Machine Learning II - Group D - MBD16 - Competition

Instructions

Calling below function after running the RMD file will output a Data Frame with the 'id' variable and the prediction for 'status_group' including a header for the test dataset that lays on the filepath.

```
evaluate('filepath')
```

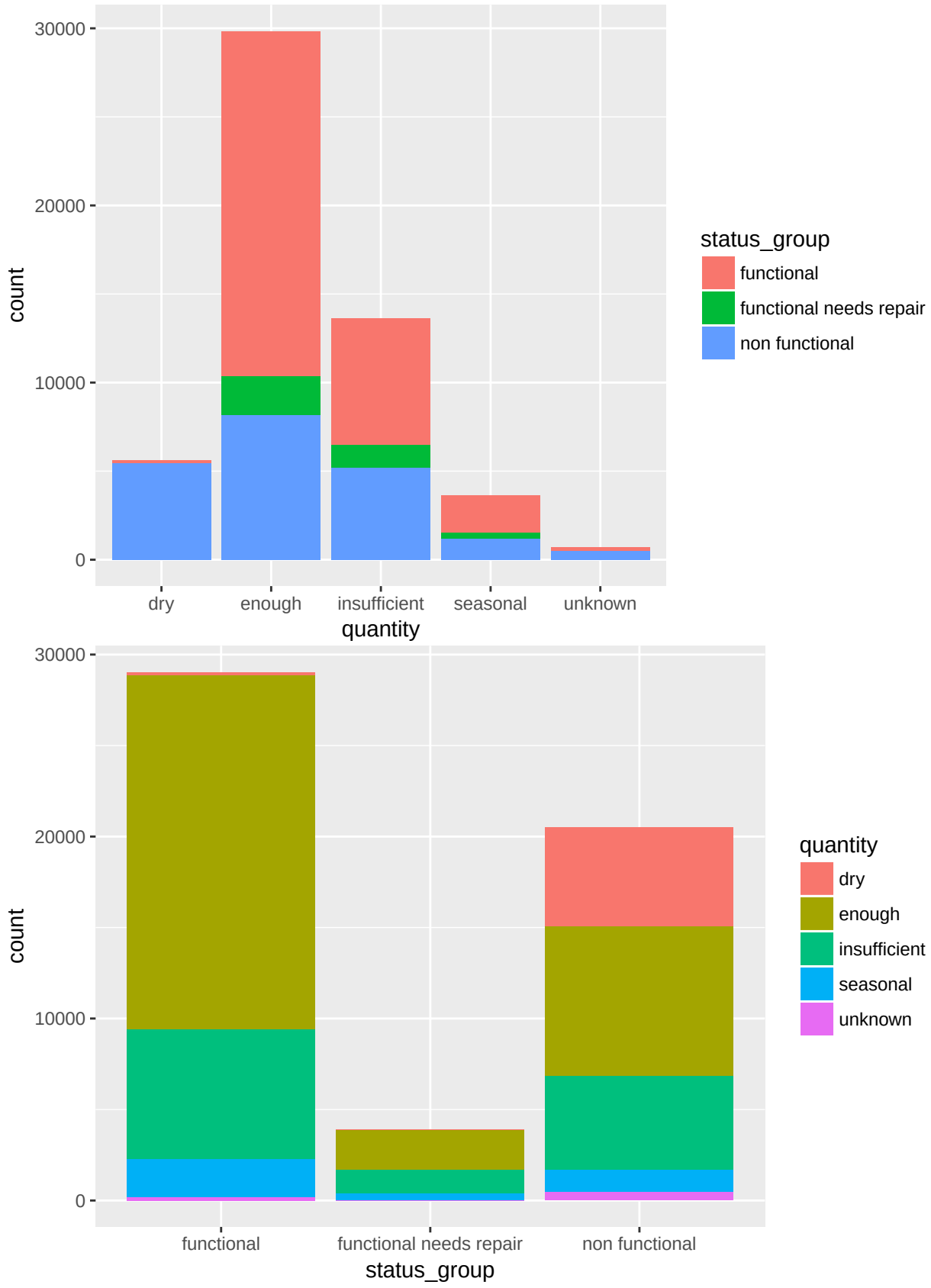
Table of content

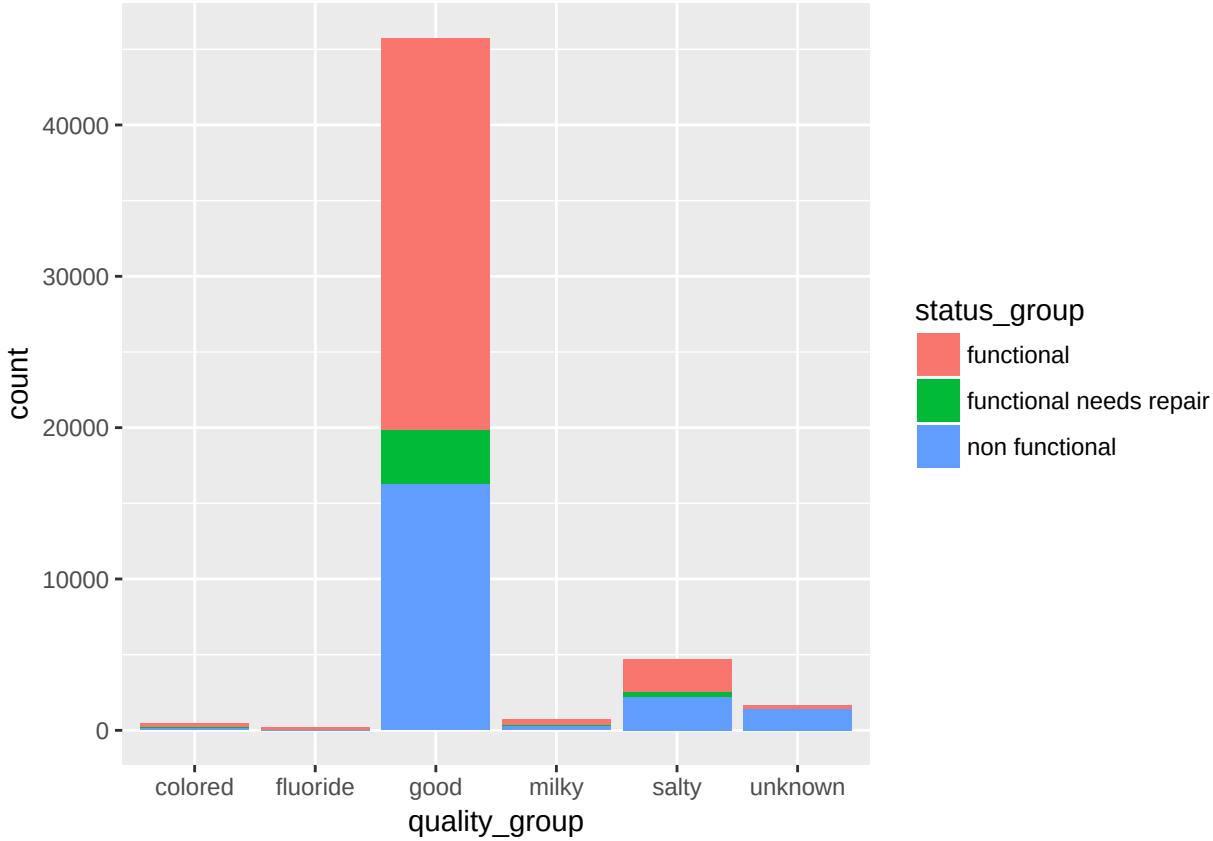
1. Data Exploration

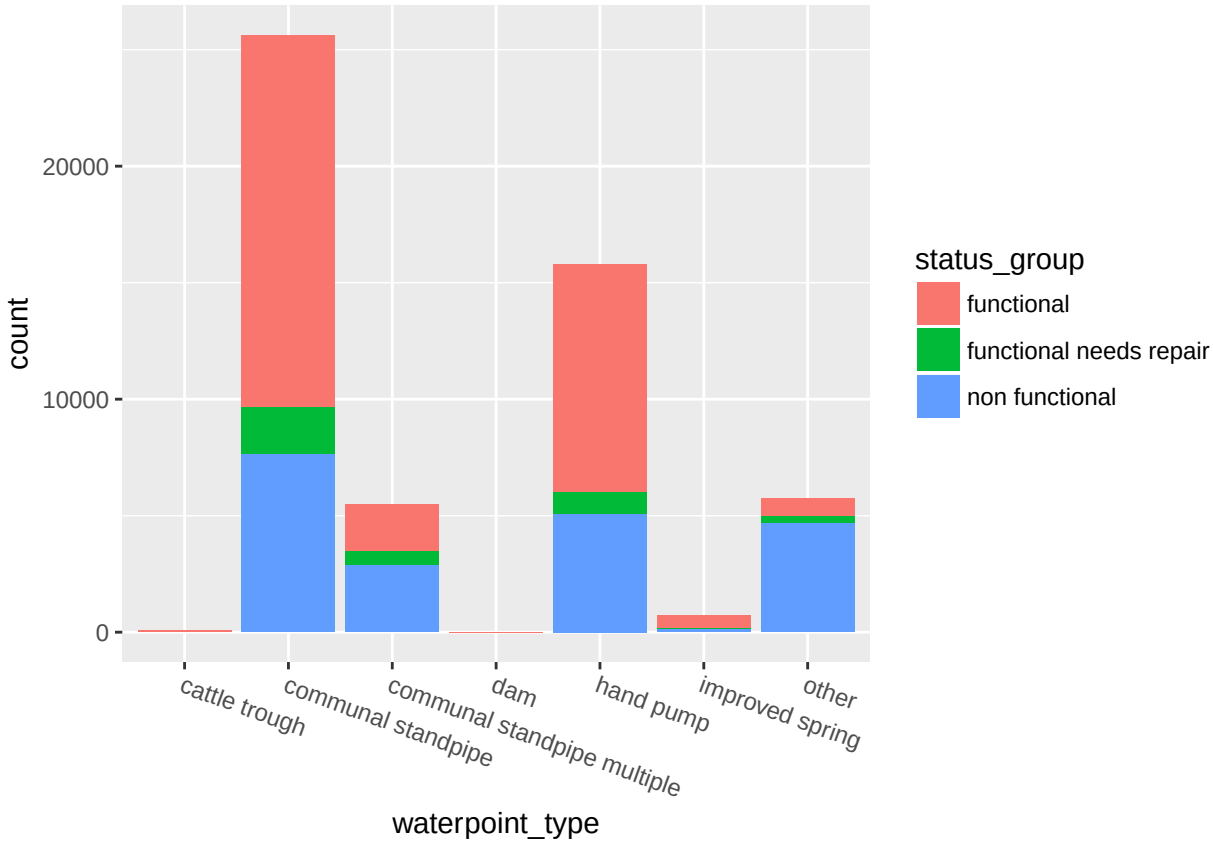
Before starting engineering the features, the dataset is explored by plotting relevant features.

