PumpItUp

#————————————————

Author: Soheil Moattar Berenguer

Date: 2022_02_19

Description: Model for pump status prediction

Input: Train, Test datasets provided by Drivendata.org (https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/)

#————————————————

#Abstract:

The objective of this study was to determine the functioning status of water pumps using a dataset describing different aspects of the pumps. For this aim, a random forest model was used obtaining a final 0.8189 submission score. This was mainly achieved by means of feature engineering processes like: Missing value imputation, lumping, frequency encoding and the creation of new variables.

##Import Libraries

```
library(dplyr)    # For Data Manipulation
library(data.table) # For Fast Data
library(inspectdf) # Automatic Data Exploration Analysis
library(gmodels) #CrossTable()
library(plyr) ## rbind.fill (combine data frames with different columns)
library(ggplot2) ## For awesome graphics
library(ranger)      # fast randomForest
library(magrittr)    # To perform complex pipings
library(ggplot2)     # Very nice charts
library(forcats)     # Manage factor variables
library(missRanger)  #Robust Algorithms for Imputation
library(recipes)    # To perform knn Imputation
library(lubridate)   # Handle date variables efficiently
library(h2o) ## random forest for classification
library(stringi)   # For handling strings
```

Loading Data

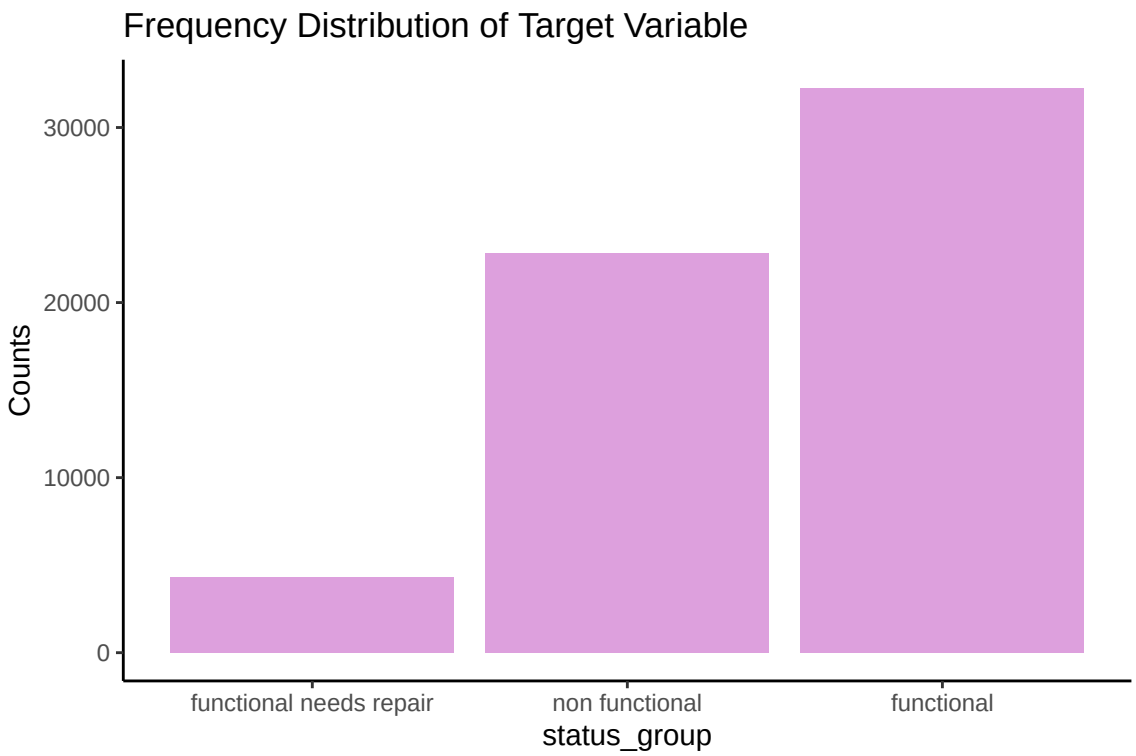
```
#Get Current directory
getwd()


#Put path of the directory containing the data files
x <-paste("/Users/soheilmoattarmohammadiberenguer/Documents/",
      "Complu_Uni/HomeWork/MachineLearning with R/delivery/Data",
       sep="")
```

```
#Se the specified path as new directory
setwd(x)
```

```
# Dump the csv files into R objects

#The 0 is supposed to illustrate that it´s
#the initial raw data

# The data.table argument is used to get a dataframe object

df_train0=fread("train_features.csv",data.table=FALSE)

df_test0=fread("test_features.csv",data.table=FALSE)

train_labels0=fread("Labels.csv",data.table=FALSE)
# For repeatable results
set.seed(123)
```

In order to confirm that the labels dataset has been loaded correctly we count the instances of each of the values and compare it with the graphic illustrated at Drivendata.org

```
# Display value counts of the distinct values
# in a descending order


tbl<-as.data.frame(train_labels0 %>%
  dplyr::count(status_group)%>%
  arrange(-n))
#tbl



#fct_reorder: orders the variable "status_group" according to
#the variable "n"
tbl %>%
    ggplot(aes(x = fct_reorder(status_group,n), y = n)) +
    geom_col(fill = "plum") +
      labs(
    title = "Frequency Distribution of Target Variable ",
    x = "status_group",
    y = "Counts",
    ) +
  theme_classic()
```



File Inspection

Before anything, I convert the levels to lower case, in case there is random capitalization.

```
#train dataset
chr.cols = df_train0 %>% summarise_each(funs(is.character(.))) %>%
  unlist() %>% which() %>% names()
df_train0 = df_train0 %>% mutate_each( funs(tolower), one_of(chr.cols))

#test dataset
chr.cols = df_test0 %>% summarise_each(funs(is.character(.))) %>%
  unlist() %>% which() %>% names()

df_test0 = df_test0 %>% mutate_each( funs(tolower), one_of(chr.cols))
```

```
df_train0 %>%head(,n=5)
```

For a neater experience, the results of this command have been appended to the end of the document.

Just by taking a quick glance at our data we find out a few things: 1- The date_recorded predictor doesn't tell us anything in it's current format it will probably need to be changed

2- Some 0 values in the construction_year variable.

3- There are 3 different categorical variables describing the extraction, with different granularity levels. Probably it will be unnecessary to keep all of the three variables describing the extraction. It is possible that keeping only one or two of them will be sufficient.

4- "unknown" categories in the payment and payment type, which probably indicate NA values. Also, once again, it seems like these two are quasi-identical features.

and etc. . .

If they both have the same values in the "id" column then an inner join can be used.

```
#which(df_train0$id != train_labels0$id) #Does the job but meh..

all.equal(df_train0$id , train_labels0$id) #More elegant
```

The "id" variable in both the features and labels dataframes are identical.

```
#Inner Join

df_train <- merge(x = df_train0, y = train_labels0, by = "id", all = TRUE)
```

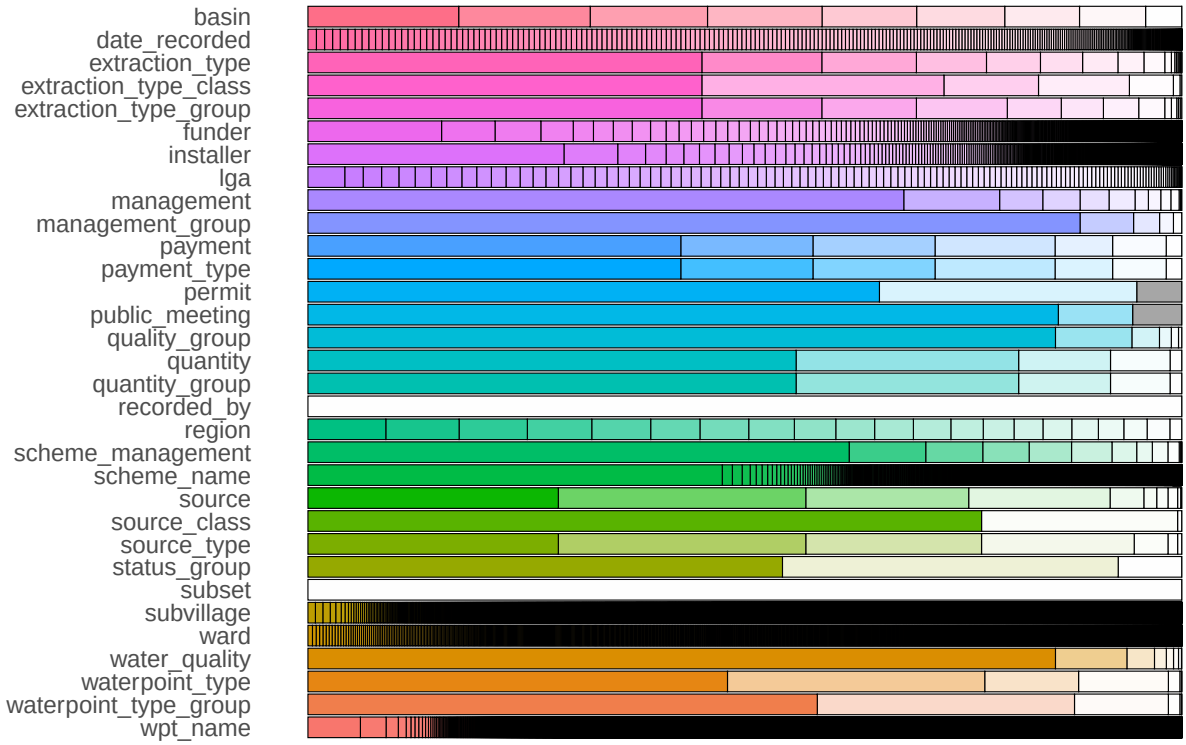
To make things easier and don't fall into any mistakes I will also concatenate the two train and test dataframes.

```
#This way splitting them for the models will be easier.
df_train$subset <- "train"
df_test0$subset <- "test"


#Concatenate
df1 <- plyr::rbind.fill(df_train, df_test0)
```

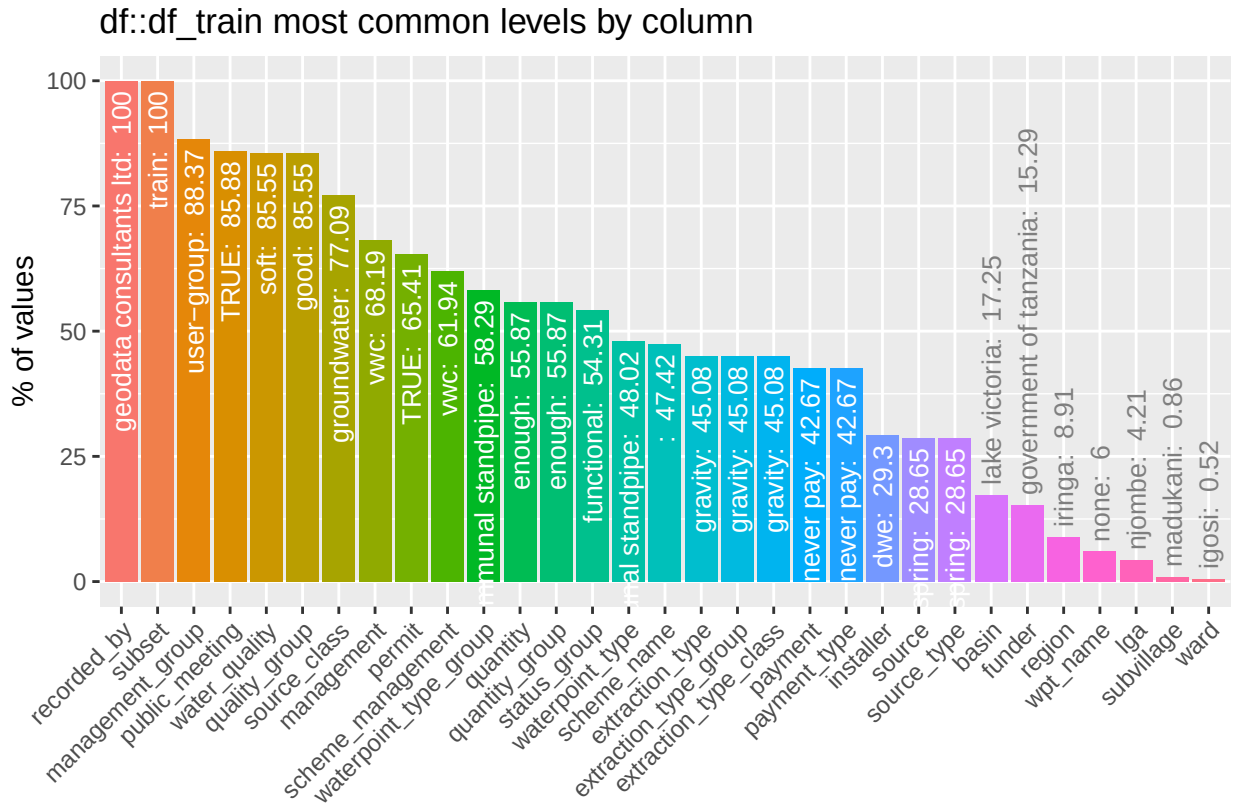
df_train contains the raw train plus the labels. Now df1 contains the raw train, test and labels.

