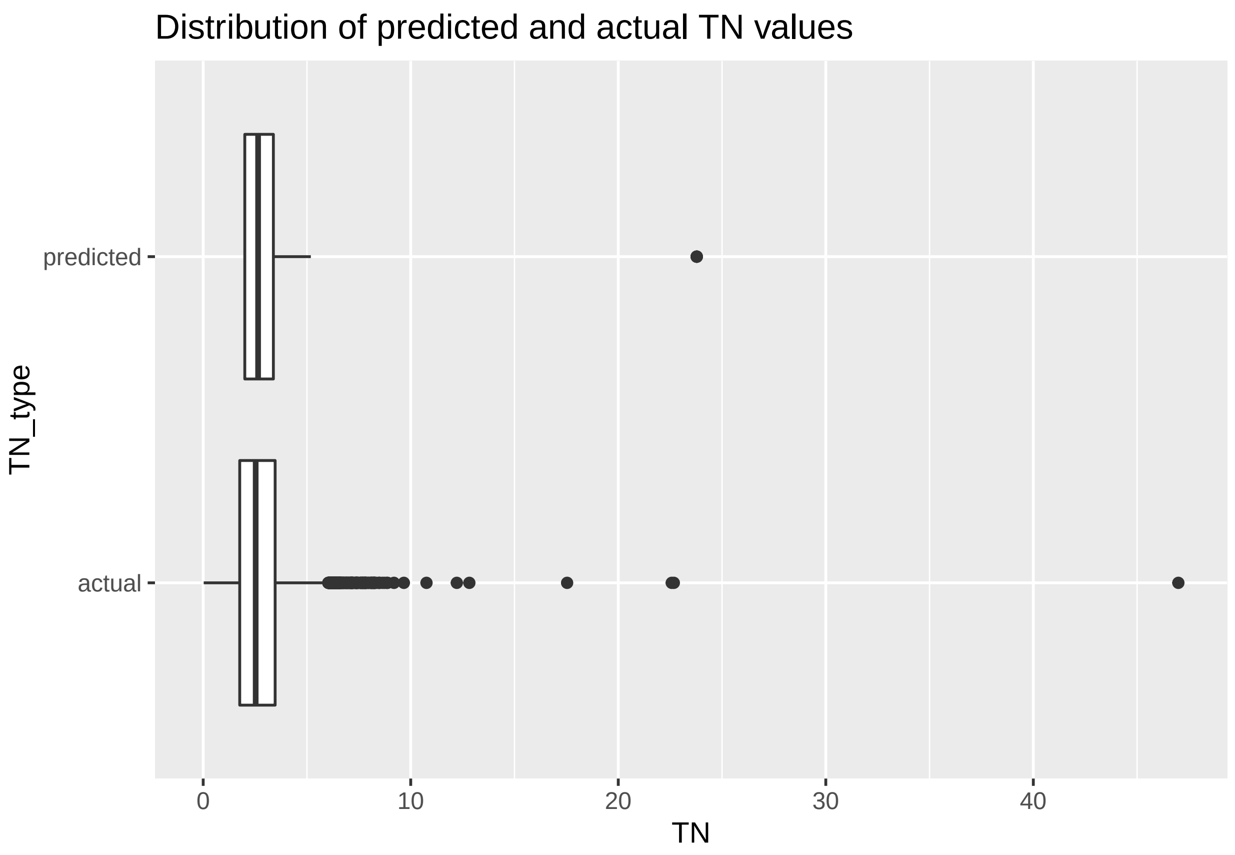
```
##
## Regression tree:
## rpart(formula = TN ~ ., data = tn_train, method = "anova", control = rpart.control(cp = 0.011))
##
## Variables actually used in tree construction:
## [1] COND   DO     FLDNUM season SS     TEMP
##
## Root node error: 133803/25750 = 5.1962
##
## n= 25750
##
##          CP nsplit rel error  xerror    xstd
## 1 0.051362      0   1.00000 1.00009 0.36034
## 2 0.037825      1   0.94864 0.95155 0.36093
## 3 0.033000      2   0.91081 0.91864 0.36208
## 4 0.012054      3   0.87781 0.88961 0.36211
## 5 0.011000      6   0.84165 0.91174 0.36250
```

```
tn_test$predicted <- predict(tr.TN_rpart, tn_test)

# Summary stats
error_df <- add_errors(error_df, "TN", tn_test)
distribution_df <- add_distribution(distribution_df, "TN", tn_test)
```

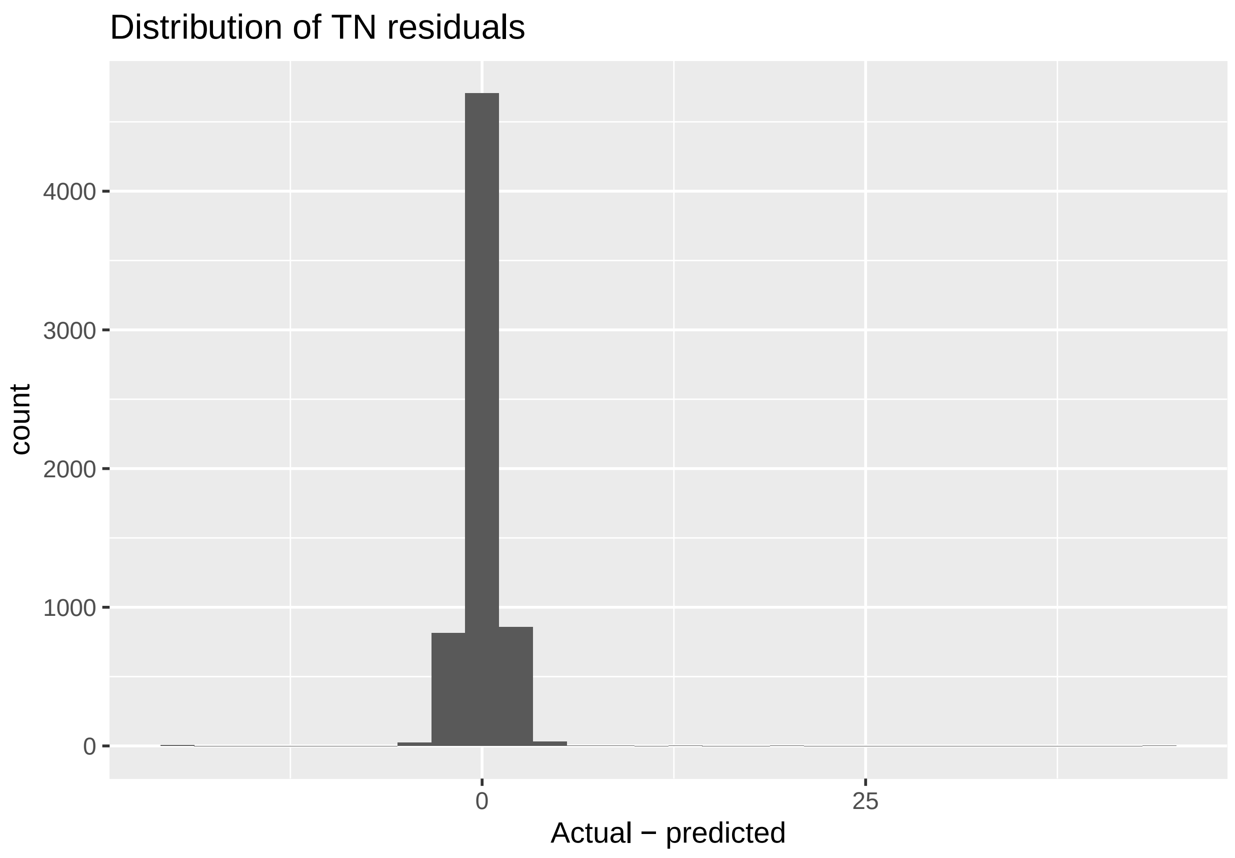
```r
# Plots
ggplot(data = tn_test, aes(x = TN, y = predicted)) +
  geom_point(alpha = 0.25) +
  geom_abline() +
  labs(title = "Predicted versus actual TN values")
```



```r
ggplot(data = rbind(data.frame(TN = c(tn_test$TN), TN_type = "actual"),
                    data.frame(TN = c(tn_test$predicted), TN_type = "predicted"))) +
  geom_boxplot(aes(x = TN, y = TN_type, group = TN_type)) +
  labs(title = "Distribution of predicted and actual TN values")
```



```
ggplot(data = tn_test, aes(x = TN - predicted)) +
  geom_histogram(bins = 30) +
  labs(title = "Distribution of TN residuals", x = "Actual - predicted")
```



```
tn_train_imp <- impute_data(tn_train, "TN", tn_train)
```

Predicting with imputed data

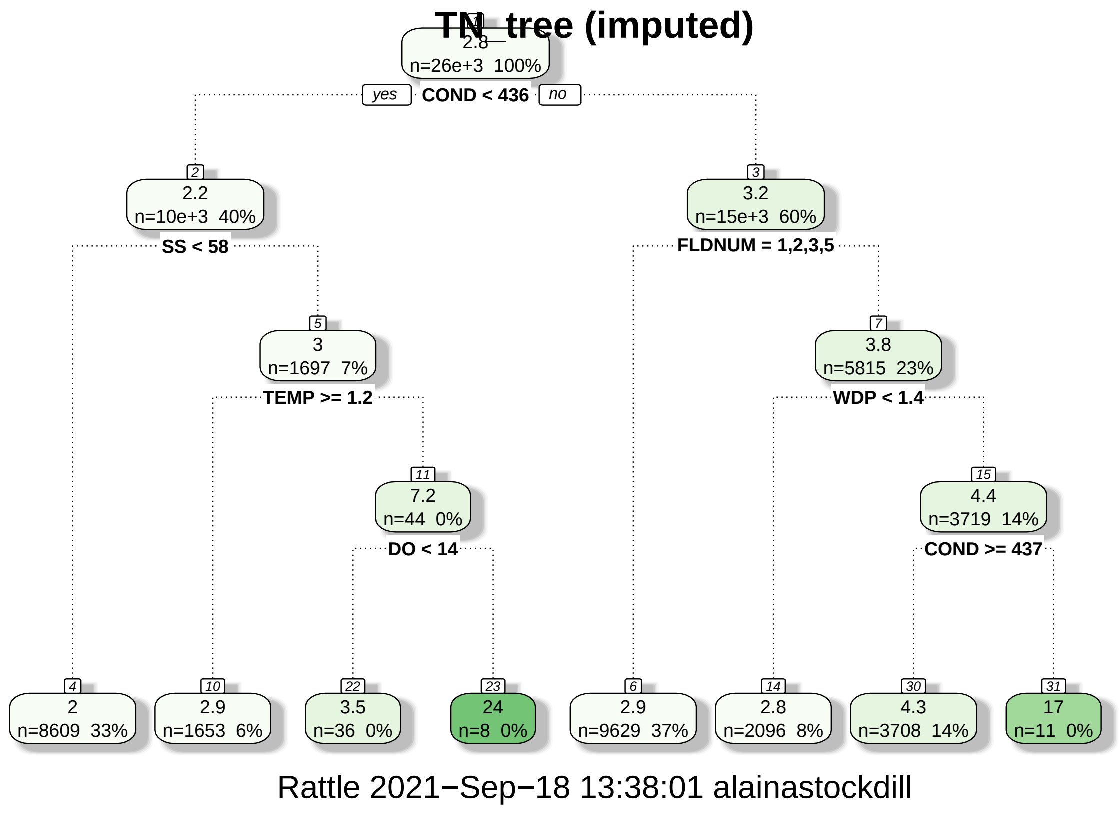
```
## [1] "TP imputation is 0.163"
## [1] "TURB imputation is 21"
## [1] "COND imputation is 461"
## [1] "VEL imputation is 0.1"
## [1] "WDP imputation is 2.23"
## [1] "CHLcal imputation is 16.52467"
## [1] "SS imputation is 26.355"
## [1] "TEMP imputation is 14.6"
## [1] "SECCHI imputation is 41"
## [1] "DO imputation is 9.6"
```

```
tn_test_imp <- impute_data(tn_test, "TN", tn_test)
```