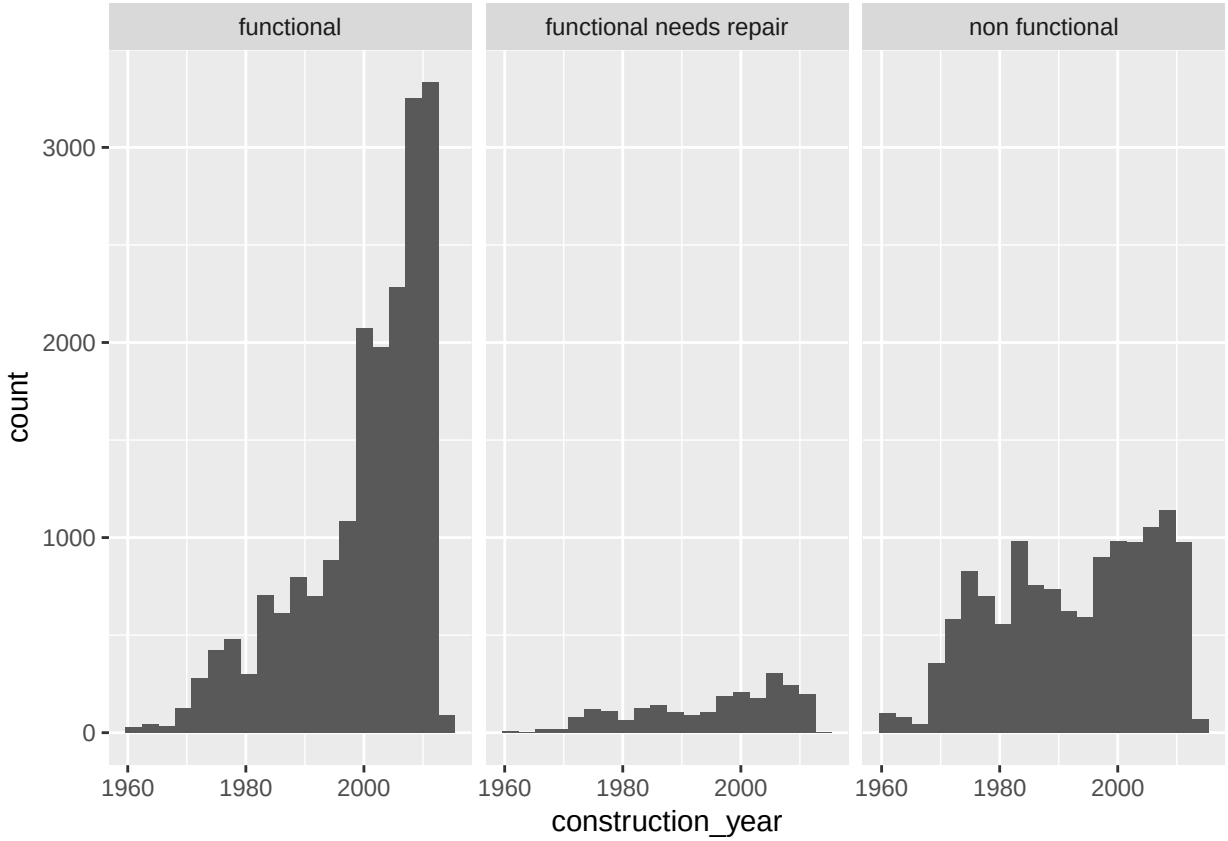
```
FALSE status_group
FALSE              functional functional needs repair          non functional
FALSE                   29034                    3926                   20500
```