#Exploratory Data Analysis

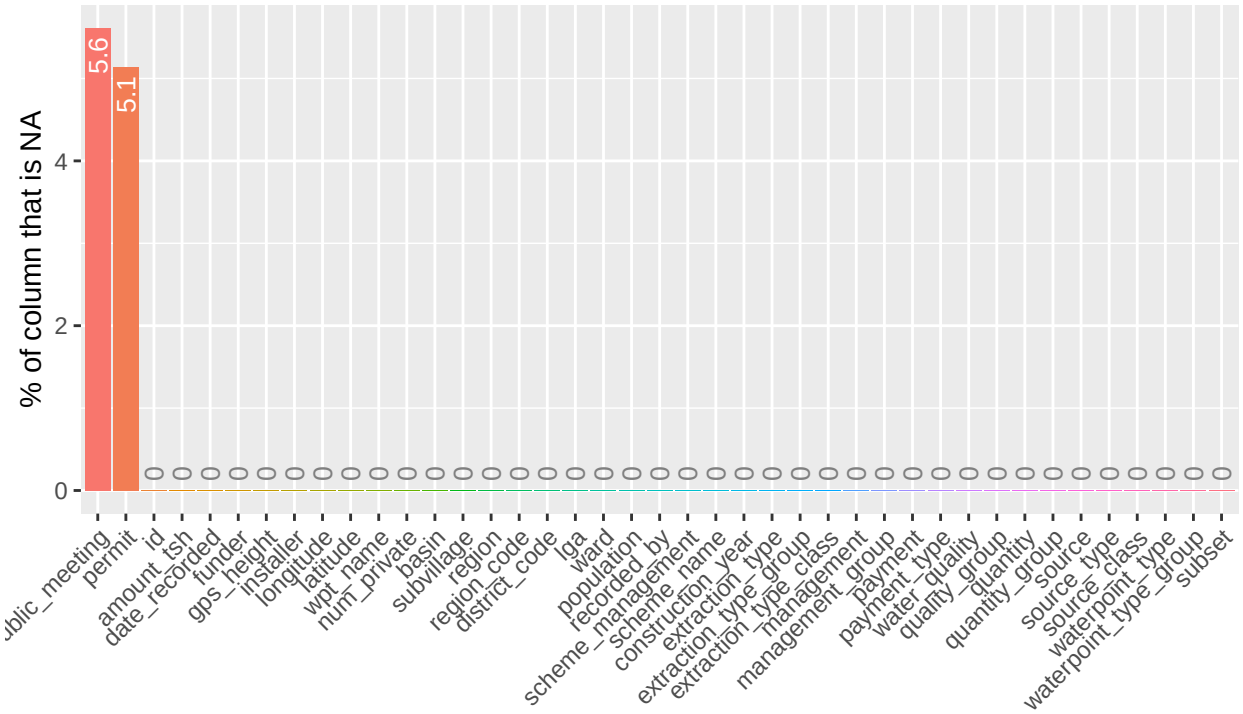
```
# ! The functions in this section belong to the inspectdf library


# Horizontal bar plot for categorical column composition

inspect_cat(df_train) %>% show_plot()
```




```
# Bar plot of most frequent category for each categorical column
x <- inspect_imb(df_train)
show_plot(x,)
```

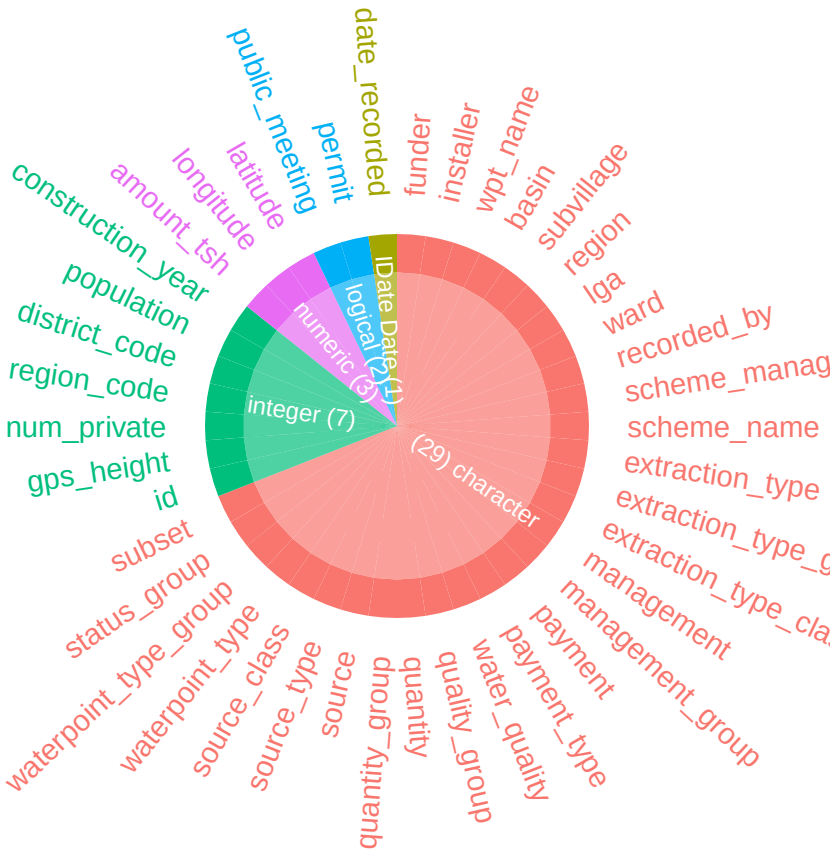



```
# Occurence of NAs in each column ranked in descending order

# The traget variable won't  be included
x <- inspect_na(subset(df_train, select = -c(status_group) ))
show_plot(x)
```




```
# Histograms for numeric columns
#The "id" is excluded since it has no statistical significance
x <- inspect_num(subset(df_train, select = -c(id) ))
show_plot(x)
```


```
# Barplot of column types
inspect_types(df_train) %>% show_plot()
```



Commentaries:

Graphic (1): This graphic displays the different levels in the categorical features and their frequency. It is pretty obvious that the variable "recorded by" only takes a single value so it will be removed from our dataset. In the features that the color turns black probably we have way too many levels and something must be done. Also in a lot of the categorical predictors categories with low frequency can be seen which later in the feature engineering phase they will be handled.

(! I don't know why the names of the labels are not being displayed on the graphic)

Graphic (2): In the variable "recorded_by", *GeoData Consultants Ltd* make up 100% of the values. This denotes a zero_variability predictor, which we already observed in Graphic (1). Also starting from "management_group", and moving to the right side we see a couple of variables where one category makes up for a high precentage of occurences. When resampling we'll have to be cautious with these one to not leave out "rare" categories.

Graphic (3):

*NA*s make up around 5% of the predictors "public_meeting" & "permit" but very probably there are gonna be missing values in the rest of variables too, just with a different value.

Graphic (4):

Then, in the "construction_year" too, a lot of 0s can be seen which should later be converted to *NA*.

Something similar happens to "gps_height", where plenty of zero values along with negative values can be observed. But is this correct? Is it possible to have a negative or even zero gps height?

In addition, the o values in "latitude" and "longitude" are suspicious and must be further evaluated.

In the "population" predictor, most values are concentrated at 0 and also the existence of outliers causes the plot to expand to values up to 30000 .

In "region_code", the majority of values are concentrated between 0 and 25 which means that more pumps have been installed in these regions. It will be interesting to check if there is some

Graphic (5): It seems like the type assigned to the variables is right. Although later maybe it will be a good idea to change the datatype from *char* to *factor*.

```
df1<- df1 %>% select ( - recorded_by)
```

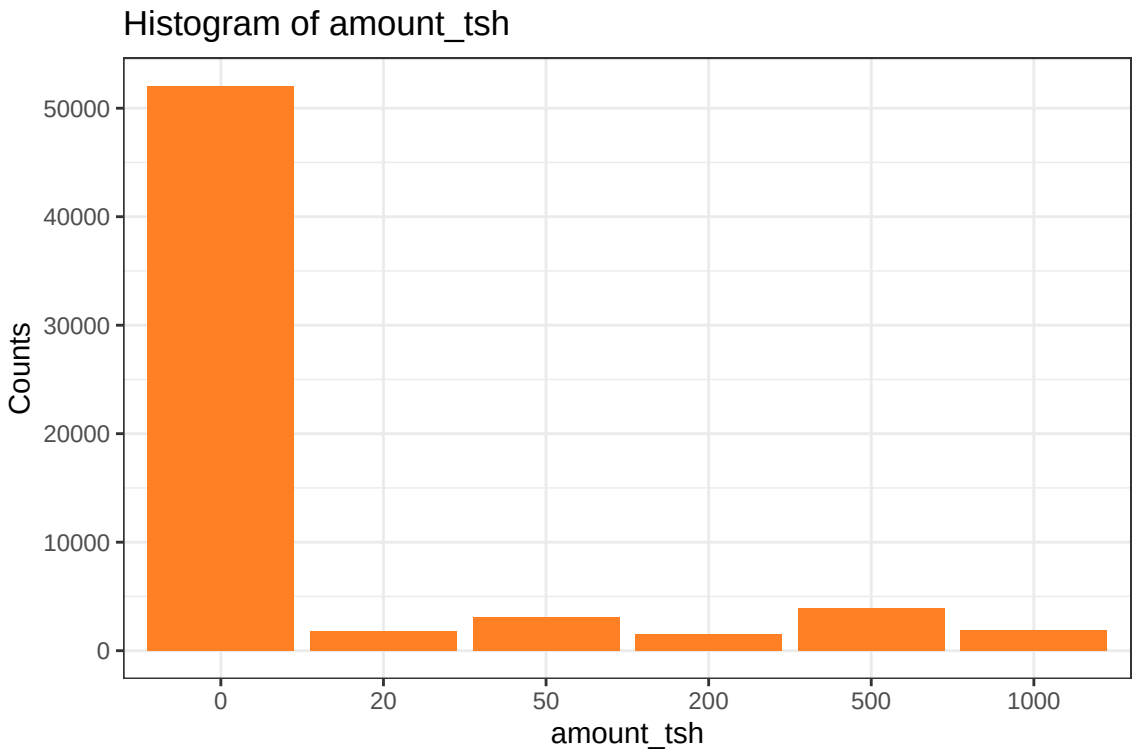
To make sure that the correct type is assigned to each predictor, counting it's unique values seems like a good practice.

```
#subset(train, select = -c(status_group) )
sapply(Filter(is.numeric,subset(df1, select = -c(status_group))),function(x) length(unique(x)))
```