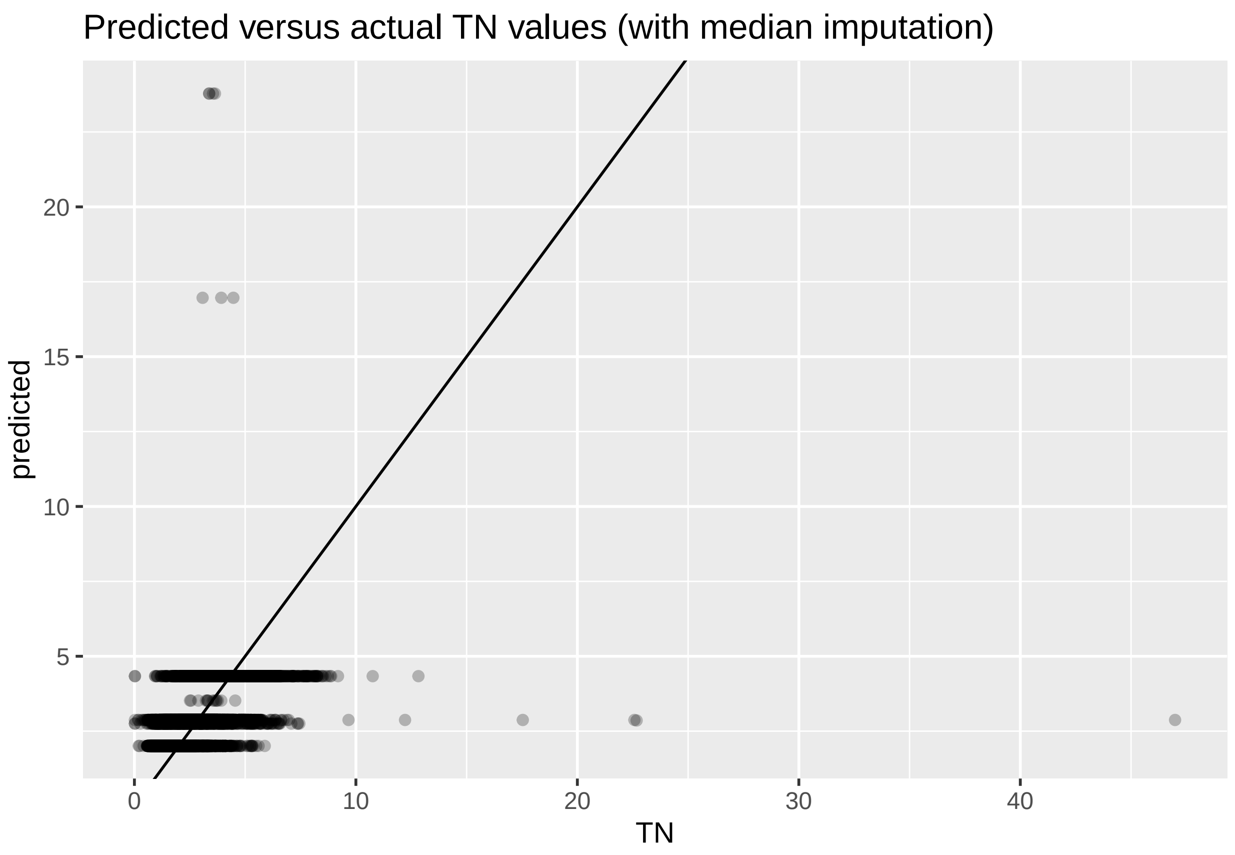
```
## [1] "TP imputation is 0.163"
## [1] "TURB imputation is 21"
## [1] "COND imputation is 460"
## [1] "VEL imputation is 0.1"
## [1] "WDP imputation is 2.2"
## [1] "CHLcal imputation is 16.3806"
## [1] "SS imputation is 26.6"
## [1] "TEMP imputation is 14.8"
## [1] "SECCHI imputation is 40"
## [1] "DO imputation is 9.8"
## [1] "predicted imputation is 2.64513018058984"
```

```
tr.TN_imp_rpart <- rpart(TN ~ .,
                         data = tn_train_imp,
                         method = "anova",
                         control = rpart.control(cp = .011, usesurrogate = 0) )
```

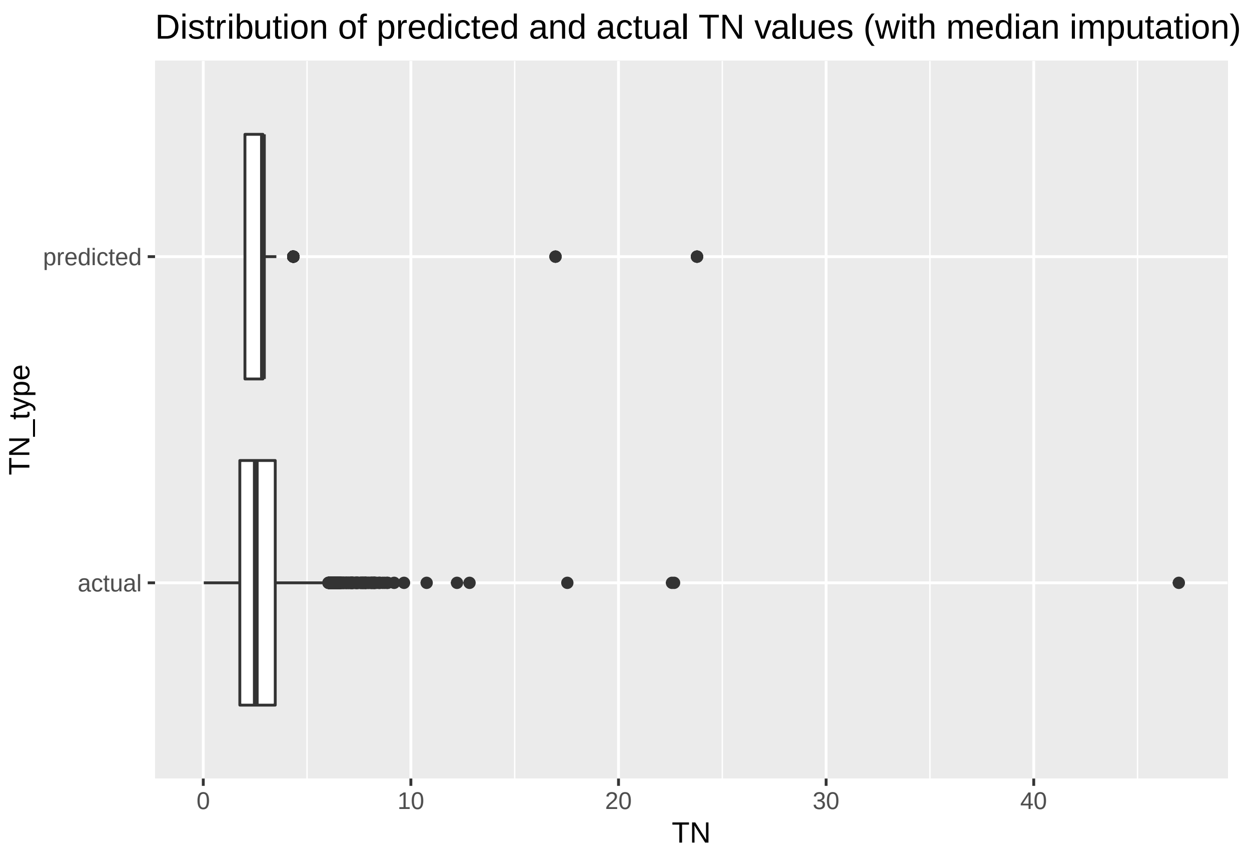
```
fancyRpartPlot(tr.TN_imp_rpart, main = "TN_tree (imputed)")
```



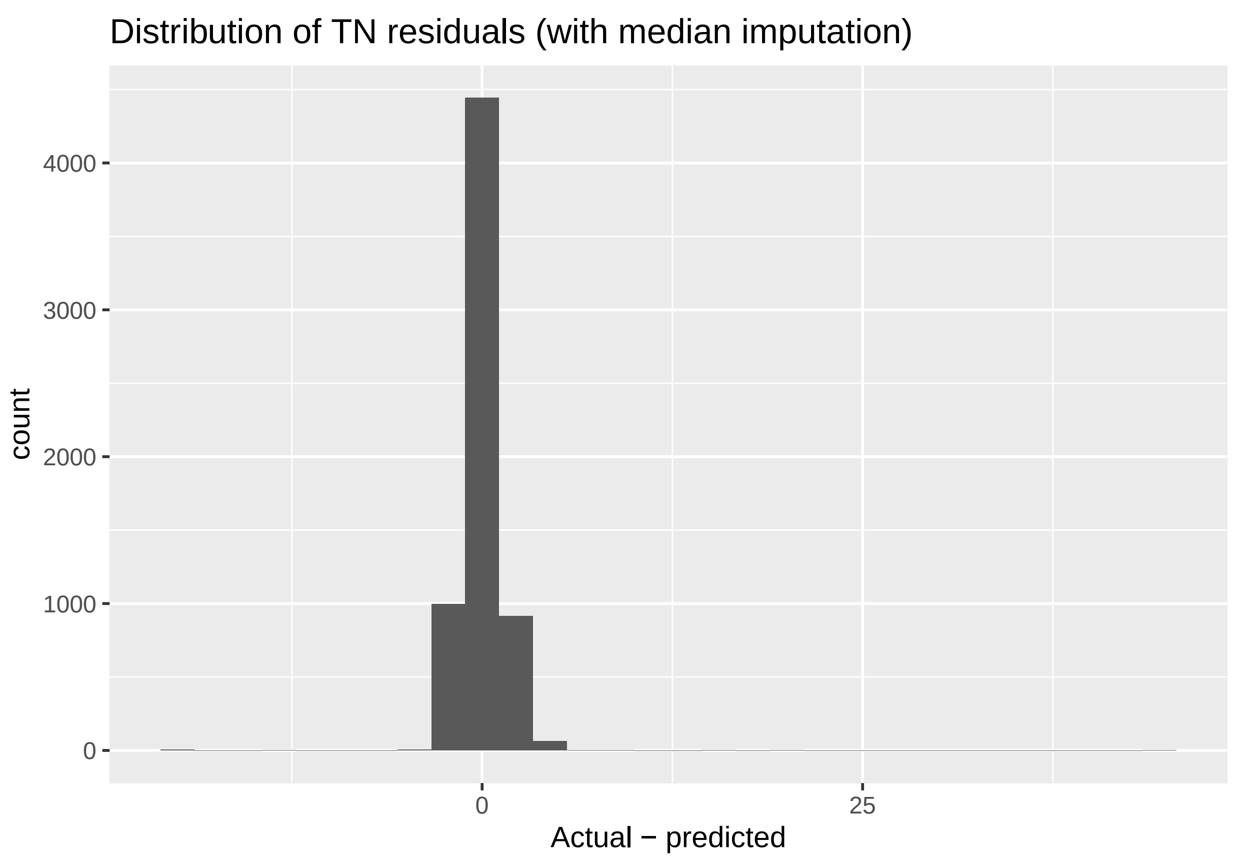
```
tn_test_imp$predicted <- predict(tr.TN_imp_rpart, tn_test_imp)

# Summary stats
error_df_imp <- add_errors(error_df_imp, "TN", tn_test_imp)
distribution_df_imp <- add_distribution(distribution_df_imp, "TN", tn_test_imp)


# Plots
ggplot(data = tn_test_imp, aes(x = TN, y = predicted)) +
  geom_point(alpha = 0.25) +
  geom_abline() +
  labs(title = "Predicted versus actual TN values (with median imputation)")
```



```
ggplot(data = rbind(data.frame(TN = c(tn_test_imp$TN), TN_type = "actual"),
                    data.frame(TN = c(tn_test_imp$predicted), TN_type = "predicted"))) +
  geom_boxplot(aes(x = TN, y = TN_type, group = TN_type)) +
  labs(title = "Distribution of predicted and actual TN values (with median imputation)")
```



```
ggplot(data = tn_test_imp, aes(x = TN - predicted)) +
  geom_histogram(bins = 30) +
  labs(title = "Distribution of TN residuals (with median imputation)", x = "Actual - predicted")
```