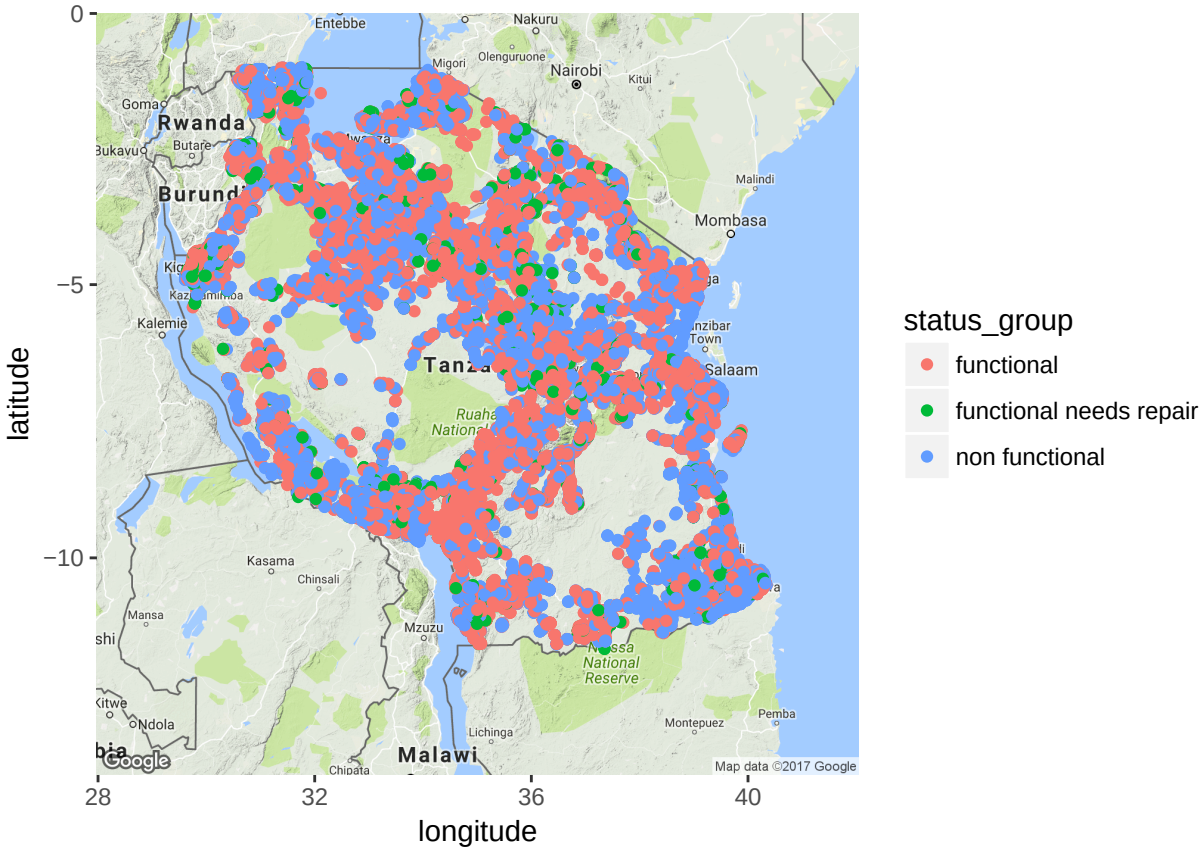
```
FALSE status_group
FALSE              functional functional needs repair          non functional
FALSE              0.54309764              0.07343808              0.38346427
```