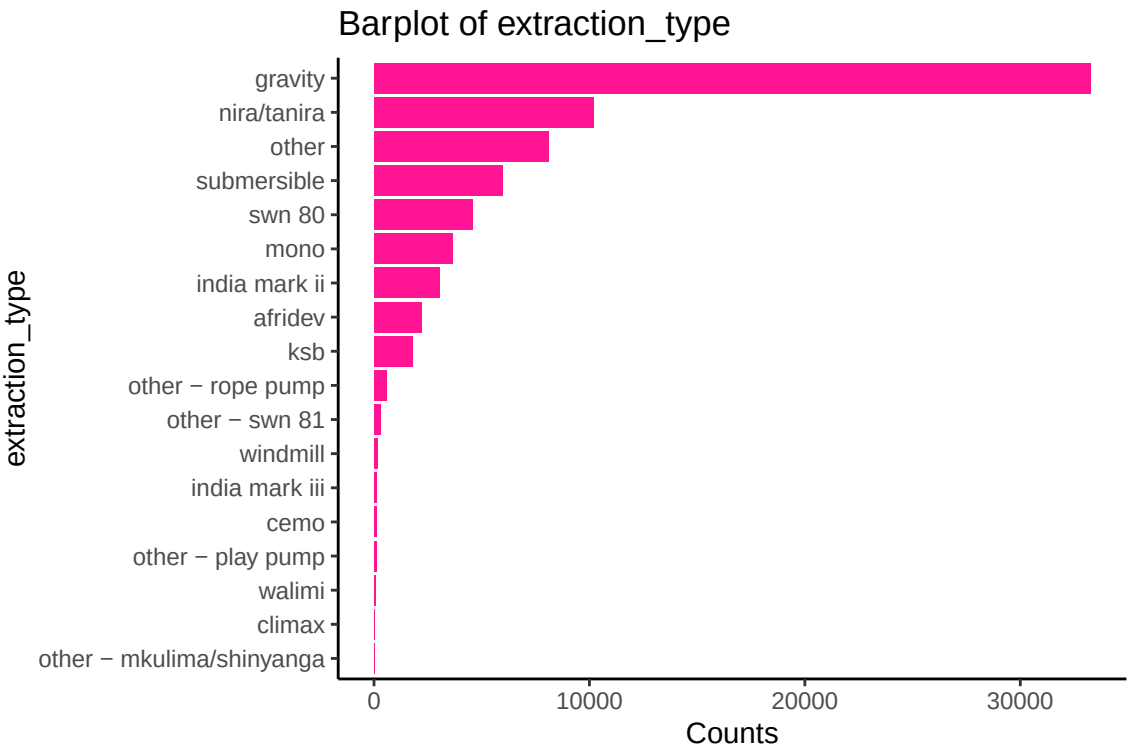
```
##                id        amount_tsh        gps_height         longitude
##             74250               102              2456             71870
##          latitude       num_private       region_code     district_code
##             71869                68                27                20
##        population construction_year
##              1128                55
```

At first sight, maybe only "region_code" and "district_code" seem like they could have categorical/factor type.

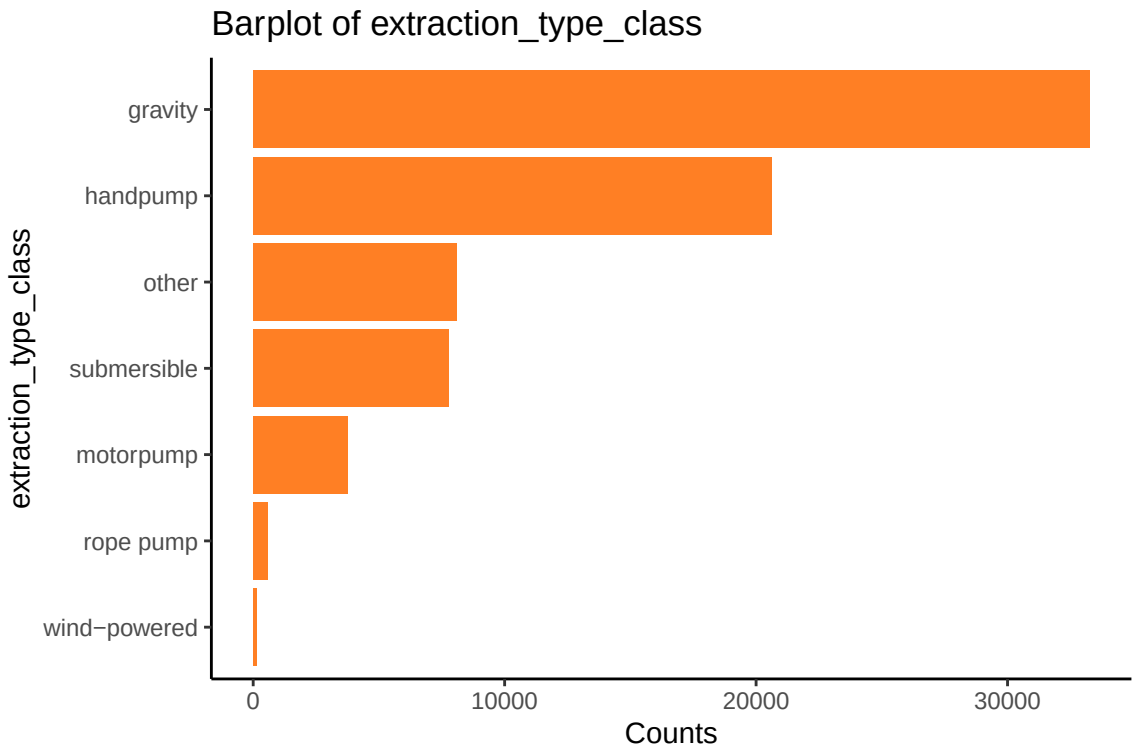
```
tbl<-df1 %>%
     dplyr::count(amount_tsh)%>%
     arrange(-n)%>%head(6)
#
#

 tbl %>%
    ggplot(aes(x = fct_reorder(as.factor(amount_tsh),amount_tsh), y = n)) +
    geom_col(fill = "#FF7F24")+
    labs(
    title = "Histogram of amount_tsh ",
    x = "amount_tsh",
    y = "Counts",
    ) +
  theme_bw()
```




```
tbl1<-df1 %>%
     dplyr::count(extraction_type)%>%
     arrange(-n)


tbl1 %>%
    ggplot(aes(x = fct_reorder(extraction_type,n), y = n)) +
    geom_col(fill = "deeppink")+
    coord_flip()+
        labs(
    title = "Barplot of extraction_type",
    x = "extraction_type",
    y = "Counts",
    ) +
  theme_classic()
```




```
tbl2<-df1 %>%
    dplyr::count(extraction_type_class)%>%
    arrange(-n)


tbl2 %>%
    ggplot(aes(x = fct_reorder(extraction_type_class,n), y = n)) +
    geom_col(fill = "#FF7F24")+
    coord_flip()+
   labs(
    title = "Barplot of extraction_type_class",
    x = "extraction_type_class",
    y = "Counts",
    ) +
  theme_classic()
```



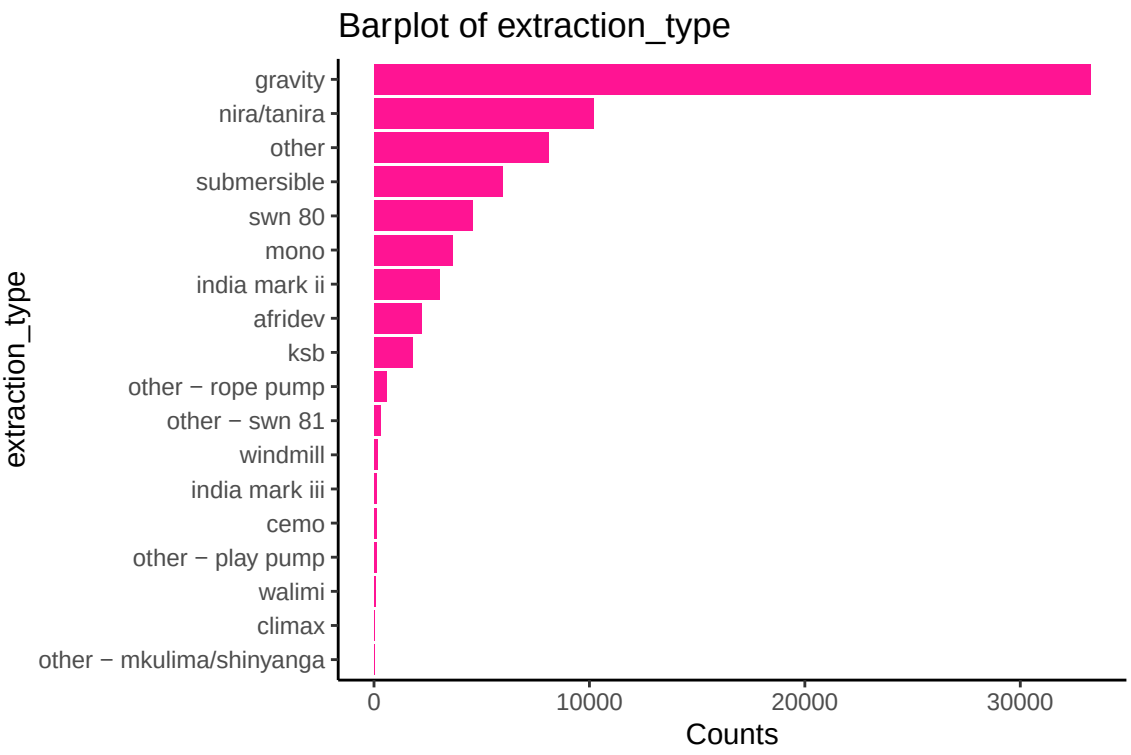
According to this graphic & also Graphic (1), 0s make up for the majority of the values in "amount_tsh", Total static head (amount water available to waterpoint); and then as the value of "amount_tsh" increases, the frequency decreases drastically.

Further into our analysis, at the feature engineering step, it will be interesting to get a binary categorical feature (0 = No water is available , 1= some water is available).