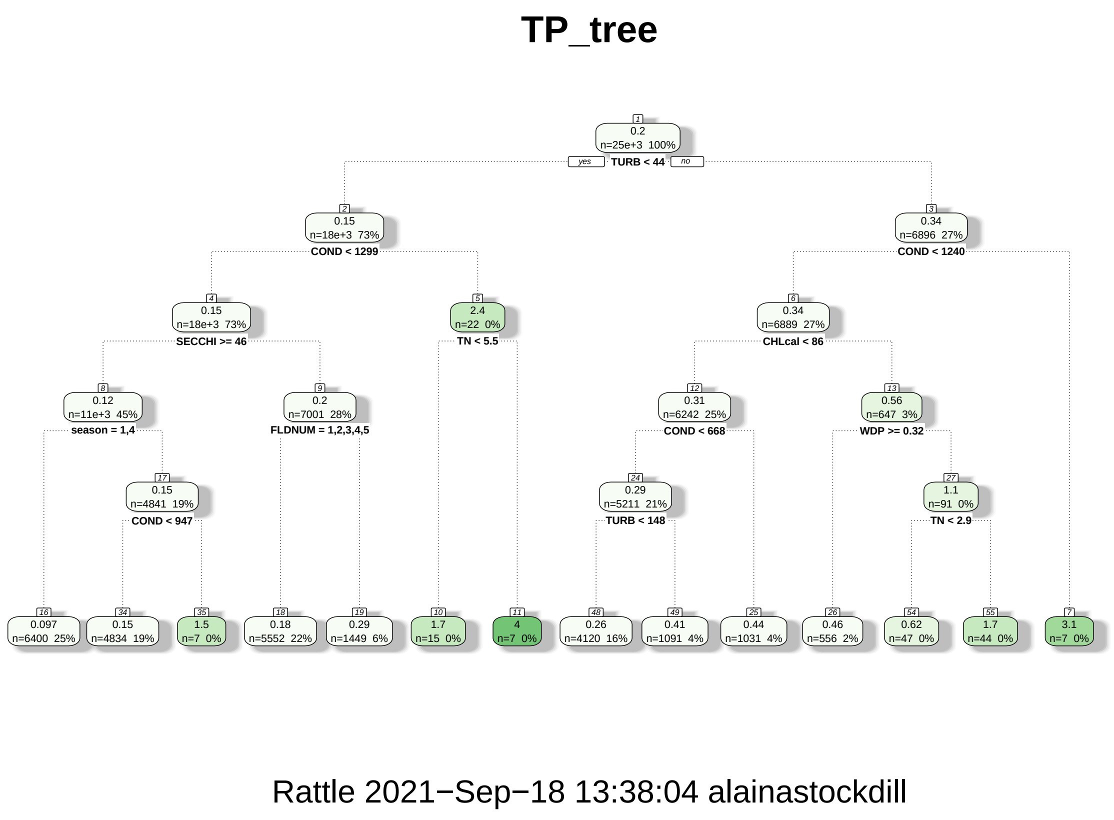


TP model

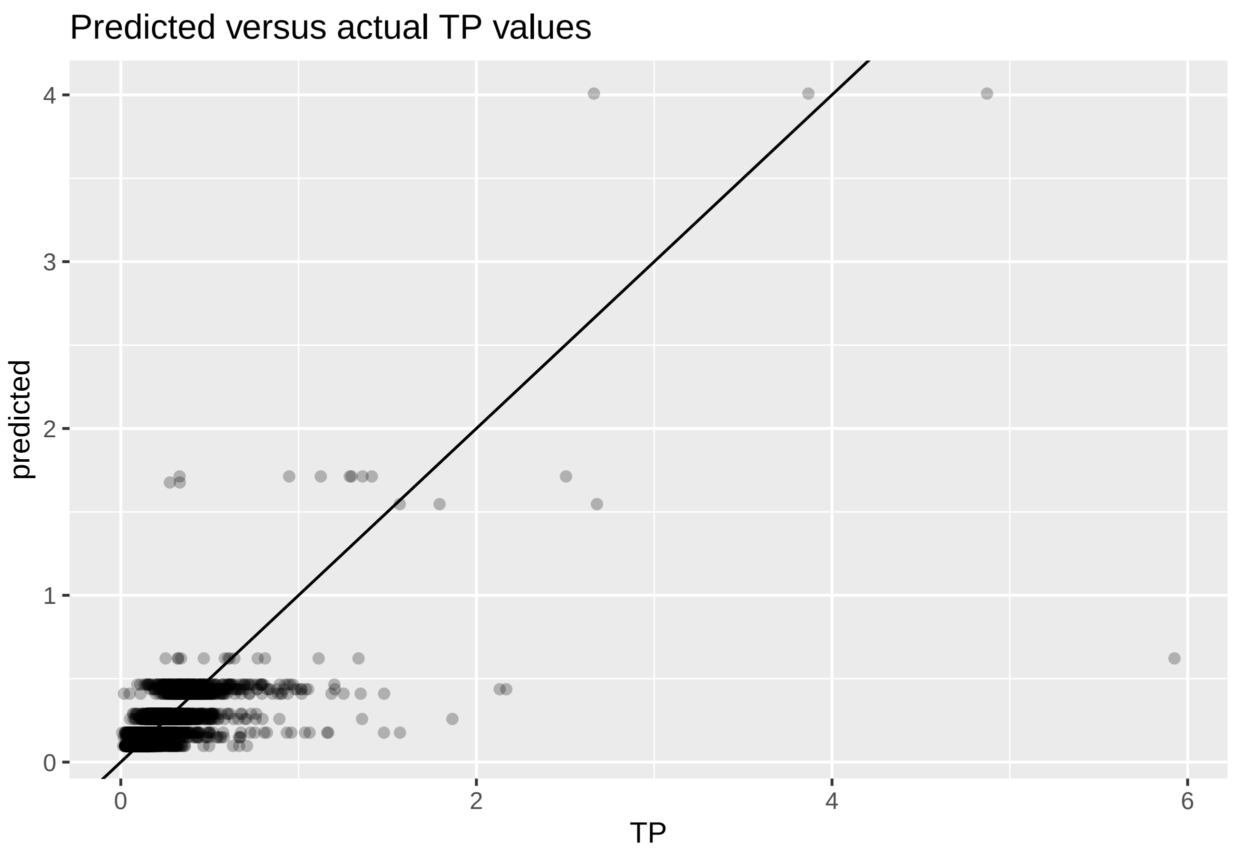
```
#Create a new data set that removes only the na TN values
 water_TP = rt_data %>% filter(!is.na(TP))

set.seed(571)
sample_size = 0.80 * nrow(water_TP)
train_indices <- sample(seq_len(nrow(water_TP)), size = sample_size)

tp_train <- water_TP[train_indices, ]
tp_test <- water_TP[-train_indices, ]

tr.TP_rpart <- rpart(TP ~ .,
                     data = tp_train,
                     method = "anova",
                     control = rpart.control(cp = .011) )
```

Predicting with surrogate splits

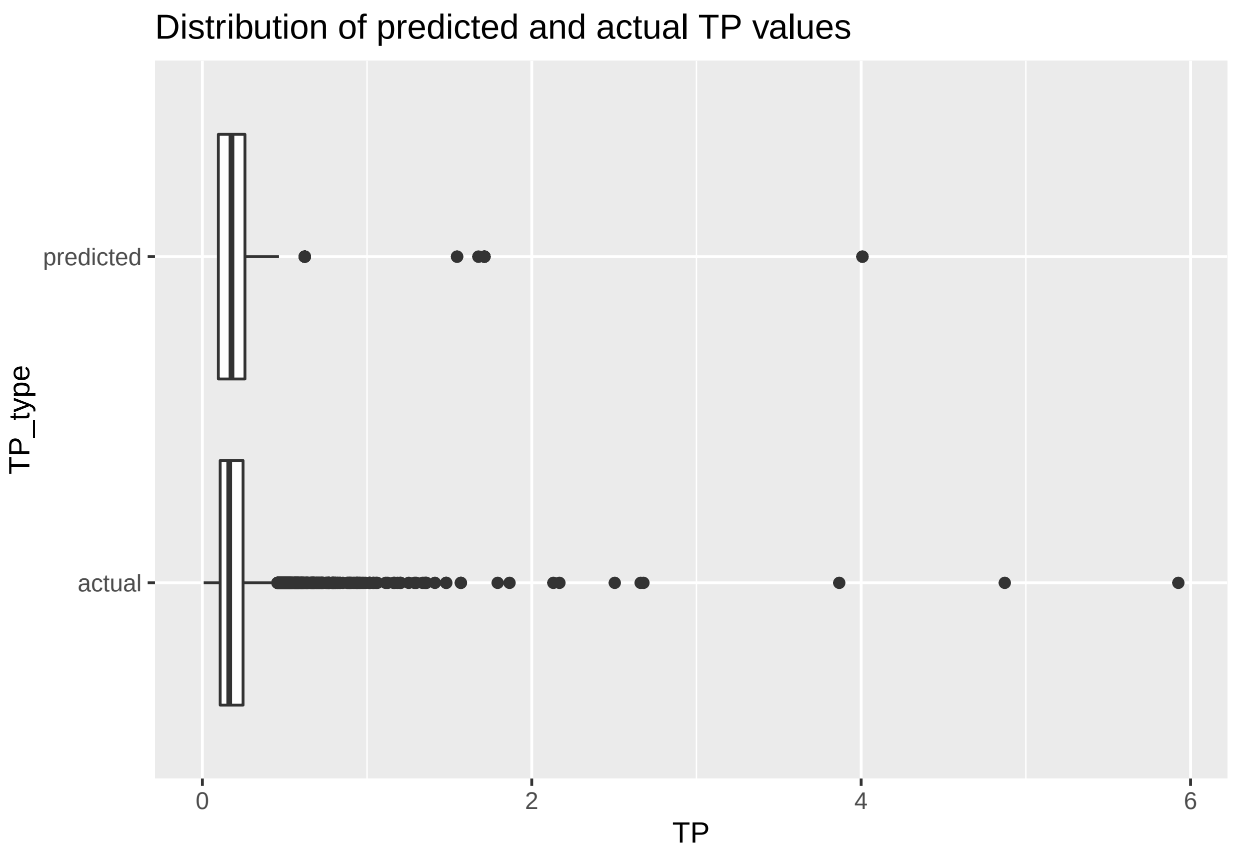
```
fancyRpartPlot(tr.TP_rpart, main = "TP_tree")
```