30000
20000
10000
0
count
dry
enough
insufficient
seasonal
unknown
quantity
status_group
functional
functional needs repair
non functional
30000
20000
10000
0
count
functional
functional needs repair
non functional
status_group
quantity
dry
enough
insufficient
seasonal
unknown


count
40000
30000
20000
10000
0
colored
fluoride
good
milky
salty
unknown
quality_group
status_group
functional
functional needs repair
non functional


count
20000
10000
0
cattle trough
communal standpipe
communal standpipe multiple
dam
hand pump
improved spring
other
waterpoint_type
status_group
functional
functional needs repair
non functional


functional
functional needs repair
non functional
count
3000
2000
1000
0
1960
1980
2000
construction_year


latitude
0
−5
−10
28
32
36
40
longitude
Rwanda
Burundi
Tanzania
Nairobi
Mombasa
Malawi
Kasama
Mzuzu
Lichinga
Montepuez
Pemba
Ndola
Kalemie
Salaam
Map data ©2017 Google
status_group
functional
functional needs repair
non functional

Above visualizations help choose relevant features that are possible strong predictors.

The value 'dry' of the 'quality' variable seems to isolate non functional waterpumps. Similarly, functional pumps tend to have a 'enough' value. This makes the 'quality' feature a possible strong predictor.

2. Feature Engineering

An additional feature which represents the number of days since the data was obtained, is created.

Unrealistic, impossible and 'NA' values are imputed using mice and random forest.

Reducing factor levels for categorical variables 'installer' and 'funder'. Same values with different name variations are combined to a single level. Rare levels are combined into a single level 'Others'.

```
# Converting date to datetime class
finalWP$date_recorded <- as.Date(finalWP$date_recorded)

# Creating new feature from recorded date
finalWP$days_since_last_recorded <- max(finalWP$date_recorded)-finalWP$date_recorded
finalWP$days_since_last_recorded <- as.integer(finalWP$days_since_last_recorded)

# Selecting features with missing values for imputation
impWP <- select(finalWP,id,population,construction_year)
impWP$population[impWP$population == 0] <- NA
impWP$construction_year[impWP$construction_year == 0]<- NA

# Performing mice imputation, based on random forests
# miceMod <- mice(impWP[, !names(impWP) %in% "id"], method="rf")
```

```
# Generating the completed data
miceOutput <- complete(miceMod)

# Adding imputed values to dataset
finalWP <- select(finalWP,-c(population,construction_year))
finalWP <- cbind(finalWP,miceOutput)

# Converting funder & installer to lowercase
finalWP$funder <- as.character(finalWP$funder)
finalWP$installer <- as.character(finalWP$installer)
chr.cols <- finalWP %>% summarise_each(funs(is.character(.))) %>% unlist() %>% which() %>% names()
finalWP <- finalWP %>% mutate_each(funs(tolower), one_of(chr.cols))

# Installer - reducing factor levels
finalWP$installer <- as.factor(finalWP$installer)
finalWP$installer[finalWP$installer == "" | finalWP$installer == 0 | finalWP$installer == "-"] <- NA
finalWP$installer[finalWP$installer == "gove" | finalWP$installer == "gover" | finalWP$installer == "cer
finalWP$installer[finalWP$installer == "commu"] <- "community"
finalWP$installer[finalWP$installer == "danid"] <- "danida"
finalWP$installer[finalWP$installer == "word" | finalWP$installer == "wo" | finalWP$installer == "word l

levels_installer = 11

installerNames <- names(summary(finalWP$installer)[1:levels_installer])
installer <- factor(finalWP$installer, levels=c(installerNames, "Other"))
installer[is.na(installer)] <- "Other"
```

```r
finalWP$installer <- installer

# Funder - reduce factor levels
finalWP$funder <- as.factor(finalWP$funder)
finalWP$funder[finalWP$funder == "" | finalWP$funder == 0] <- NA

levels_funder = 16

funderNames <- names(summary(finalWP$funder)[1:levels_funder])
funder <- factor(finalWP$funder, levels=c(funderNames, "Other"))
funder[is.na(funder)] <- "Other"
finalWP$funder <- funder

# Imputing missing/incorrect latitudes & longitudes

finalWP$longitude[finalWP$lga =="Bariadi" & finalWP$longitude == 0] <- 34.33104
finalWP$latitude[finalWP$lga =="Bariadi" & finalWP$latitude == -0.00000002] <- -2.69166

finalWP$longitude[finalWP$lga =="Geita" & finalWP$longitude == 0] <- 32.23135
finalWP$latitude[finalWP$lga =="Geita" & finalWP$latitude == -0.00000002] <- -2.88504

finalWP$longitude[finalWP$lga =="Magu" & finalWP$longitude == 0] <- 33.25879
finalWP$latitude[finalWP$lga =="Magu" & finalWP$latitude == -0.00000002] <- -2.45705

# Setting missing values to False
finalWP$public_meeting <- ifelse(finalWP$public_meeting == "TRUE", "True", "False")
finalWP$public_meeting <- as.factor(finalWP$public_meeting)
finalWP$public_meeting[is.na(finalWP$public_meeting)] <- "False"

finalWP$permit <- ifelse(finalWP$permit == "TRUE", "True", "False")
finalWP$permit <- as.factor(finalWP$permit)
finalWP$permit[is.na(finalWP$permit)] <- "False"
```