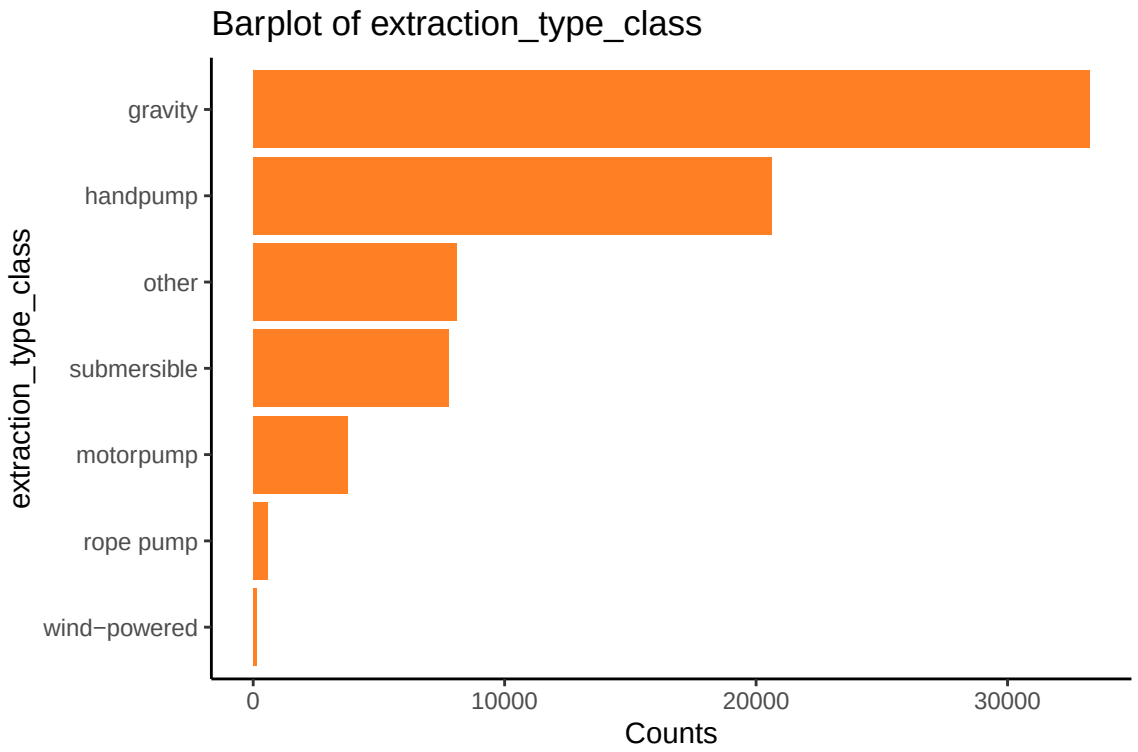
Also binning could be performed as a last resource (Max Kuhn recommends using it as a last resource cause it may create "false" connections and lead to overfitting)

Even though the names do not appear on Graphic (1) there are some "suspicious" categories. "extraction_type" and "extraction_type_class" group in one hand and "payment" and "payment_type", too, seem to have the same categories. So a value count of these suspicious predictors will be needed to further determine our suspicions.

```
tbl1<-df1%>%
     dplyr::count(extraction_type)%>%
     arrange(-n)


tbl1 %>%
    ggplot(aes(x = fct_reorder(extraction_type,n), y = n)) +
    geom_col(fill = "deeppink")+
    coord_flip()+
        labs(
    title = "Barplot of extraction_type",
    x = "extraction_type",
    y = "Counts",
    ) +
  theme_classic()
```

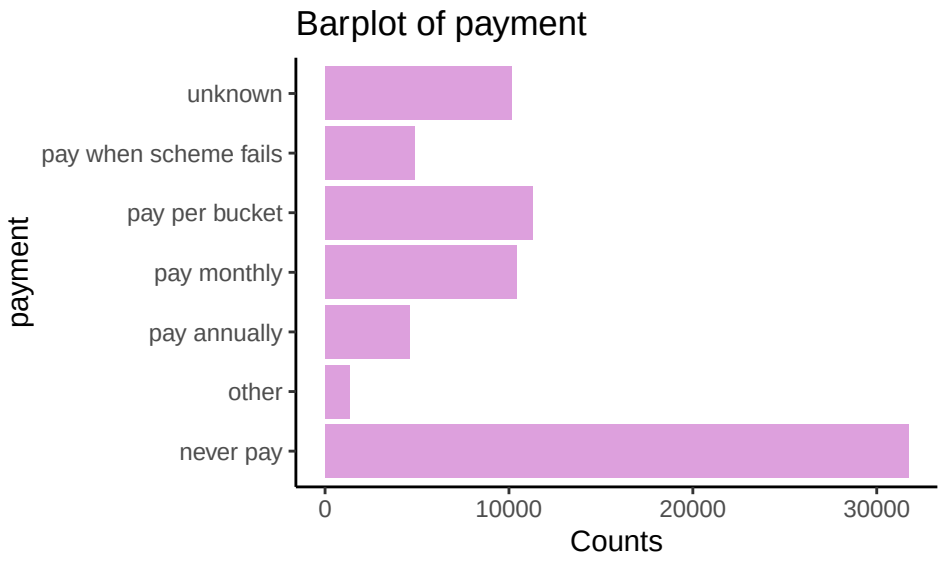



```
tbl2<-df1 %>%
    dplyr::count(extraction_type_class)%>%
    arrange(-n)


tbl2 %>%
    ggplot(aes(x = fct_reorder(extraction_type_class,n), y = n)) +
    geom_col(fill = "#FF7F24")+
    coord_flip()+
   labs(
    title = "Barplot of extraction_type_class",
    x = "extraction_type_class",
    y = "Counts",
    ) +
  theme_classic()
```



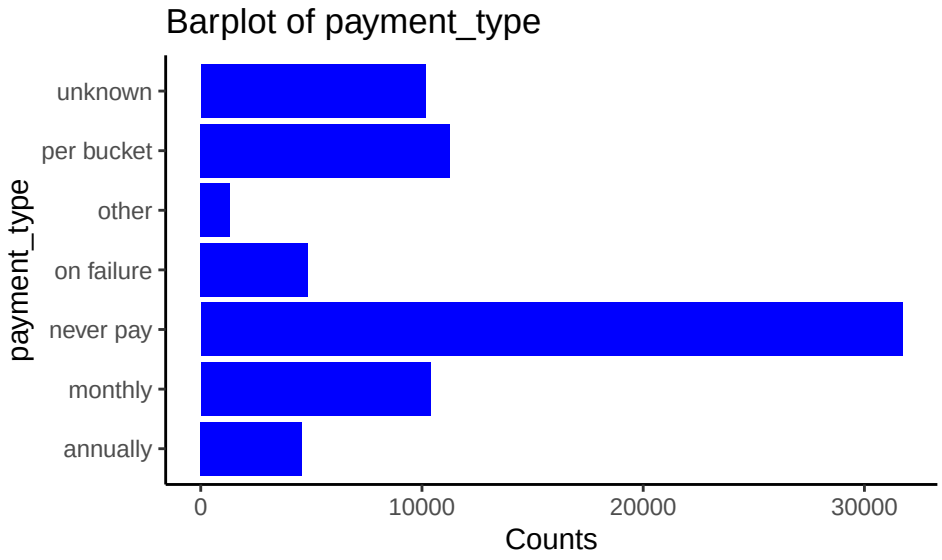
By plotting the frequency of the 5 most frequent classes, it can be observed that the most frequent category in both predictors has the same name and frequency in both of them but especially in the case of "extraction_type_class" we can see some categories with a very low representation, which will need further handling. At this point, we lack sufficient evidence to be able to firm-handedly drop one of the two predictors.

```
tbl1<-df1 %>%
    dplyr::count(payment)%>%
     arrange(-n)


tbl1 %>%
    ggplot(aes(x = payment, y = n)) +
    geom_col(fill = "plum")+
      coord_flip()+
    labs(
    title = "Barplot of payment",
    x = "payment",
    y = "Counts",
    ) +
  theme_classic()
```




```
tbl2<-df1 %>%
dplyr::count(payment_type)%>%
arrange(-n)
```

```
tbl2 %>%
    ggplot(aes(x = payment_type, y = n)) +
    geom_col(fill = "blue")+
        coord_flip()+
       labs(
    title = "Barplot of payment_type",
    x = "payment_type",
    y = "Counts",
    ) +
  theme_classic()
```

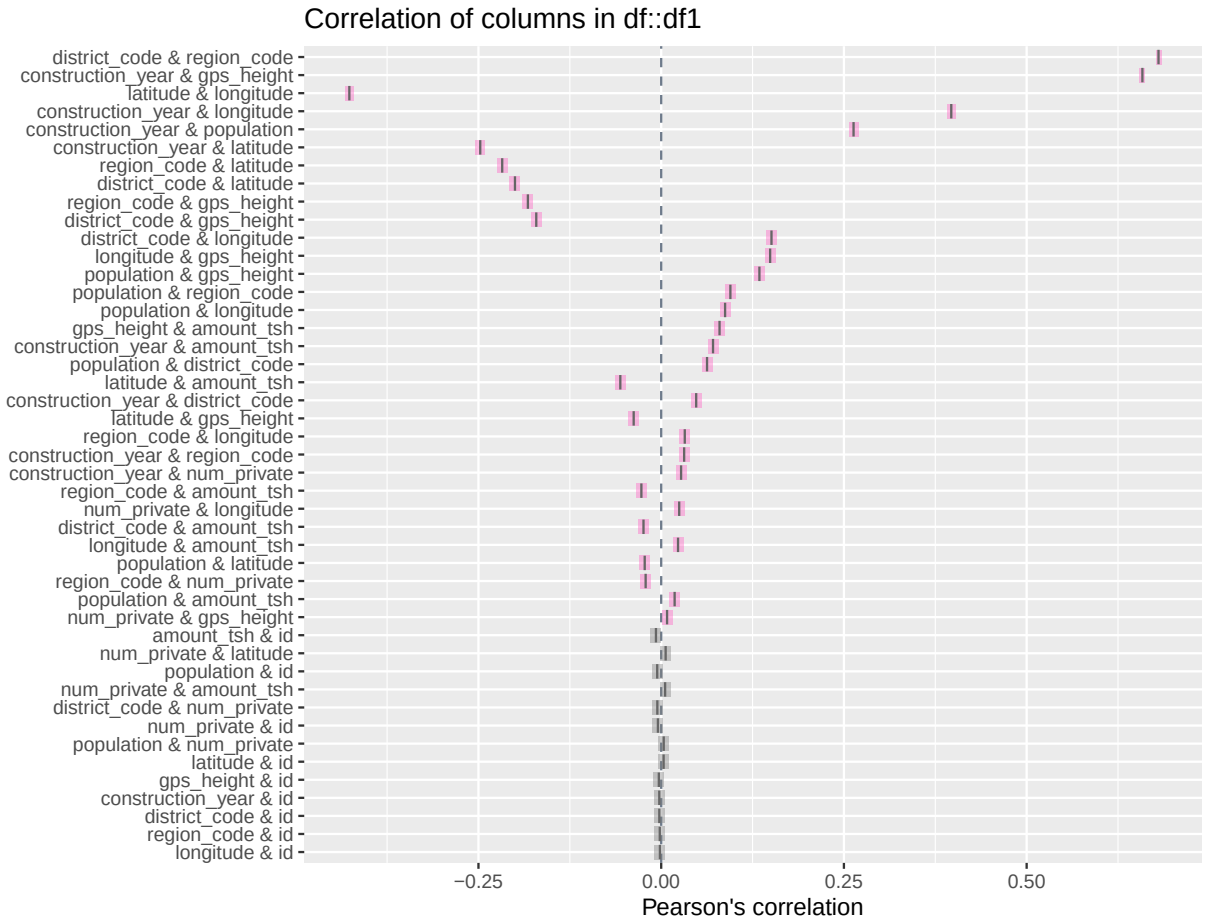


```
df1<- subset(df1, select = -c(payment) )
```

In this case, both predictors have exactly the same categories with the same frequency but with slightly different names. So "payment" will be dropped, since "payment_type" seems like a more descriptive label.