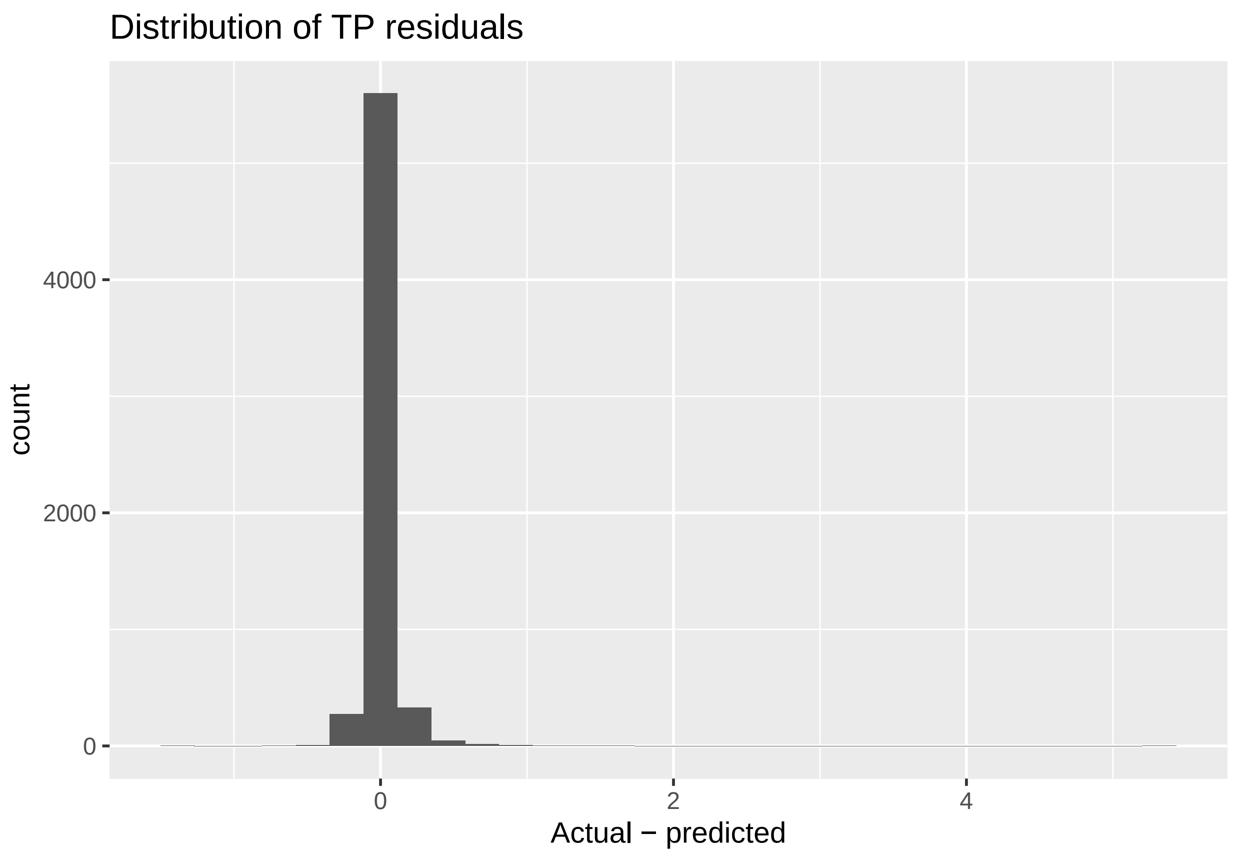
```
tp_test$predicted <- predict(tr.TP_rpart, tp_test)

# Summary stats
error_df <- add_errors(error_df, "TP", tp_test)
distribution_df <- add_distribution(distribution_df, "TP", tp_test)


# Plots
ggplot(data = tp_test, aes(x = TP, y = predicted)) +
  geom_point(alpha = 0.25) +
  geom_abline() +
  labs(title = "Predicted versus actual TP values")
```



```
ggplot(data = rbind(data.frame(TP = c(tp_test$TP), TP_type = "actual"),
                    data.frame(TP = c(tp_test$predicted), TP_type = "predicted"))) +
  geom_boxplot(aes(x = TP, y = TP_type, group = TP_type)) +
  labs(title = "Distribution of predicted and actual TP values")
```



```
ggplot(data = tp_test, aes(x = TP - predicted)) +
  geom_histogram(bins = 30) +
  labs(title = "Distribution of TP residuals", x = "Actual - predicted")
```



```
tp_train_imp <- impute_data(tp_train, "TP", tp_train)
```

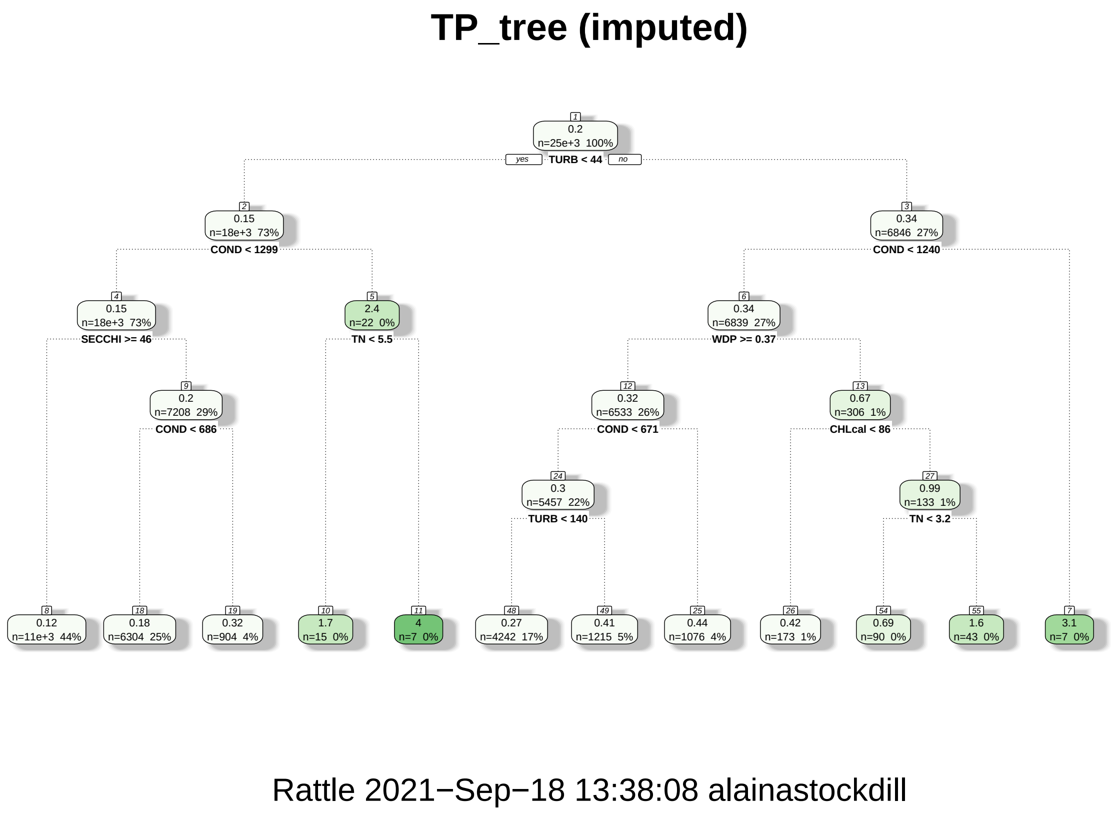
Predicting with imputed data

```
## [1] "TN imputation is 2.535"
## [1] "TURB imputation is 21"
## [1] "COND imputation is 461"
## [1] "VEL imputation is 0.1"
## [1] "WDP imputation is 2.23"
## [1] "CHLcal imputation is 16.39806"
## [1] "SS imputation is 26"
## [1] "TEMP imputation is 14.5"
## [1] "SECCHI imputation is 42"
## [1] "DO imputation is 9.7"
```

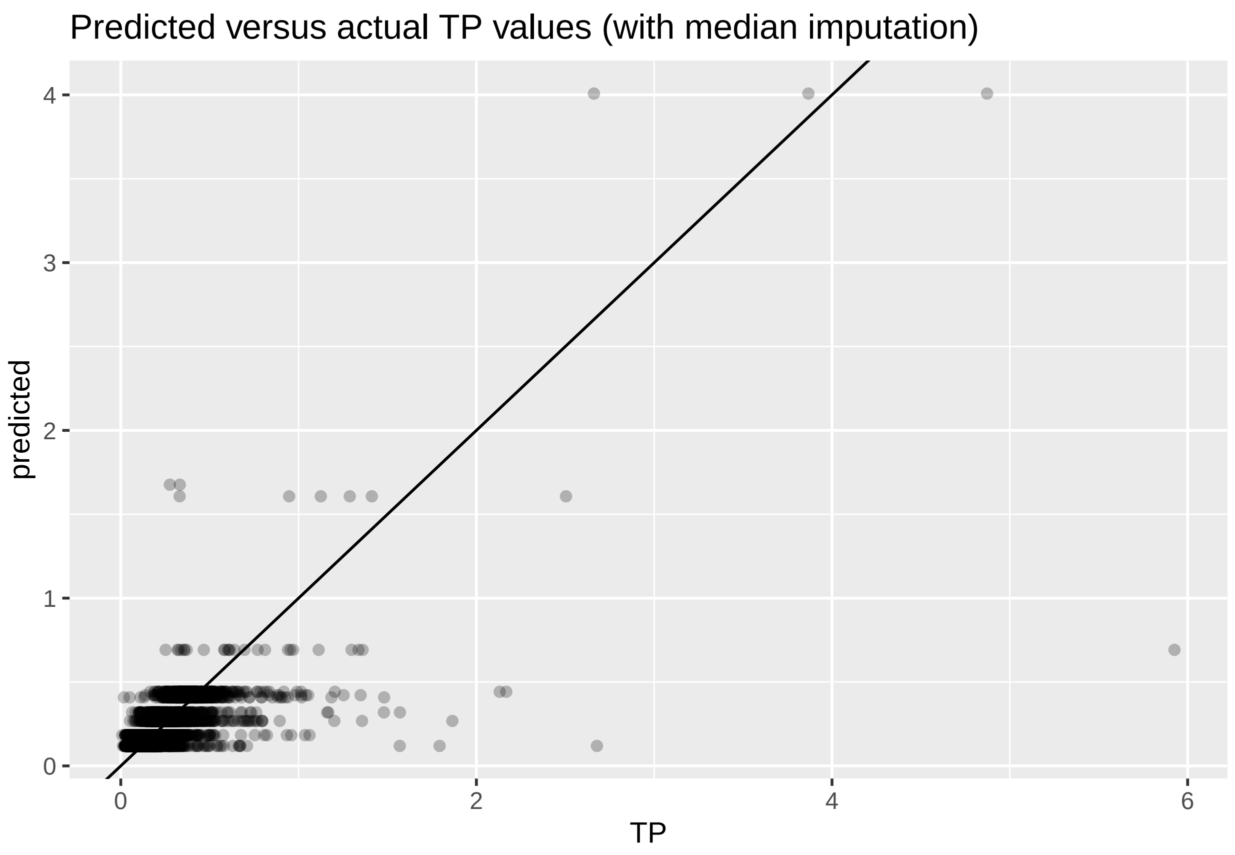
```
tp_test_imp <- impute_data(tp_test, "TP", tp_test)
```

```
## [1] "TN imputation is 2.566"
## [1] "TURB imputation is 21"
## [1] "COND imputation is 461"
## [1] "VEL imputation is 0.1"
## [1] "WDP imputation is 2.2"
## [1] "CHLcal imputation is 16.27266"
## [1] "SS imputation is 26.4"
## [1] "TEMP imputation is 14.6"
## [1] "SECCHI imputation is 42"
## [1] "DO imputation is 9.7"
## [1] "predicted imputation is 0.17667417146974"
```