3. Feature Selection

Unneccesary, non-interpretable, constant or unclear features are removed. Feature importance and selection will be handled by random forest in the modeling section.

```r
# Removing redundant features
finalWP <- finalWP[, -which(names(finalWP) == "recorded_by")] # only one value (organization which reco
finalWP <- finalWP[, -which(names(finalWP) == "quantity_group")] # same as quantity
finalWP <- finalWP[, -which(names(finalWP) == "region_code")] # code for region
finalWP <- finalWP[, -which(names(finalWP) == "date_recorded")] # Date row was entered (not a factor de
finalWP <- finalWP[, -which(names(finalWP) == "num_private")] #undefined,id field
finalWP <- finalWP[, -which(names(finalWP) == "district_code")]
finalWP <- finalWP[, -which(names(finalWP) == "quality_group")] #identical to water_quality
finalWP <- finalWP[, -which(names(finalWP) == "payment_type")] # similar to payment
finalWP <- finalWP[, -which(names(finalWP) == "scheme_management")] #similar to management
finalWP <- finalWP[, -which(names(finalWP) == "management_group")] #similar to management
finalWP <- finalWP[, -which(names(finalWP) == "source")] #similiar to source_type
finalWP <- finalWP[, -which(names(finalWP) == "subvillage")] #too many levels
finalWP <- finalWP[, -which(names(finalWP) == "wpt_name")] #too many levels
```

```
finalWP <- finalWP[, -which(names(finalWP) == "ward")]
finalWP <- finalWP[, -which(names(finalWP) == "lga")]
finalWP <- finalWP[, -which(names(finalWP) == "scheme_name")]
finalWP <- finalWP[, -which(names(finalWP) == "extraction_type")]
finalWP <- finalWP[, -which(names(finalWP) == "extraction_type_group")]
finalWP <- finalWP[, -which(names(finalWP) == "waterpoint_type_group")]
finalWP <- finalWP[, -which(names(finalWP) == "amount_tsh")]
```

4. Modeling Section

Using tuneRF function the optimal value 'mtry = 4' to use in a random forest algorithm is computed.

A model using Boosing trees algorithm gives a higher test error rate than Random forests.