Correlation Plot

```
# Correlation between numeric columns + confidence intervals
x <- inspect_cor(df1)
show_plot(x)
```



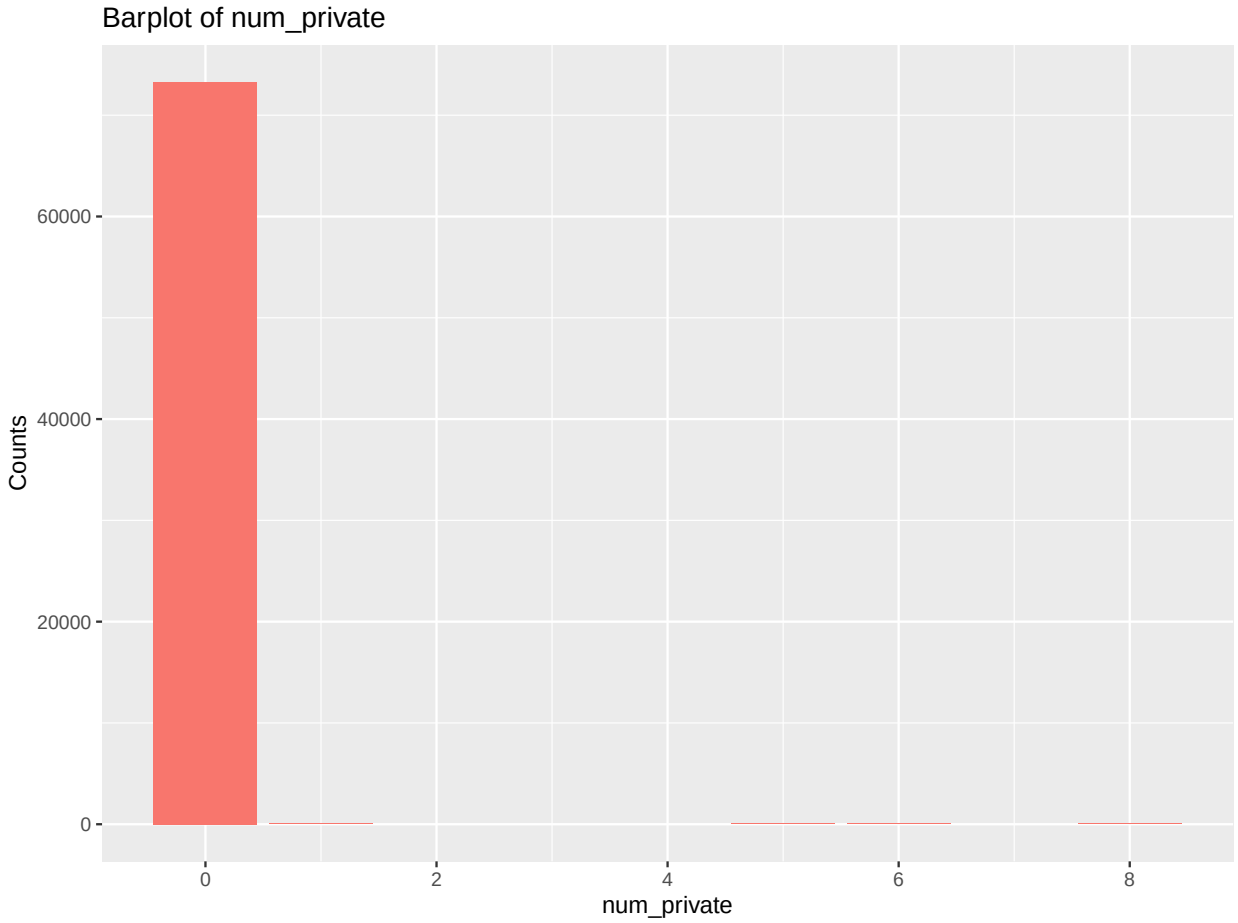
The most remarkable fact is the small correlation between "construction_year" with geo variables such as "gps_height" and "latitude". Other than that no statistically relevant between the features can be observed.

"num_private"

In addition, the predictor "num_private" seems suspicious and no description of it is disclosed so let's try to get know it better.

```r
#Sort by frequency in a descending order
tbl1<-df1 %>%
dplyr::count(num_private)%>%
arrange(-n)%>% head(5)
```

```r
tbl1%>%
    ggplot(aes(x = num_private, y = n)) +
    geom_col(fill = "#F8766D")+
         labs(
    title = "Barplot of num_private",
    x = "num_private",
    y = "Counts",
    )
```



Baseline Model

Following the infamous "Minimum Effective Dose" principle, why perform complex transformations and feature engineering when we may get decent results with our "raw" data and save ourselves some serious

effort?

In addition, this first model can serve other purposes such as: 1- acting as a baseline model so that later we can build up and improve starting from this initial model 2- It's a quick and efficient way to gauge the importance of our predictors in their current shape. In other words, it can be a substitute for other discriminatory models such as Vcramer, Chisquare test and etc...

Luckily, R can handle categorical features in their original form without the need to perform encodings or convert them to any other numerical format.

The *Ranger* library requires that the target is of type factor.

```
df1_numeric<- df1 %>% select(where(is.numeric))
df1_numeric%>%names()
```

```
##  [1] "id"                "amount_tsh"        "gps_height"
##  [4] "longitude"         "latitude"          "num_private"
##  [7] "region_code"       "district_code"     "population"
## [10] "construction_year"
```

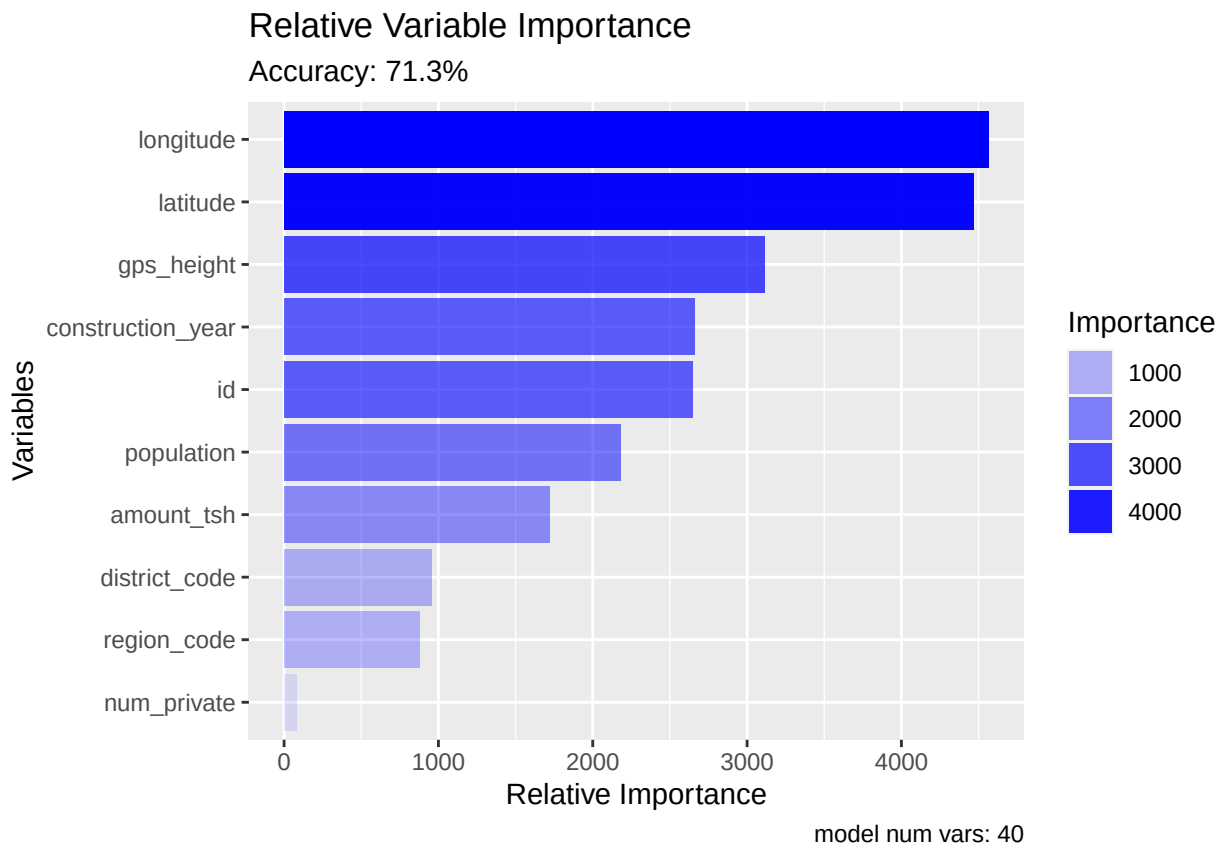
```
#Create dataframe for submission

train<- df1%>%filter(subset=="train")
test<- df1%>%filter(subset=="test")


train_numeric<- train %>%
              select(where(is.numeric),status_group) %>%
              mutate(status_group = as.factor(status_group))

mydata<-train_numeric

# If it gives you an error, make sure that your target i
#is a factor

mymodel <- ranger(
                  status_group ~ .,
                  data      =  mydata ,
                  num.trees = 500,
                  mtry      = 3,
                  importance="impurity",
                  seed=123
                )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

As a starting point, a 0.712 accuracy isn't horrible. This means that out of 100 outcomes, the model would correctly predict 71 of them.

Although, in our case, getting the variable importance is where the real value of this initial attempt lies.

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()
```

```r
#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

Comentaries:

The variables related to the geographical position of the pump seem to have a great influence on the pump health.

Also, the "construction_year" displays a significance importance, which seems to be in accordance with the common intuition that material goods have a finite lifetime.

The "id" variable ranks pretty in importance. For commercial purposes this importance would be neglected, since it´s probably just a matter of chance.

And finally, the "mysterious" variable "num_private" is barely relevant in it's current form.

Note to self: Remember that data engineering is all about getting signals from the features/variables at hand. So when a feature provides us little signal we transform that feature in a way so that we can obtain a signal from it.

#— Confusion Matrix. #– Predicted Values vs. Actual values

```
mymodel$confusion.matrix
```

Count-wise,the model tends to miss-classify the *functional* category more than the other two categories. Although when looked as a percentage the model shows the highest percent of miss-classification in the *functional needs repair* class where for every 7 correct predictions it almost 35 wrong predictions are done, so that's way worse than tossing a coin (50% chance). At a first sight, It could be said that the current model lacks sensitivity and that's probably the reason it predicts so bad the category with the least frequency.

#— Submission

#– Prediction

```
pred_val <- predict( mymodel , data = df_test0)$predictions
head(pred_val)
```

#– Prepare submission

```
sub_df <- data.frame(
  id = df_test0$id,
  status_group = pred_val
)
```

#– Save submission

```
fwrite(sub_df, "./MySubmissions/ranger_vars_numerical_0713.csv", nThread = 3 )
```

_ Local score: 0.713 - Submission Score: 0.71

```
sapply(Filter(is.character,df_train),function(x) length(unique(x)))
```

Here it can be seen that in some of the variables of character class, the number of unique categories is way over the number where common label encoding techniques such as OneHotEncoding or Dummies can be applied. Just as an arbitrary value, if the number of unique values in a categorical variable is higher than let's say 15 or 20, then the conventional mentioned methods will leave us with a lot of extra columns in our dataframe, which aside from not being visually aesthetic, it can dimnish the computational speed of the model.

#Combine the Numerical Dataframe with some of the Categorical Features

So now the question arises of which categorical predictors should we choose along with our previously selected numerical predictors. Some people would choose not to include categorical predictors with let's say over 100 or 150 distinct categories but here we will only choose the categorical predictors with less than 25 different categories.

```
#Choose just the character columns and calculate the unique occurences of
#each category ,convert to dataframe and drop the last row which contains the target
#variable since it is already included in the numerical dataframe
train_chartype<-sapply(Filter(is.character,df_train),function(x) length(unique(x)))%>% as.data.frame()%



# Rename the row names and columns
names(train_chartype)[1]<-"Unique_Values"
train_chartype$Variables<-rownames(train_chartype)
rownames(train_chartype)<-NULL
```

In this section, the name of categorical columns with no more than 25 distinct values are extracted and converted to a vector

```
#-- Only categorical variables with up to 25 distinct categories will be included
lev_good <- train_chartype %>%
  filter( Unique_Values <= 25) %>%
  #-- This selects just the Variables column (the column containing the name of the variables)
  select(Variables) %>%
  #-- Convert to vector
  pull(Variables)
```

```
#length(lev_good) -> 18


#Get only the variables with less than 18 distinct levels
#and also exclude status_group since it is already included in the numeric train df
dfOrichargood<- subset(df_train, select = c(lev_good) )%>%select(-status_group)
```

Bind the approved categorical predictors to our previously prepared numerical dataframe.