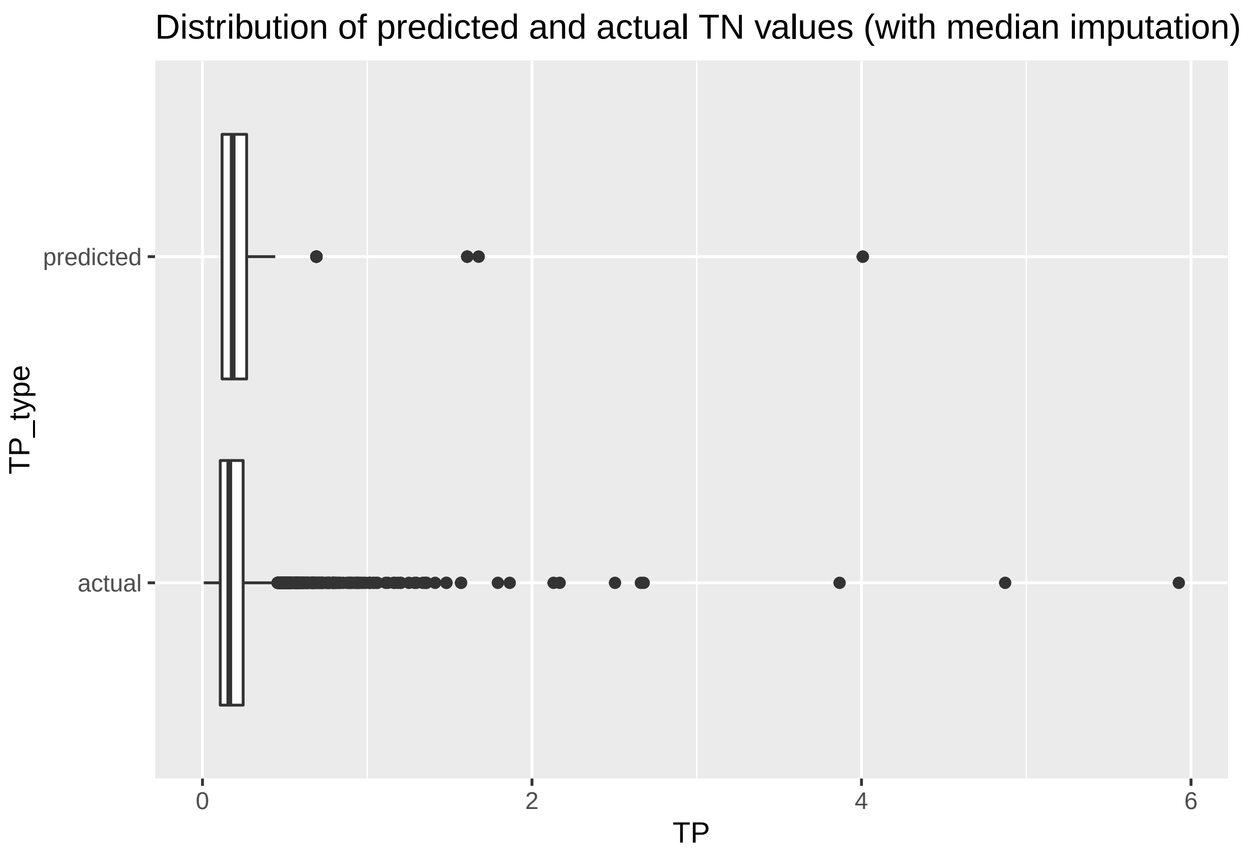
```
tr.TP_imp_rpart <- rpart(TP ~ .,
                    data = tp_train_imp,
                    method = "anova",
                    control = rpart.control(cp = .011, usesurrogate = 0) )

fancyRpartPlot(tr.TP_imp_rpart, main = "TP_tree (imputed)")
```



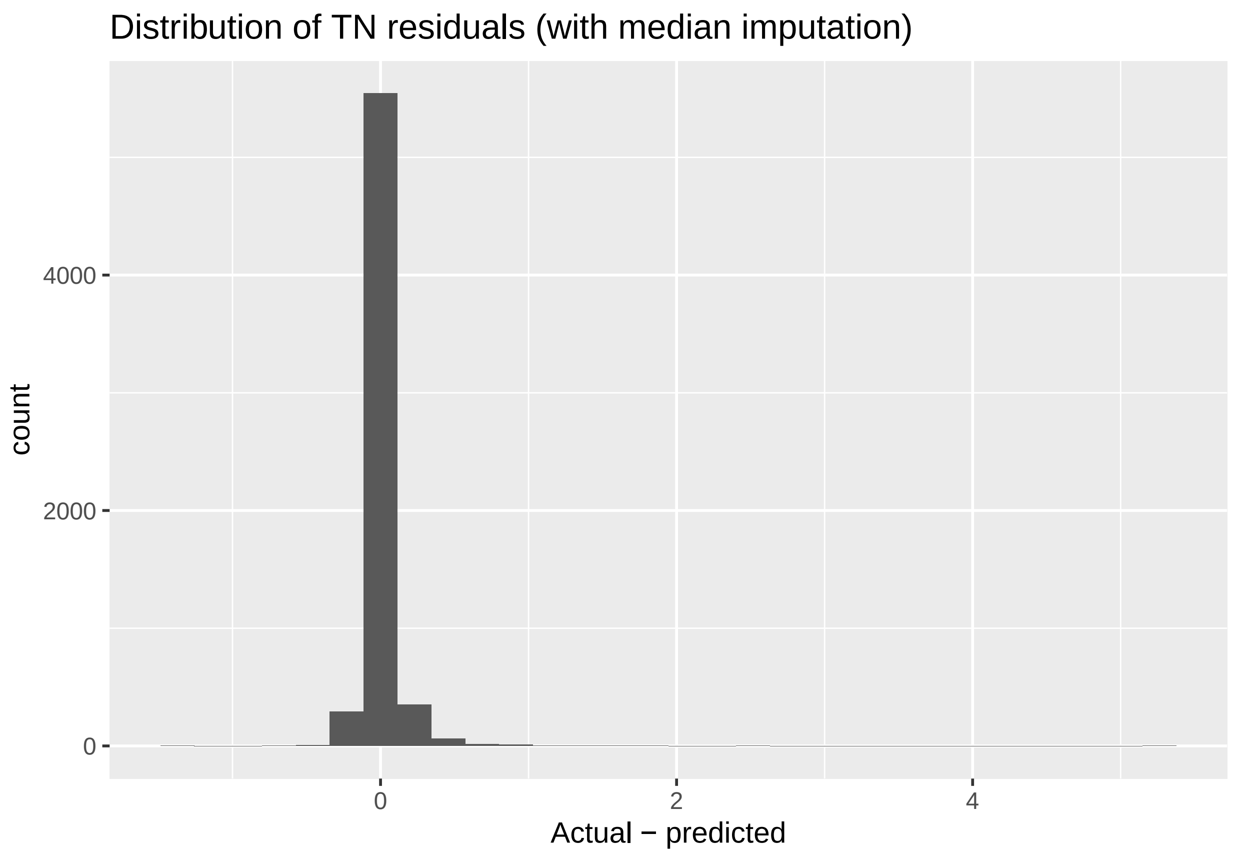
```
tp_test_imp$predicted <- predict(tr.TP_imp_rpart, tp_test_imp)

# Summary stats
error_df_imp <- add_errors(error_df_imp, "TP", tp_test_imp)
distribution_df_imp <- add_distribution(distribution_df_imp, "TP", tp_test_imp)


# Plots
ggplot(data = tp_test_imp, aes(x = TP, y = predicted)) +
  geom_point(alpha = 0.25) +
  geom_abline() +
  labs(title = "Predicted versus actual TP values (with median imputation)")
```



```r
ggplot(data = rbind(data.frame(TP = c(tp_test_imp$TP), TP_type = "actual"),
                    data.frame(TP = c(tp_test_imp$predicted), TP_type = "predicted"))) +
  geom_boxplot(aes(x = TP, y = TP_type, group = TP_type)) +
  labs(title = "Distribution of predicted and actual TN values (with median imputation)")
```



```
ggplot(data = tp_test_imp, aes(x = TP - predicted)) +
  geom_histogram(bins = 30) +
  labs(title = "Distribution of TN residuals (with median imputation)", x = "Actual - predicted")
```