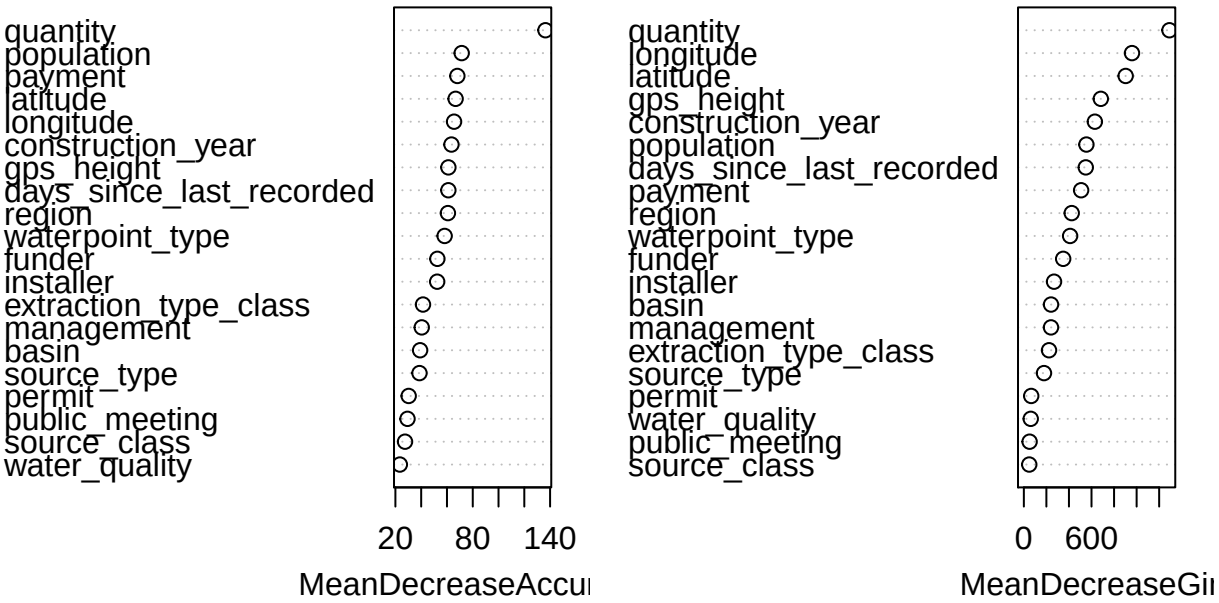
```
# train/test split
set.seed(1234)
sample <- sample.split(finalWP$status_group, SplitRatio = .7)
train <- subset(finalWP, sample == TRUE)
test <- subset(finalWP, sample == FALSE)

# Commented for notebook efficiency:
# Tuning Randomforest for optimal mtry parameter
#rf.all.tune <- tuneRF(finalWP[,-19], finalWP[,19], ntreeTry=800, stepFactor=1.5)

# Randomforest 2
set.seed(12345)
rf.all2 <- randomForest(status_group ~ .-id, mtry=4,ntree = 350,data=train,importance=TRUE)

varImpPlot(rf.all2)
```

rf.all2




```
yhat.rf2 <- predict(rf.all2 ,newdata=test,type = "response")

confusionMatrix(test$status_group,yhat.rf2)
```

```
## Confusion Matrix and Statistics
##
##                          Reference
## Prediction                functional functional needs repair
##   functional                    3939                       94
##   functional needs repair        293                      182
##   non functional                 464                       41
##                          Reference
## Prediction                non functional
##   functional                         288
##   functional needs repair             76
##   non functional                    2112
##
## Overall Statistics
##
##                Accuracy : 0.8323
##                  95% CI : (0.8236, 0.8407)
##     No Information Rate : 0.6271
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6772
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
```

```
##                      Class: functional Class: functional needs repair
## Sensitivity                     0.8388                          0.57413
## Specificity                     0.8632                          0.94855
## Pos Pred Value                  0.9116                          0.33031
## Neg Pred Value                  0.7610                          0.98054
## Prevalence                      0.6271                          0.04233
## Detection Rate                  0.5260                          0.02430
## Detection Prevalence            0.5770                          0.07357
## Balanced Accuracy               0.8510                          0.76134
##                      Class: non functional
## Sensitivity                         0.8530
## Specificity                         0.8993
## Pos Pred Value                      0.8070
## Neg Pred Value                      0.9253
## Prevalence                          0.3306
## Detection Rate                      0.2820
## Detection Prevalence                0.3494
## Balanced Accuracy                   0.8761
```