```
# drop the previouly created random variables from the numerical dataframe,
# and bind it with the newly created dataframe containing the desired categorical variables
# and finally relocate the target variable so that it's placed as the last column in the
#dataframe to make it more visually appealing and finally convert the target to factor type
#so that it's digestable for the ranger random forest model




dataTrainnumschargood <- cbind(train_numeric,dfOrichargood) %>%
  relocate( status_group, .after = waterpoint_type_group)%>%
  mutate(status_group = as.factor(status_group))



mydata<-dataTrainnumschargood


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                   status_group ~ .,
                   data      =  mydata  ,
                   num.trees = 500,
                   mtry      = 3,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
```

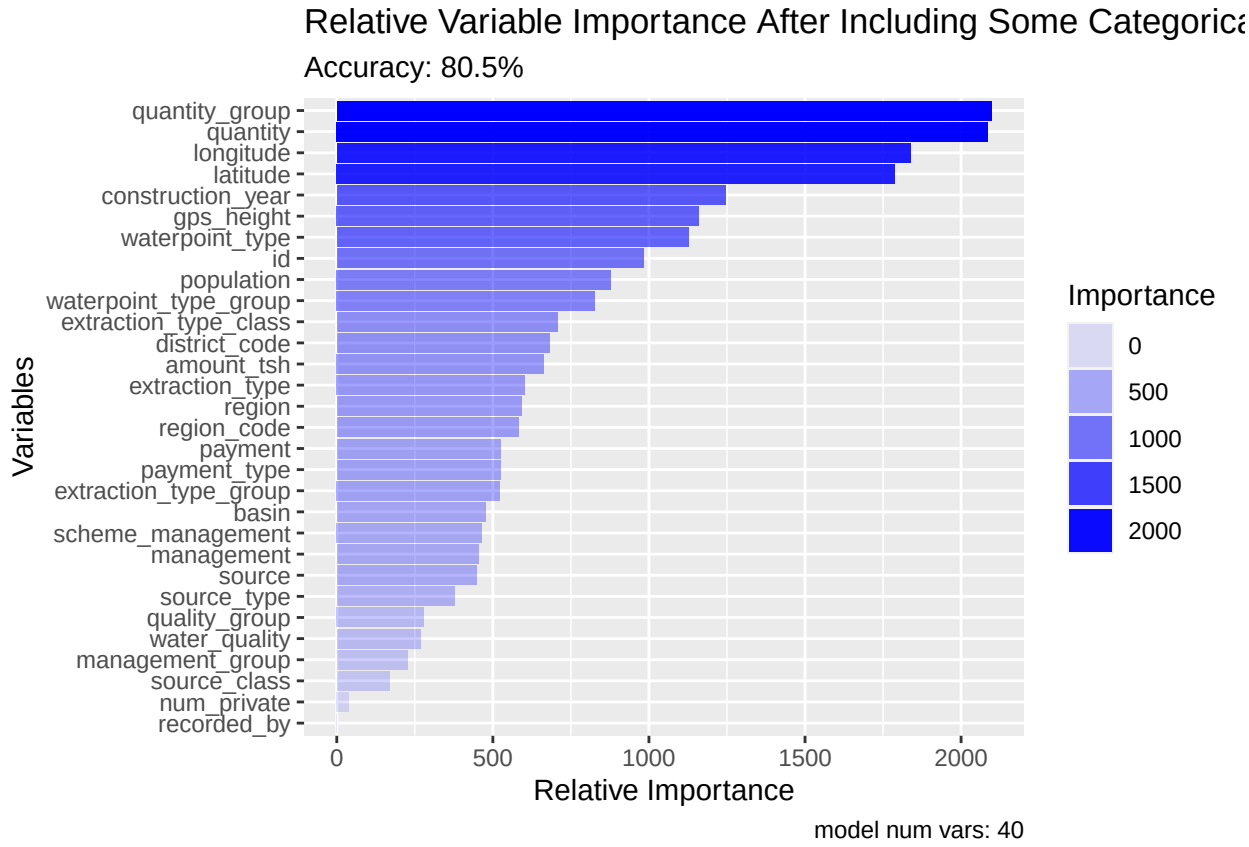
```
## [1] 0.1950337
```

```
accu_val <- 1 - error_val
accu_val
```

```
## [1] 0.8049663
```

Wow! That's almost a 10% improvement just by including almost 75% percent of the categorical variables, and we haven't even started the feature engineering stage.

```r
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Including Some Categorical Variables",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

It can be seen that the two quantity-related variables, which describe the amount of water passing through the pump, score the highest in importance followed by the "geo-variables" , and the construction_year. Also, the "waterpoint_type" predictor seems to have a remarkable relevance followed by the "id" column , which as we said earlier it's relevance in the model might be just a matter of chance.

And from the bottom up, it seems like the variables related to the water quality and source such as "source_class", "water_quality" and "quality_group" don't seem to play a significant role, at least in their current shape.

And curiously the variable related to how the waterpoint is managed (management_group) doesn't seem to be important either.

#— Confusion Matrix. #– Predicted Values vs. Actual values

```
mymodel$confusion.matrix
```

It is pretty obvious that the predicting ability of the *functional needs repair* class has improved a lot (now it only 3:1 compared to the previous model which was almost 7:1). The *non functional* category predictions have also seen an improvement while the *functional* category have experienced the less improvement.

#—– Submission #– Prediction

```
pred_val <- predict( mymodel , data = df_test0)$predictions
head(pred_val)
```

#– Prepare submission

```
sub_df <- data.frame(
  id = df_test0$id,
  status_group = pred_val
)
```

#– Save submission

```
fwrite(sub_df, "./MySubmissions/ranger_vars_numerical_and_goodcats_0805.csv", nThread = 3 )
```

- Local Score: 0.805
- Submission Score: 0.808

The result in the validation set is even better than the result in the train set, that could be a sign that we are moving in the right direction and also avoiding overfitting.

Model with almost all the variables

This time, before performing any feature improvement, almost all the variables will be included.

It should be noted that the ranger package cannot handle variables of logical type or *NA* values, so the two logical variables "public_meeting" and "permit", which by coincidence are the only variable with "official" *NA* values will be the only discarded features.

```
#contains features+label
train<- df1%>%filter(subset=="train")

test<-df1%>%filter(subset=="test")

dfAlmostAll_data<-subset(train, select = -c(public_meeting,permit))%>%mutate(status_group = as.factor(s


mydata<-dfAlmostAll_data


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                   status_group ~ .,
                   data      = mydata  ,
                   num.trees = 500,
                   mtry      = 3,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
```

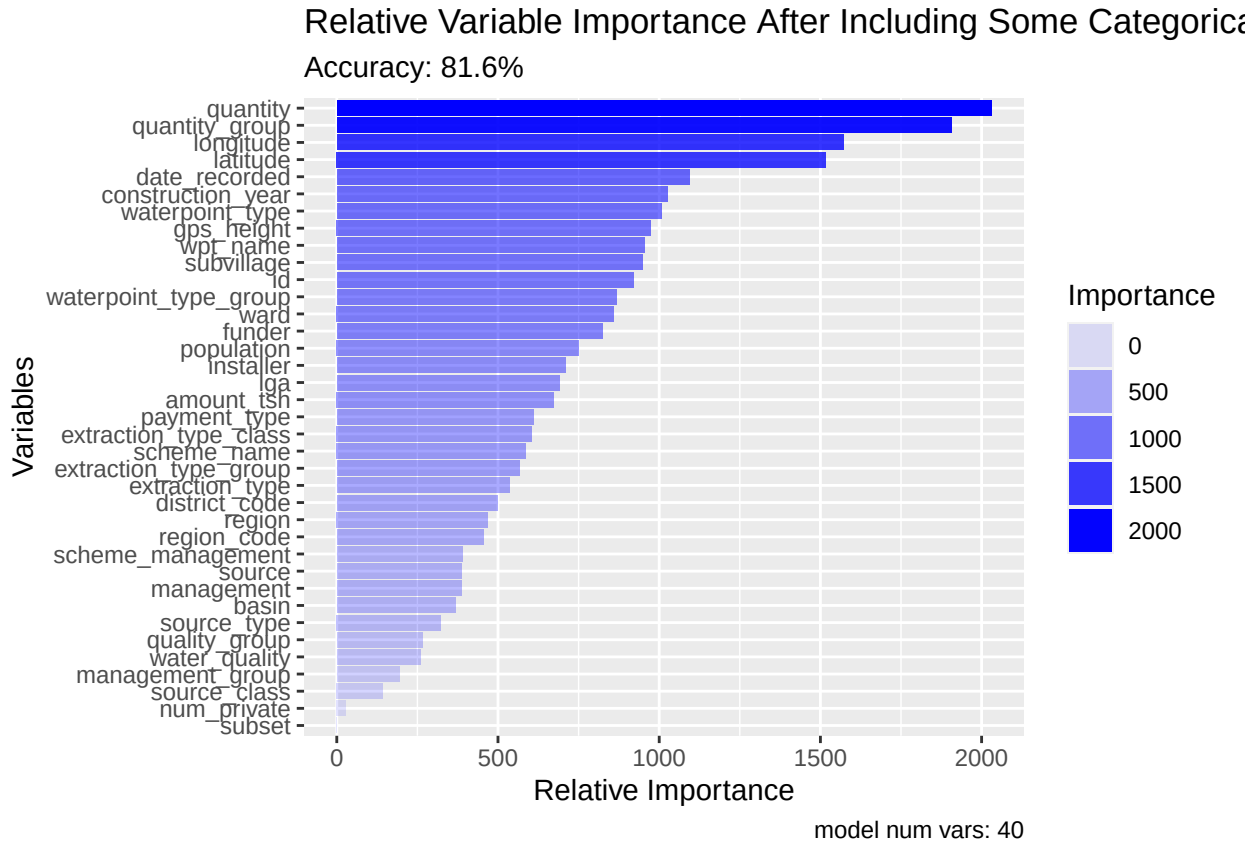
```
## [1] 0.1839731
```

```
accu_val <- 1 - error_val
accu_val
```

```
## [1] 0.8160269
```

By including some of the other categorical features, the local score has improved almost 1%.

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Including Some Categorical Variables",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

Now the difference between the first two quantity related variables with the longitude and latitude variables has increased. In addition the newly added "date_recorded" variable seems to be pretty important although I would say this is just a matter of chance since this variable has still not been converted to the right format.

#— Confusion Matrix. #– Predicted Values vs. Actual values

```
mymodel$confusion.matrix
```

Again, some minuscule improvements can be seen in the confusion matrix table.

#— Submission #– Prediction

```
pred_val <- predict( mymodel , data = df_test0)$predictions
head(pred_val)
```

#– Prepare submission

```
sub_df <- data.frame(
  id = df_test0$id,
  status_group = pred_val
)
```

#– Save submission

```
fwrite(sub_df, "./MySubmissions/ranger_almostall_cats_0816.csv", nThread = 3 )
```

- Local Score: 0.8071
- Submission Score: 0.816

It can be seen that the inclusion of the new rest of the categorical variables didn't help improve the validation accurracy even 1%.

#Include All Variables: Including the logic variables : permit & public_meeting

As mentioned earlier these two variables were also the only two variables with *NA* values in them. So an imputation process will be carried out on them before including them in the model.

```
#This step is repetitive but it helps us not mess up

#train contains features+labels


train<- df1%>%filter(subset=="train")

test<-df1%>%filter(subset=="test")
```

Percent of *NA*s in train dataset

```
#Number NAs Train
sum(is.na(train$permit))*100/nrow(train)
```

```
## [1] 5.144781
```

```
sum(is.na(train$public_meeting))*100/nrow(train)
```

```
## [1] 5.612795
```

Percent of *NA*s in test dataset

```
sum(is.na(test$permit))*100/nrow(test)
sum(is.na(test$public_meeting))*100/nrow(test)
```

At least the percent of missing values is not high in neither the train nor test dataset and actually they have really similar distributions. But as it could be observed from the Graphic (1) obtained by *inspectdf* library the *NA*s in these two variables could be considered almost as a proper category.

Now the question arises that which imputation method is the most convenient?