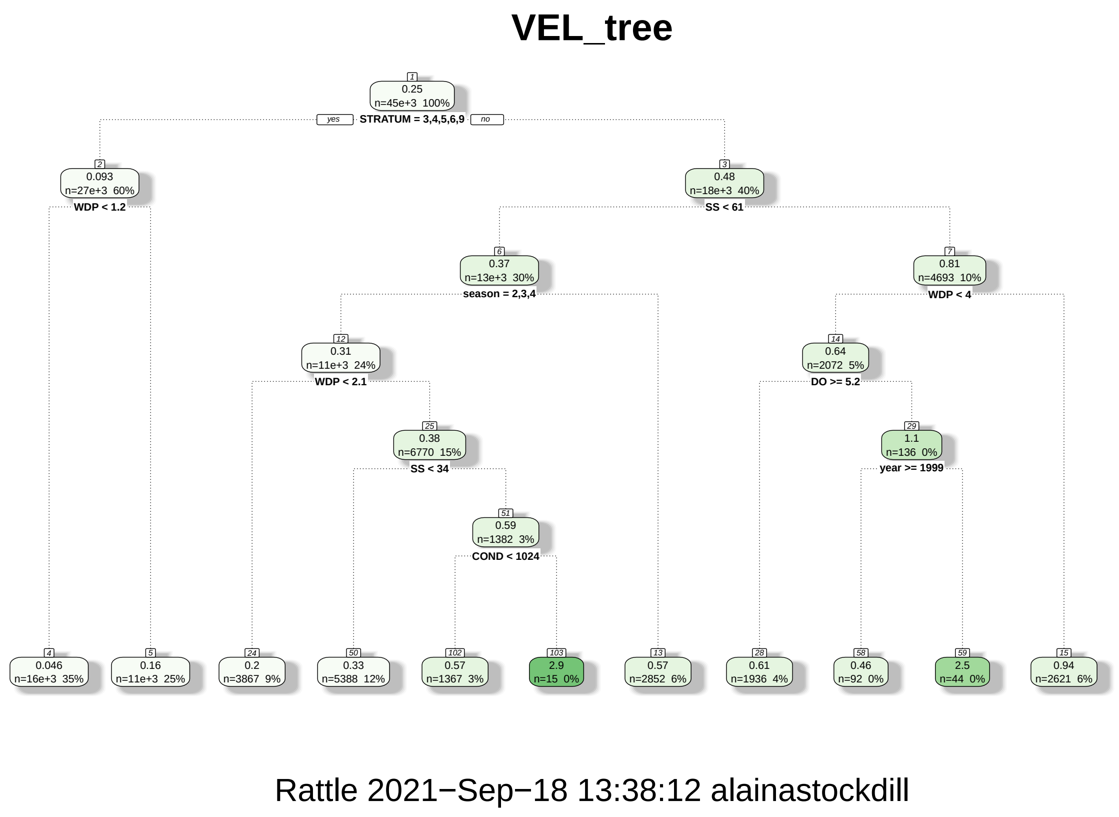


VEL model

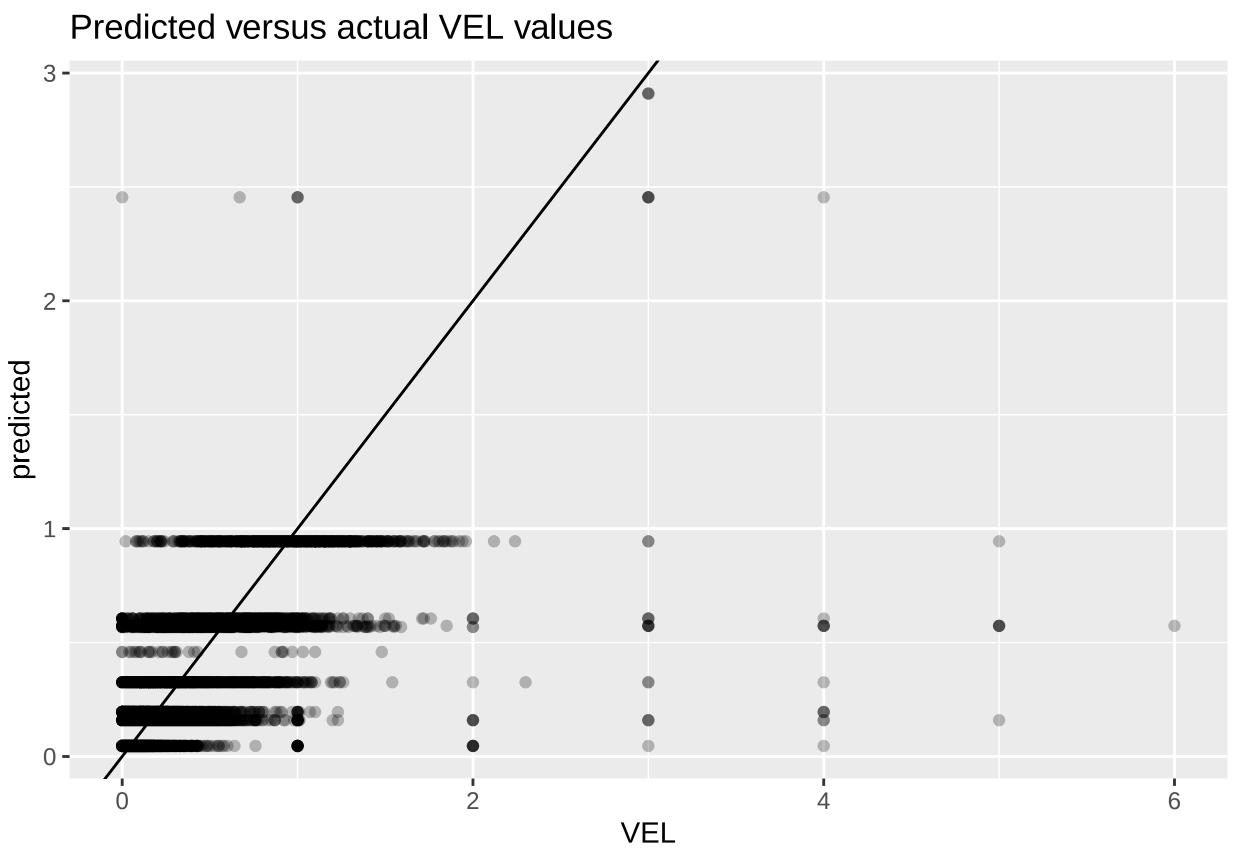
```
#Create a new data set that removes only the na TN values
 water_VEL = rt_data %>% filter(!is.na(VEL))

sample_size = 0.80 * nrow(water_VEL)
train_indices <- sample(seq_len(nrow(water_VEL)), size = sample_size)

vel_train <- water_VEL[train_indices, ]
vel_test <- water_VEL[-train_indices, ]

tr.VEL_rpart <- rpart(VEL ~ .,
                     data = vel_train,
                     method = "anova",
                     control = rpart.control(cp = .011) )
```

Predicting with surrogate splits

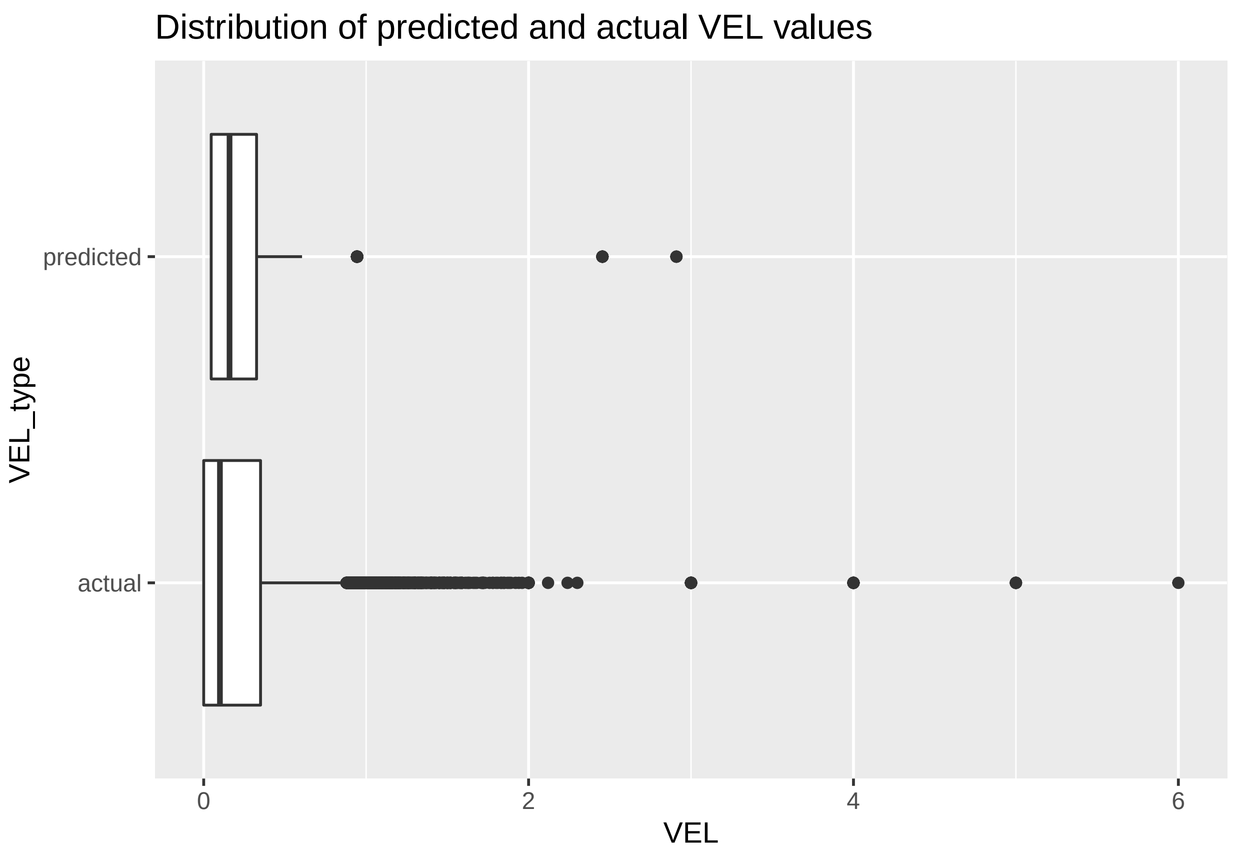
```r
fancyRpartPlot(tr.VEL_rpart, main = "VEL_tree")
```



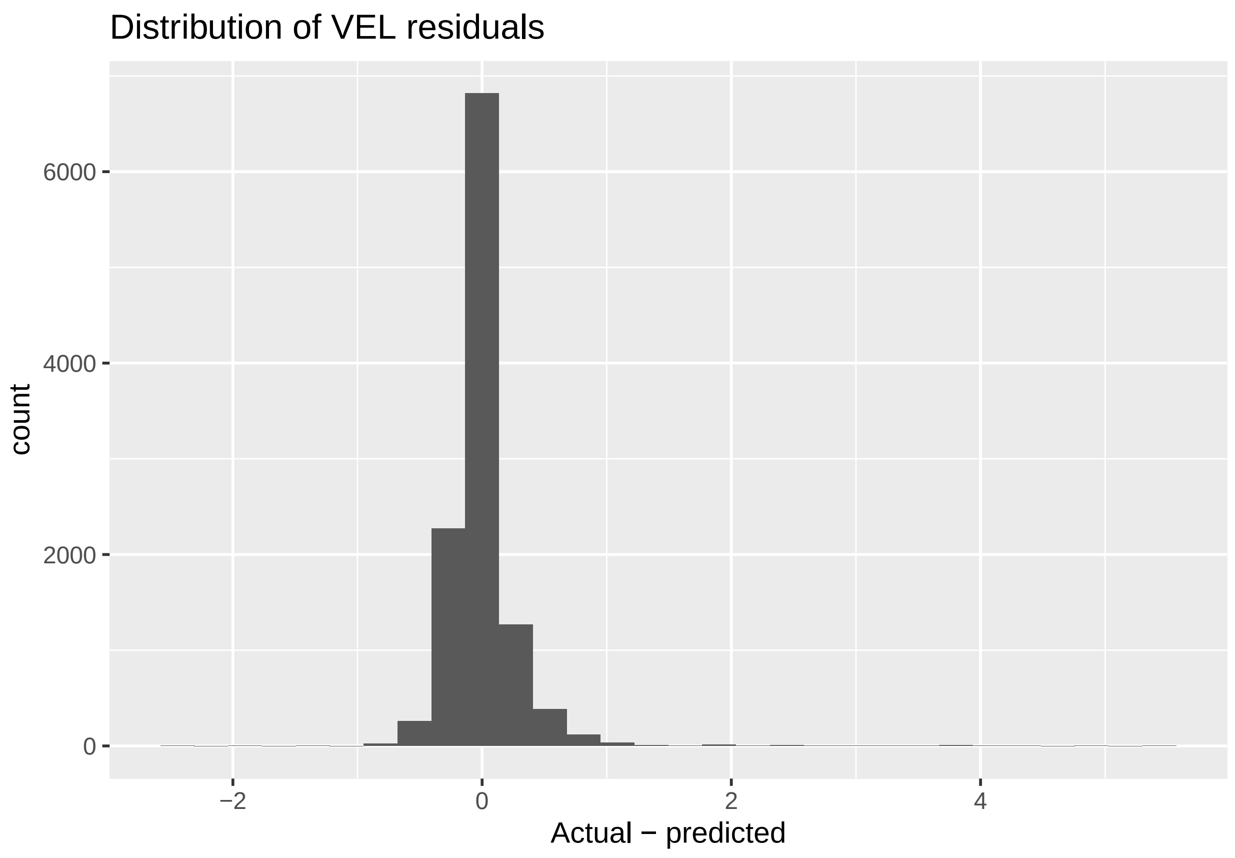
```r
vel_test$predicted <- predict(tr.VEL_rpart, vel_test)

# Summary stats
error_df <- add_errors(error_df, "VEL", vel_test)
distribution_df <- add_distribution(distribution_df, "VEL", vel_test)


# Plots
ggplot(data = vel_test, aes(x = VEL, y = predicted)) +
  geom_point(alpha = 0.25) +
  geom_abline() +
  labs(title = "Predicted versus actual VEL values")
```



```
ggplot(data = rbind(data.frame(VEL = c(vel_test$VEL), VEL_type = "actual"),
                    data.frame(VEL = c(vel_test$predicted), VEL_type = "predicted"))) +
  geom_boxplot(aes(x = VEL, y = VEL_type, group = VEL_type)) +
  labs(title = "Distribution of predicted and actual VEL values")
```



```r
ggplot(data = vel_test, aes(x = VEL - predicted)) +
  geom_histogram(bins = 30) +
  labs(title = "Distribution of VEL residuals", x = "Actual - predicted")
```



```
vel_train_imp <- impute_data(vel_train, "VEL", vel_train)
```