While using the mean or median value for imputation is kind of common to fill the *NA* values, it's such a naive way to handle the missing data in the dataset. Most of the times, the value of a variable in a particular row is correlated/related with the values of the other columns in that same row.

That's where a more intuitive and sophisticated imputing method based on using a random forest model or even k-means model to fill the missing value comes into play. This methods leverage the predicting ability of the previously mentioned models to fill the *NA* values using the rest of the columns as predictors.

```
#If there are any logical cols select them and the %<>% pipe to simultaneously
#convert them to numeric and save in the dataframe
df1%<>% mutate_if(is.logical,as.numeric)
```

```
# Select all columns except the target
all_data<- subset(train)
names(all_data)
```

```
##  [1] "id"                  "amount_tsh"          "date_recorded"
##  [4] "funder"              "gps_height"          "installer"
##  [7] "longitude"           "latitude"            "wpt_name"
## [10] "num_private"         "basin"               "subvillage"
## [13] "region"              "region_code"         "district_code"
```

```
## [16] "lga"                   "ward"                  "population"
## [19] "public_meeting"        "scheme_management"     "scheme_name"
## [22] "permit"                "construction_year"     "extraction_type"
## [25] "extraction_type_group" "extraction_type_class" "management"
## [28] "management_group"      "payment_type"          "water_quality"
## [31] "quality_group"         "quantity"              "quantity_group"
## [34] "source"                "source_type"           "source_class"
## [37] "waterpoint_type"       "waterpoint_type_group" "status_group"
## [40] "subset"
```

```
target<-train$status_group
```

#MissRanger Libary for Imputation Now that the two logical variables have been converted to 0s and 1s, the imputation will be carried out using the missranger library. This library automatically detects the missing records and uses a random forest model to impute them.

Before that, it is crucial to make sure that our dataset has a dataframe format.

```
df1_Imputed <- missRanger(subset(df1,select= -c(status_group)), pmm.k = 3, num.trees = 100)


#df1_Imputed does not contain the target variable
```

It can be seen that the missranger() library has used the rest of variables in the dataframe to impute the two variables containing *NA*s by means of a random forest algorithm.

The train and test dataset are separated again to be used in the random forest algorithm.

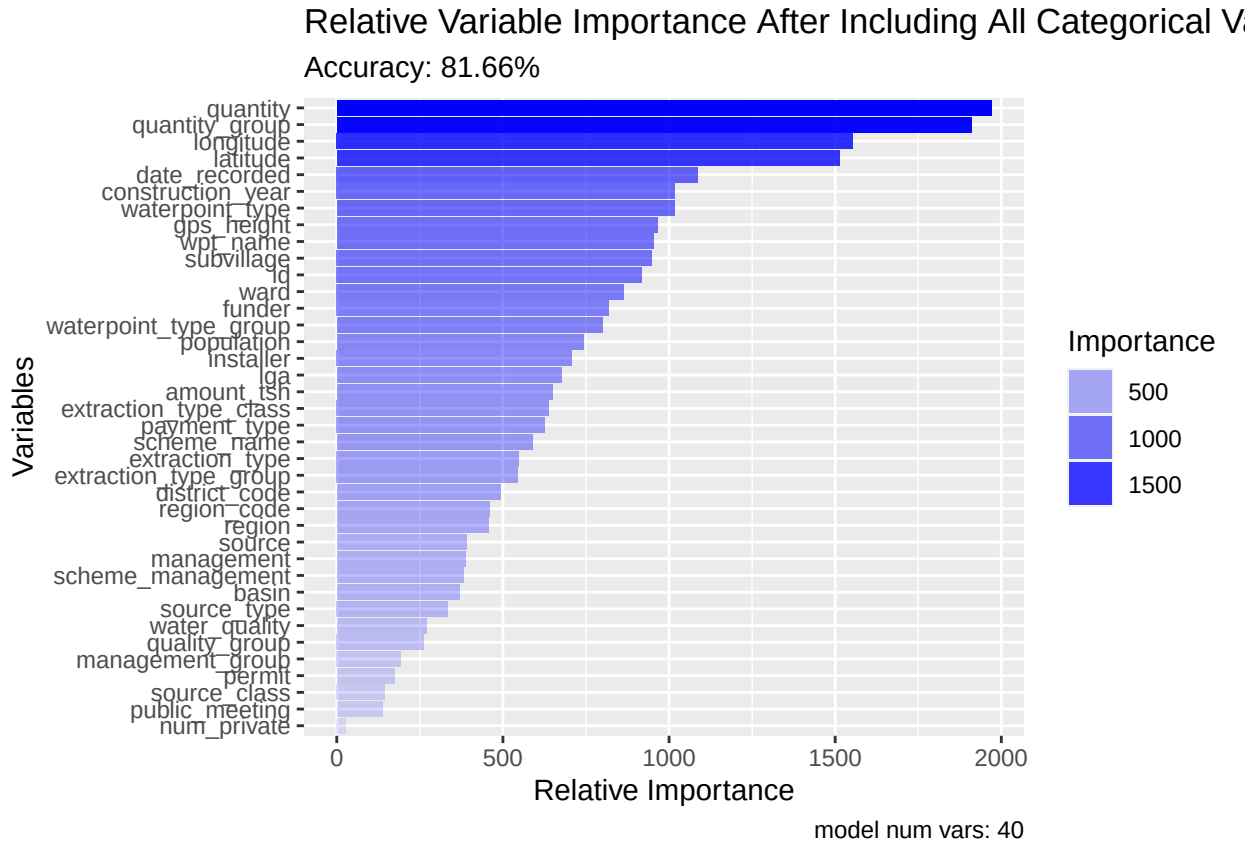
```
#contains only the imputed features
train1_Imputed<- df1_Imputed%>% filter(subset=='train')
test1_Imputed<- df1_Imputed%>% filter(subset=='test')




train1_Imputed <- merge(x = train1_Imputed, y = train_labels0, by = "id", all = TRUE)
```

```
#target should be of type factor
train1_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train1_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                  status_group ~ .,
                  data      = mydata  ,
                  num.trees = 500,
```

```
                    mtry      = 3,
                    importance="impurity",
                    seed=123
                  )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

By including some of the other categorical features, the local score has improved almost 0.1%.

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Including All Categorical Variables",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

None of the logical variables that have been imputed score high in importance.

#— Confusion Matrix. #– Predicted Values vs. Actual values

```
mymodel$confusion.matrix
```

Almost no improvement at this point.

#—– Submission #– Prediction

```
pred_val <- predict( mymodel , data = test1_Imputed)$predictions
head(pred_val)
```

#– Save submission

```
fwrite(sub_df, "./MySubmissions/ranger_almostall_cats_08166.csv", nThread = 3 )
```

- Local Score: 0.806
- Submission Score: 0.8166

It can be seen that the inclusion of the new rest of the categorical variables didn't help improve neither the validation accurracy nor the local accuracy.

#Feature Engineering

Now that we have reached an acceptable baseline with all the predictors existent in the raw data, the time has come to get more signal from the predictors in our dataset by cleaning and/or transforming our variables in case it is needed.

As a reminder:

- df1_Imputed ==> contains all train and test features (- labels) after the imputation of the logical variables
- train1_Imputed ==> contains train dataframe features after the imputation of the logical variables
- test1_Imputed ==> contains test dataframe features after the imputation of the logical variables

---

#quantity, quantity_group

These two variables deal with the quantity of water handled by the pump and both of them rank very high in importance, so they will be the first ones in the process.

```
df1_Imputed%>% group_by(quantity, quantity_group) %>% tally()
```

The variables are identical so "quantity_group" will be dropped and "quantity" will be kept as it is.

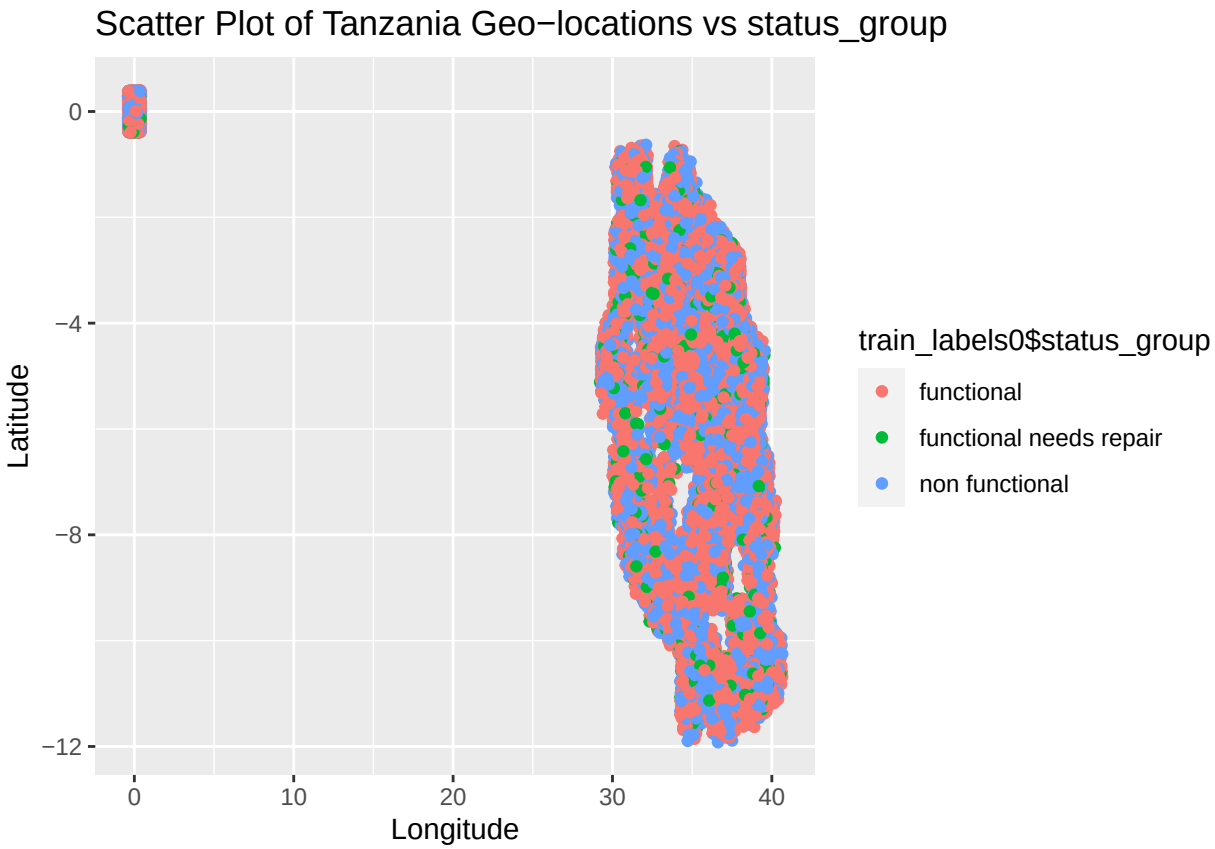
```
#drop variable scheme_name
df1_Imputed %<>%select(-quantity_group)
train1_Imputed %<>%select(-quantity_group)
test1_Imputed %<>%select(-quantity_group)
```

#Engineering the Geo-predictors

Since the geo-predictors such as longitude, latitude and gps-height seemed to hold a great importance, the next step will be to get more signal and cleanse this variables.

Since we don't have the target variable for the test dataframe(test1_Imputed) the plotting will be performed using only the train dataframe(train1_Imputed)

```
# NA lot lat. Tanzania Map.


train1_Imputed%>%
  #filter(type=="train") %>%
  ggplot(.,aes(longitude, latitude, color=train_labels0$status_group)) + geom_point(alpha=.5) + geom_ji
  labs(
    title = "Scatter Plot of Tanzania Geo-locations vs status_group ",
    x = "Longitude",
    y = "Latitude",
  )
```