Predicting with imputed data

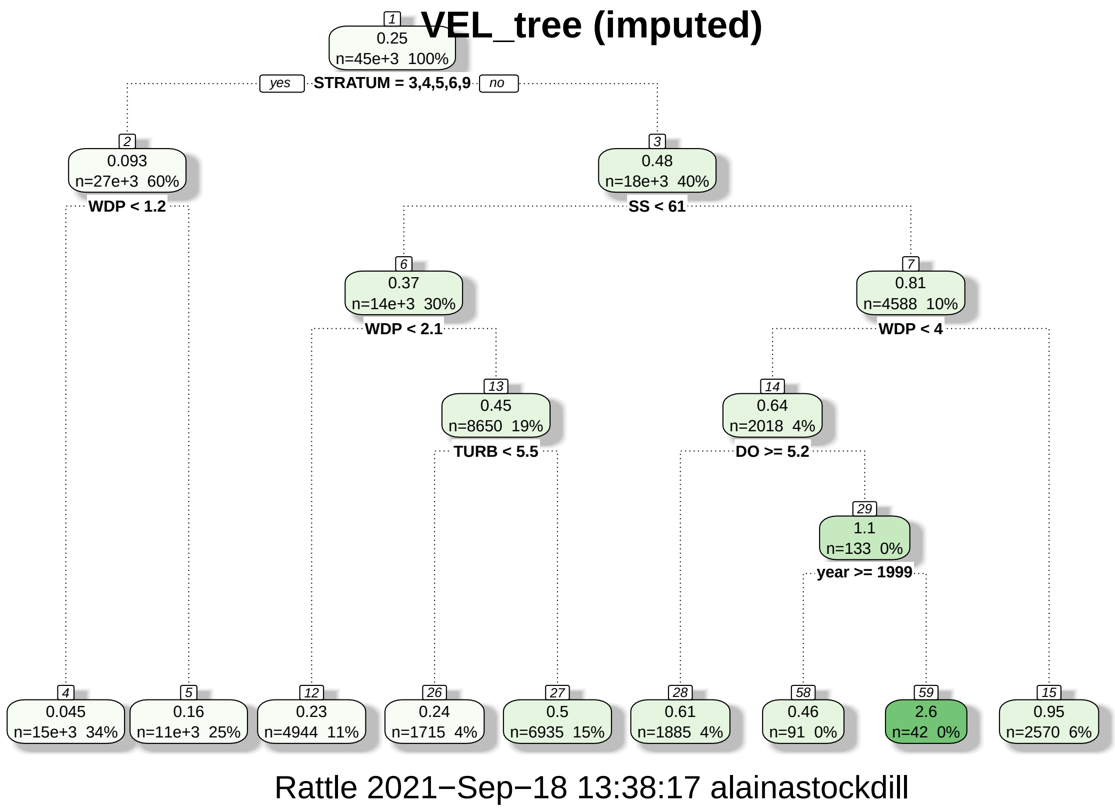
```
## [1] "TP imputation is 0.15"
## [1] "TN imputation is 2.3895"
## [1] "TURB imputation is 16"
## [1] "COND imputation is 443"
## [1] "WDP imputation is 1.5"
## [1] "CHLcal imputation is 16.54293"
## [1] "SS imputation is 20.4"
## [1] "TEMP imputation is 14.4"
## [1] "SECCHI imputation is 46"
## [1] "DO imputation is 9.9"
```

```
vel_test_imp <- impute_data(vel_test, "VEL", vel_test)
```

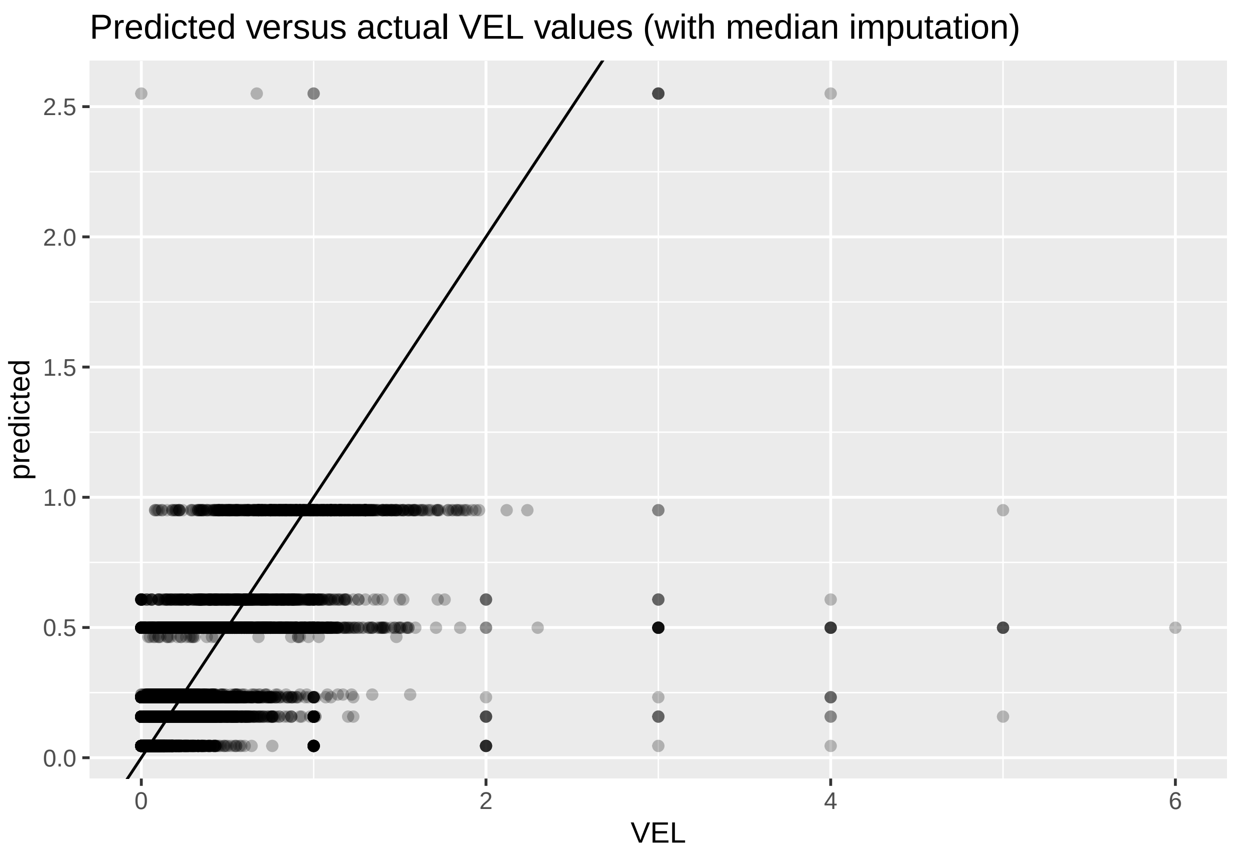
```
## [1] "TP imputation is 0.148"
## [1] "TN imputation is 2.375"
## [1] "TURB imputation is 16"
## [1] "COND imputation is 443"
## [1] "WDP imputation is 1.5"
## [1] "CHLcal imputation is 16.59902"
## [1] "SS imputation is 20"
## [1] "TEMP imputation is 14.4"
## [1] "SECCHI imputation is 46"
## [1] "DO imputation is 9.9"
## [1] "predicted imputation is 0.159098642533932"
```

```
tr.VEL_imp_rpart <- rpart(VEL ~ .,
                     data = vel_train_imp,
                     method = "anova",
                     control = rpart.control(cp = .011, usesurrogate = 0) )
```

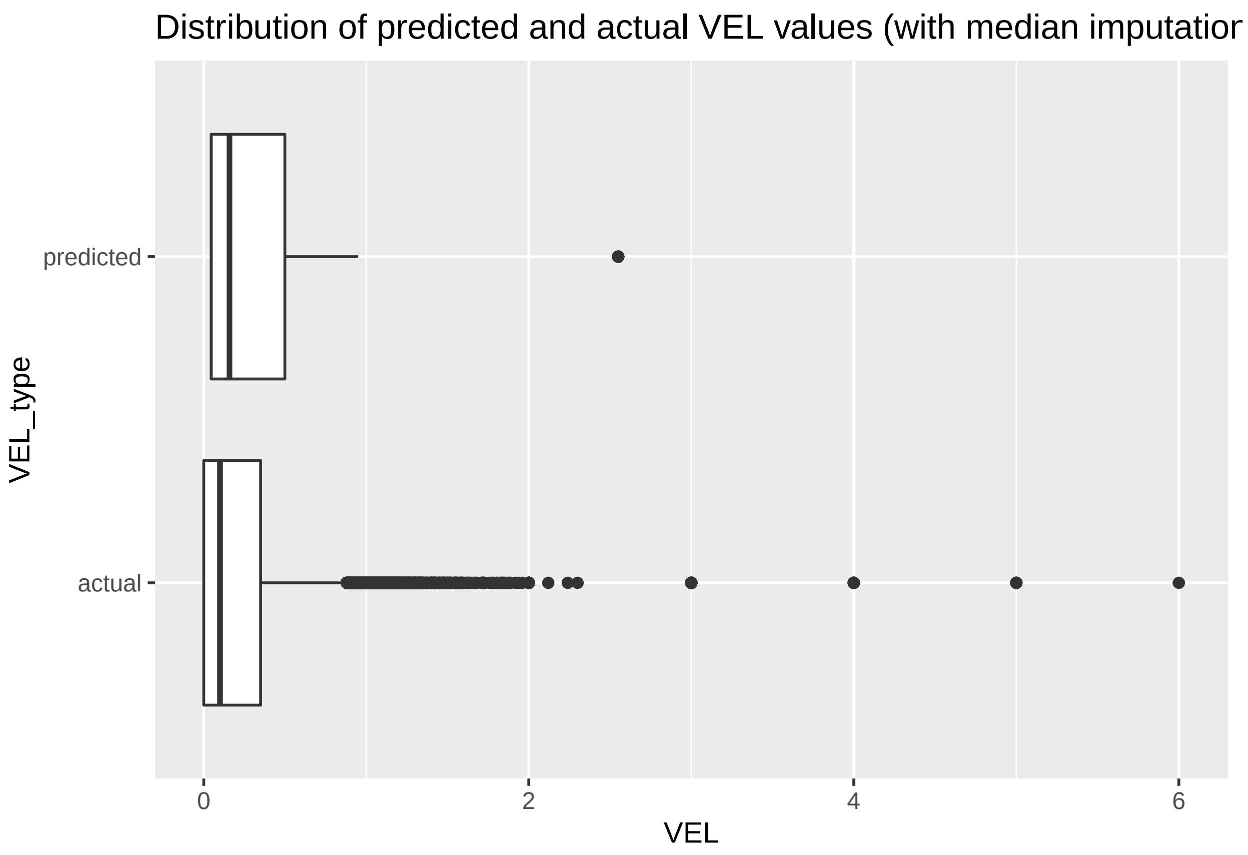
```
fancyRpartPlot(tr.VEL_imp_rpart, main = "VEL_tree (imputed)")
```



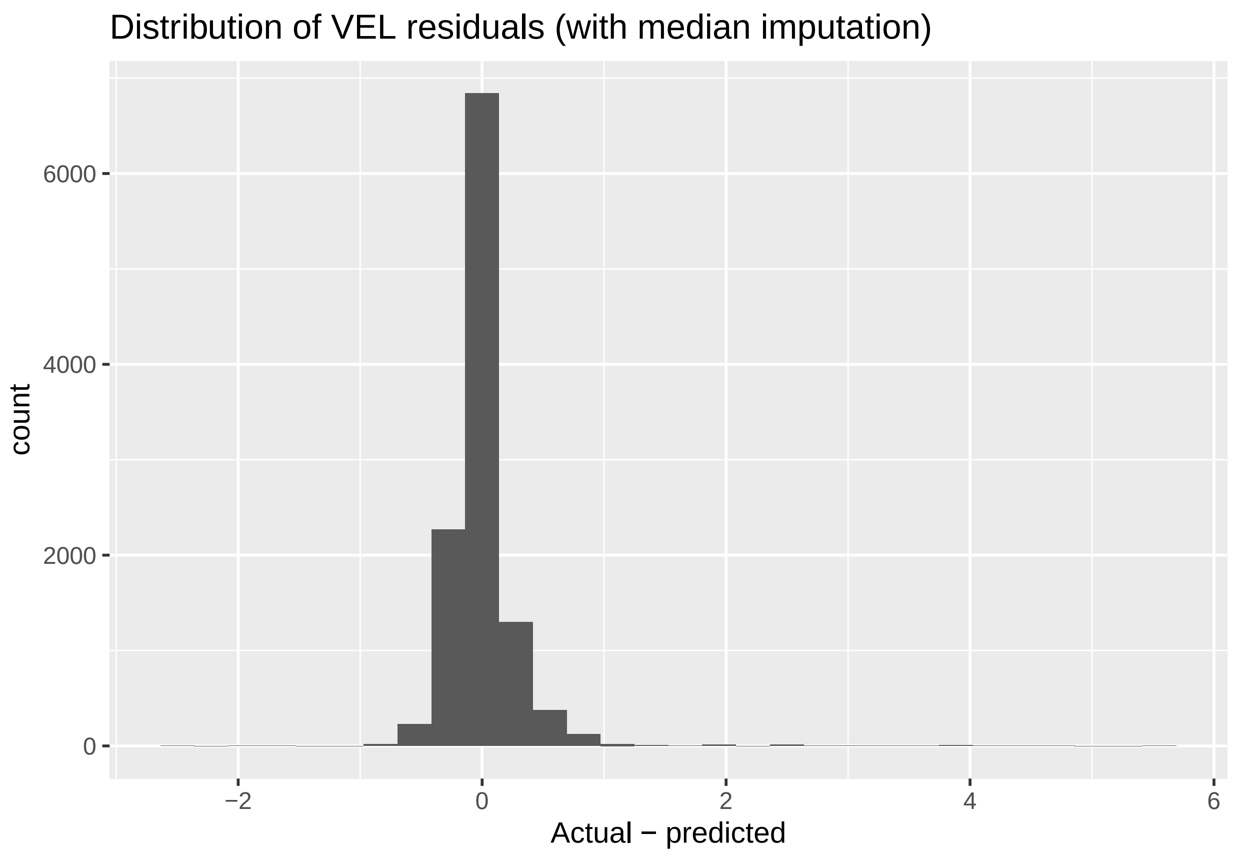
```
vel_test_imp$predicted <- predict(tr.VEL_imp_rpart, vel_test_imp)

# Summary stats
error_df_imp <- add_errors(error_df_imp, "VEL", vel_test_imp)
distribution_df_imp <- add_distribution(distribution_df_imp, "VEL", vel_test_imp)


# Plots
ggplot(data = vel_test_imp, aes(x = VEL, y = predicted)) +
  geom_point(alpha = 0.25) +
  geom_abline() +
  labs(title = "Predicted versus actual VEL values (with median imputation)")
```



```
ggplot(data = rbind(data.frame(VEL = c(vel_test_imp$VEL), VEL_type = "actual"),
                    data.frame(VEL = c(vel_test_imp$predicted), VEL_type = "predicted"))) +
  geom_boxplot(aes(x = VEL, y = VEL_type, group = VEL_type)) +
  labs(title = "Distribution of predicted and actual VEL values (with median imputation)")
```



```r
ggplot(data = vel_test_imp, aes(x = VEL - predicted)) +
  geom_histogram(bins = 30) +
  labs(title = "Distribution of VEL residuals (with median imputation)", x = "Actual - predicted")
```



Results

After creating the models for each method, we will print the summary stats. These stats will be used to compare with the other interpolation methods we are testing. We can also determine if our model is better when the surrogate splits are used or when median imputation is used on the missing values. We conclude that the regression tree model is more accurate when surrogate splits are used.

```
print("Errors and summary stats of regression trees with surrogate splits")
```

```
## [1] "Errors and summary stats of regression trees with surrogate splits"
```

```
print(error_df)
```

```
##   Response    RSME     MAE Test.data.size
## 1       TN 1.37897 0.86778           6438
## 2       TP 0.13187 0.06106           6290
## 3      VEL 0.29690 0.15741          11236
```

```
print(distribution_df)
```

```
##      Response    Mean Minimum Quartile.1  Median Quartile.3  Maximum
## 25%        TN 2.82098 2.00432    2.00432 2.64513    2.64513 23.78200
## 25%1       TP 0.20397 0.09697    0.09697 0.17667    0.17667  4.00786
## 25%2      VEL 0.24922 0.04616    0.04616 0.15910    0.15910  2.91000
```

```
print("Errors and summary stats of regression trees with median imputation")
```

```
## [1] "Errors and summary stats of regression trees with median imputation"
```

```
print (error_df_imp)
```

```
##   Response    RSME     MAE Test.data.size
## 1       TN 1.46551 0.92896           6438
## 2       TP 0.13893 0.06574           6290
## 3      VEL 0.30112 0.15957          11236
```

```
print(distribution_df_imp)
```

```
##      Response    Mean Minimum Quartile.1  Median Quartile.3  Maximum
## 25%        TN 2.82063 2.00703    2.00703 2.87333    2.87333 23.78200
## 25%1       TP 0.20286 0.11905    0.11905 0.18328    0.18328  4.00786
## 25%2      VEL 0.24899 0.04526    0.04526 0.15788    0.15788  2.55024
```

Functions

This portion of the code will be used in order to run multiple model predictions at once.

- rmna_row: creates a subset of the entire data set that has a complete column of the variable we are currently estimating
- get_train_indices: creates a train/test split to test on our model
- make_trees: creates a regression tree to predict a target variable - saves the tree output (#https://stackoverflow.com/questions/7500219/reading-rpart-input-parameters-from-a-text-variable)
- rt_evalution: takes a target variable with a data and uses the three previous functions to output a testing data with a predicted column for that variable - incorporates the creation of a train/test split
- make_plots: makes an actual versus predicted scatter plot and box plots for each actual and predicted column of the current target variable - will save both plots
- get_model_stats: depending on the errorsBool, this function will either return a list containing the MAE and RMSE of actual versus predicted values or return a dataframe that contains the summary stats (Min, Q1, Mean, Q3, Max) for both the actual and predicted column

```
rmna_rows <- function(target_var, df) {
  df <- df %>% filter_at(vars(target_var), any_vars(!is.na(.)))
  return(df)
}

get_train_indices <- function(df) {
  sample_size = 0.80 * nrow(df)
  set.seed(4321)
  train_indices <- sample(seq_len(nrow(df)), size = sample_size)

  return(train_indices)

}

make_trees <-function(target_var, df) {

  rpart_formula <- as.formula(paste(target_var, " ~ ."))

  model_controls <- rpart.control(cp = .011)
  tr_rpart <- rpart(rpart_formula,
                    data = df,
                    method = "anova",
                      control = model_controls )

  tree_title = paste("tree_",target_var, sep = "")
  png_name = paste(tree_title, ".png", sep = "")

  png(png_name)
  tree <- fancyRpartPlot(tr_rpart, main = tree_title)
  dev.off()

  return(tr_rpart)

}


rt_evaluation  <- function(target_var, df) {
  df <- rmna_rows(target_var, df)

  indices <- get_train_indices(df)

  data_train <- df[indices, ]
  data_test <- df[-indices, ]

  model <- make_trees(target_var, data_train)

  pred_col_name <- paste("PREDICTED_", target_var, sep = "") # New predicted column
  data_test[ , pred_col_name] <-  predict(model, data_test)

  #printcp(model)

  return(data_test)
}

make_plots <- function(data_test, target_actual, target_pred) {
    X <- data_test[ , target_actual]
    Y <- data_test[ , target_pred]

    # scatter plot
    png_name = paste(target_actual, "_predvactual.png", sep = "")
    ggplot(data_test, mapping = aes(x = X, y = Y)) +
      geom_point(alpha = .02) +
      coord_equal() +
      theme(aspect.ratio = 1) + xlim(0, 4) + ylim(0, 4) +
      labs( y = target_pred, x = target_actual)
    ggsave(png_name)


}


get_model_stats <- function(data_test, errorBool) {

  data_test <- data.frame(data_test) # make sure data_test is a dataframe
```

```r
  # get the target variable
  target_pred  <- names(select(data_test, contains("PREDICTED_")))  # column name
  target_actual <- sub("PREDICTED_", "", target_pred)

  if( errorBool == TRUE )  {

    errors <- c(MAE(data_test[ , target_actual], data_test[ , target_pred]),
                RMSE(data_test[ , target_actual], data_test[ ,target_pred]))

    #errors <- errors %>% rename(errors = target_actual)

    return(errors)

  } else {

    actual_sum <- summary( data_test[ , target_actual] )
    predicted_sum <- summary( data_test[ , target_pred] )

    actual <- c(actual_sum[1], actual_sum[2], actual_sum[3],
                 actual_sum[4], actual_sum[5], actual_sum[6])

    predicted <-c(predicted_sum[1], predicted_sum[2], predicted_sum[3],
                   predicted_sum[4], predicted_sum[5], predicted_sum[6])

    model_summary <- data.frame(actual, predicted)

    names(model_summary)[1] <- target_actual
    names(model_summary)[2] <- target_pred

    # actual v pred and boxplots
    make_plots(data_test, target_actual, target_pred)

    return(model_summary)

  }
}
```

```r
#water_var_preds <- lapply( water_vars, rt_evaluation, rt_data )
#water_var_errors <- lapply( water_var_preds, get_model_stats, TRUE )
#water_var_summary <- lapply( water_var_preds, get_model_stats, FALSE )

# combine columns
#type <- c("MAE", "RMSE")

# Renames the columns of the error data frame
#TP <- water_var_errors[[1]]
#TN <- water_var_errors[[2]]
#VEL <- water_var_errors[[3]]
#model_errors <- data.frame( type, TP, TN, VEL)

# Renames the columns of the summary stat data frame
#one <- water_var_summary[[1]]
#two <- water_var_summary[[2]]
#three <- water_var_summary[[3]]
```

```
#model_summary <- data.frame(one, two, three)
```

Scratch work

```
# ### COMPLETE TN COLUMN ###
# # Create a new data set that removes only the na TN values
# water_TN = model_data %>%
#   filter_at(vars(TN), any_vars(!is.na(.)))
#
#
# sample_size = 0.80 * nrow(water_TN)
# set.seed(571)
# train_indices <- sample(seq_len(nrow(water_TN)), size = sample_size)
#
# data_train <- water_TN[train_indices, ]
# data_test <- water_TN[-train_indices, ]
#
# set.seed(4321)
# model_controls <- rpart.control(max_depth = 20)
# tr.TN_rpart <- rpart(TN ~ .,
#                      data = data_train,
#                      method = "anova",
#                      control = model_controls )
# fancyRpartPlot(tr.TN_rpart, main = "TN prediction for all years")
#
# data_test$PREDICTED_TN <- predict(tr.TN_rpart, data_test)
# summary <- summary(data_test$TN)
# pred_summary <- summary(data_test$PREDICTED_TN)
#
# TN_sum <- c(summary[1], summary[2], summary[3], summary[4], summary[5], summary[6])
# TN_pred_sum <-c(pred_summary[1], pred_summary[2], pred_summary[3], pred_summary[4], pred_summary[5], p
#
# TN_sum2 <- list(unname(summary(data_test$TN)))
# TN_pred_sum2 <- list(unname(summary(data_test$PREDICTED_TN)))
#
# model_summary <- data.frame(TN_sum, TN_pred_sum)
# model_summary2 <- data.frame(TN_sum2, TN_pred_sum2)
#
# TN <- c(MAE(data_test$TN, data_test$PREDICTED), RMSE(data_test$TN, data_test$PREDICTED), mean((data_t
#
```

```
# ### RPART MODEL ###
# # Runs on the entire data set - even with na values in all columns
# set.seed(84726)
# temp_controls = rpart.control(cp = .01)
# tr.TN_rpart <- rpart(TN ~ .,
#                      data = water,
#                      method = "anova",
#                      control = temp_controls)
# fancyRpartPlot(tr.TN_rpart, main = "TN prediction: cp = .1")
# ```

# ```{r}
```

```
# # Tree evaluation summary
# plotcp(tr.TN_rpart)
# printcp(tr.TN_rpart)
# rsq.rpart(tr.TN_rpart)
# summary(tr.TN_rpart)
```