```
  theme_classic()
```

According to Google The latitude of Tanzania is 6.3690° S, and the longitude is 34.8888° E, This means that the latitude values can only have negative (non-zero) values and the longitude can accept only positive and non-zero values, so the 0 values must be recategorized into *NA* values.

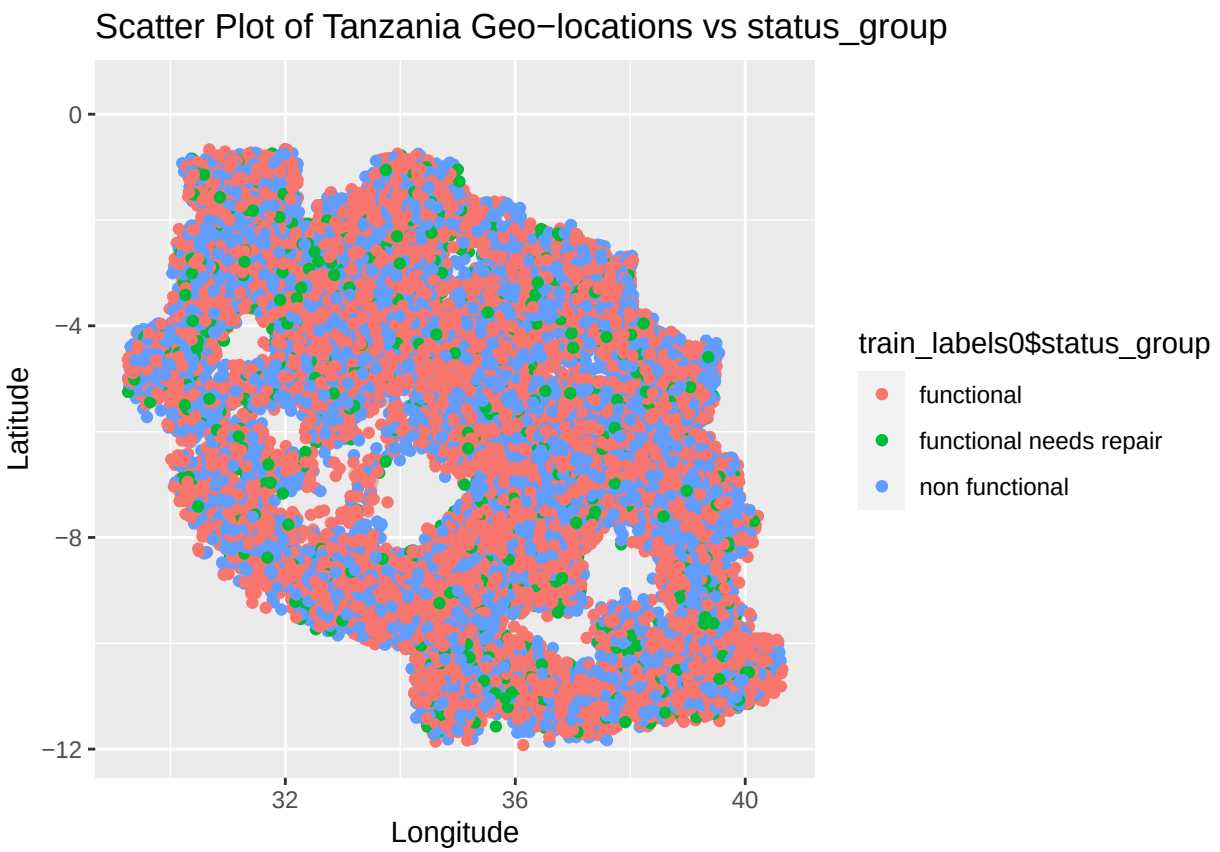
We have to keep in mind to do changes in both the train and test data.

```
train1_Imputed <- train1_Imputed %>%
  mutate(latitude = ifelse(latitude > 0, NA, latitude)) %>%
  mutate(longitude = ifelse(longitude <= 0, NA, longitude))



test1_Imputed <- test1_Imputed %>%
  mutate(latitude = ifelse(latitude > 0, NA, latitude)) %>%
  mutate(longitude = ifelse(longitude <= 0, NA, longitude))
```

After applying this change the distribution map looks way better

```
# No more 0 values
train1_Imputed%>%
  #filter(type=="train") %>%
  ggplot(.,aes(longitude, latitude, color=train_labels0$status_group)) + geom_point(alpha=.5) + geom_ji
  labs(
    title = "Scatter Plot of Tanzania Geo-locations vs status_group ",
    x = "Longitude",
    y = "Latitude",
```

```
)
```




```
  theme_classic()
```

Usually another popular and interesting thing to do with the coordinate variables is to create a *distance* variable using them. The regular distance between two points can be calculated by taking the square of the sums of the cubes of the two coordinates.

Another popular way to calculate distance is the haversine distance which calculates the arc distance between two points on earth. Just as a crazy idea I am going to calculate the haversine distance of each datapoint from the Indian Ocean (I extracted the Indian Ocean coordinates from Google Maps : -6.230069,39.034139)

Also, as I saw in google maps, there are two main regions which denote if that area is more desert like or more green and fertile. Almost in the middle of this *desert like* region the capital of Tanzania, Dodoma, can be found. So I will also create another haversine distance using the coordinates of Dodoma. (-6.206458,35.678533).

And finally the distance from Lake Victoria situated at the north of Tanzania will be also added as a variable.

I tried a couple of packages meant to calculate the haversine distance but had problems so I decided to do all the steps using a pipe. (The explanation of how the haversine distance is calculated is out of the scope of this report)

```
train1_Imputed%<>%mutate( distance = sqrt(longitude^2 + latitude^2))
```

```r
#Create haversine distance from Ocean Shore

deg2rad <- function(deg) {(deg * pi) / (180)}

R <- 6371 # radius of the Earth in Km
lat2<- -6.230069
longi2<-39.034139

phi2=deg2rad(lat2)


train1_Imputed<-
  train1_Imputed %>%
  mutate(phi1=deg2rad(latitude))%>%
  mutate(delta_phi=deg2rad(lat2-latitude))%>%
  mutate(delta_lambda = deg2rad(longi2 - longitude))%>%
  mutate(p1=sin(delta_phi / 2)**2)%>%
  mutate(p2=cos(phi1) * cos(phi2) *sin(delta_lambda / 2)**2)%>%
  mutate(p3=p1+p2)%>%
  mutate(x=sqrt(p3))%>%
  mutate(y=sqrt(1-p3))%>%
  mutate(p4 = 2*atan2 (x,y) )%>%
   mutate(haversinedist_Ocean= R*p4)%>%
  select(-c(phi1,delta_phi,delta_lambda,p1,p2,p3,x,y,p4))









#Create haversine distance from Dodoma

deg2rad <- function(deg) {(deg * pi) / (180)}

R = 6371 # radius of the Earth in Km
lat2<- -6.206458
longi2<-35.678533

phi2=deg2rad(lat2)


train1_Imputed<-train1_Imputed %>%
  mutate(phi1=deg2rad(latitude))%>%
  mutate(delta_phi=deg2rad(lat2-latitude))%>%
  mutate(delta_lambda = deg2rad(longi2 - longitude))%>%
  mutate(p1=sin(delta_phi / 2)**2)%>%
  mutate(p2=cos(phi1) * cos(phi2) *sin(delta_lambda / 2)**2)%>%
```

```
  mutate(p3=p1+p2)%>%
  mutate(x=sqrt(p3))%>%
  mutate(y=sqrt(1-p3))%>%
  mutate(p4 = 2*atan2 (x,y) )%>%
   mutate(haversinedist_Dodoma= R*p4)%>%
  select(-c(phi1,delta_phi,delta_lambda,p1,p2,p3,x,y,p4))




#Create haversine distance from Lake Victoria

deg2rad <- function(deg) {(deg * pi) / (180)}

R = 6371 # radius of the Earth in Km
lat2<- -2.069089
longi2<-32.432750

phi2=deg2rad(lat2)


train1_Imputed<-train1_Imputed %>%
  mutate(phi1=deg2rad(latitude))%>%
  mutate(delta_phi=deg2rad(lat2-latitude))%>%
  mutate(delta_lambda = deg2rad(longi2 - longitude))%>%
  mutate(p1=sin(delta_phi / 2)**2)%>%
  mutate(p2=cos(phi1) * cos(phi2) *sin(delta_lambda / 2)**2)%>%
  mutate(p3=p1+p2)%>%
  mutate(x=sqrt(p3))%>%
  mutate(y=sqrt(1-p3))%>%
  mutate(p4 = 2*atan2 (x,y) )%>%
   mutate(haversinedist_Victorialake= R*p4)%>%
  select(-c(phi1,delta_phi,delta_lambda,p1,p2,p3,x,y,p4))
```

#GPS Height

From previous analysis it was observed that the "gps_height" variable includes over 2000 distinct values.

Again, leveraging data available in Google : Lowest point Indian Ocean 0 metres (0 ft)

it is highly improbable that any pump will be placed at the sea or at the beach so any zero and negative values in this variable will be *NA*.

```
train1_Imputed <- train1_Imputed %>%
  mutate(gps_height = ifelse(gps_height <= 0, NA, gps_height))

test1_Imputed <- test1_Imputed %>%
  mutate(gps_height = ifelse(gps_height <= 0, NA, gps_height))




tbl1<-train1_Imputed %>%
```

```
dplyr::count(gps_height)%>%
arrange(-n)%>% head(10)

tbl1
```

Now, after having taken care of the *outliers* in the most important geo-variables, considering the variable importance ranking in our last RF model the next variables to deal with will be "date_recorded" and "construction_year". First the variable "construction_year" will be *fixed* since I'm gonna try to extract more info from the date variable using the "construction_year" variable, in which the present zero values must be converted to *NA* and be imputed.

#Construction Year

```
train1_Imputed <- train1_Imputed %>%
  mutate(construction_year = ifelse(construction_year == 0, NA, construction_year))



test1_Imputed <- test1_Imputed %>%
  mutate(construction_year = ifelse(construction_year == 0, NA, construction_year))
```

Now the two variables with *NA* values , "longitude" and "construction_year" will be imputed using the powerful missRanger library.

```
#Concatenate the two dataframes for the imputation process
df1_Imputed<-plyr::rbind.fill(train1_Imputed, test1_Imputed)%>%select(-status_group)



# Store the dataframe with the newly imputed construction_year and longitude variables
df2_Imputed <- missRanger(df1_Imputed, pmm.k = 3, num.trees = 100)
```

```
##
## Missing value imputation by random forests
##
##   Variables to impute:       gps_height, longitude, construction_year, distance, haversinedist_Ocean
##   Variables used to impute:  id, amount_tsh, date_recorded, funder, gps_height, installer, longitude
## iter 1:  .......
## iter 2:  .......
## iter 3:  .......
## iter 4:  .......
## iter 5:  .......
```

```
df2_Imputed%>% group_by(construction_year) %>% tally()
```

```
## # A tibble: 54 x 2
##    construction_year     n
##                <int> <int>
##  1              1960   124
##  2              1961    28
##  3              1962    36
##  4              1963   107
##  5              1964    48
##  6              1965    21
```

```
##  7              1966    19
##  8              1967   107
##  9              1968    93
## 10              1969    78
## # ... with 44 more rows
```

The train and test dataset are separated again to be used in the random forest algorithm.

As a reminder:

- df2_Imputed ==> contains all train and test features (- labels) after the imputation of the logical variables+Longitude+construction_year
- train2_Imputed ==> contains train dataframe features after the imputation of the logical variables+Longitude+construction_year
- test2_Imputed ==> contains test dataframe features after the imputation of the logical variables+Longitude+construction_year

```
#contains only the imputed features
train2_Imputed<- df2_Imputed%>% filter(subset=='train')
test2_Imputed<- df2_Imputed%>% filter(subset=='test')
```

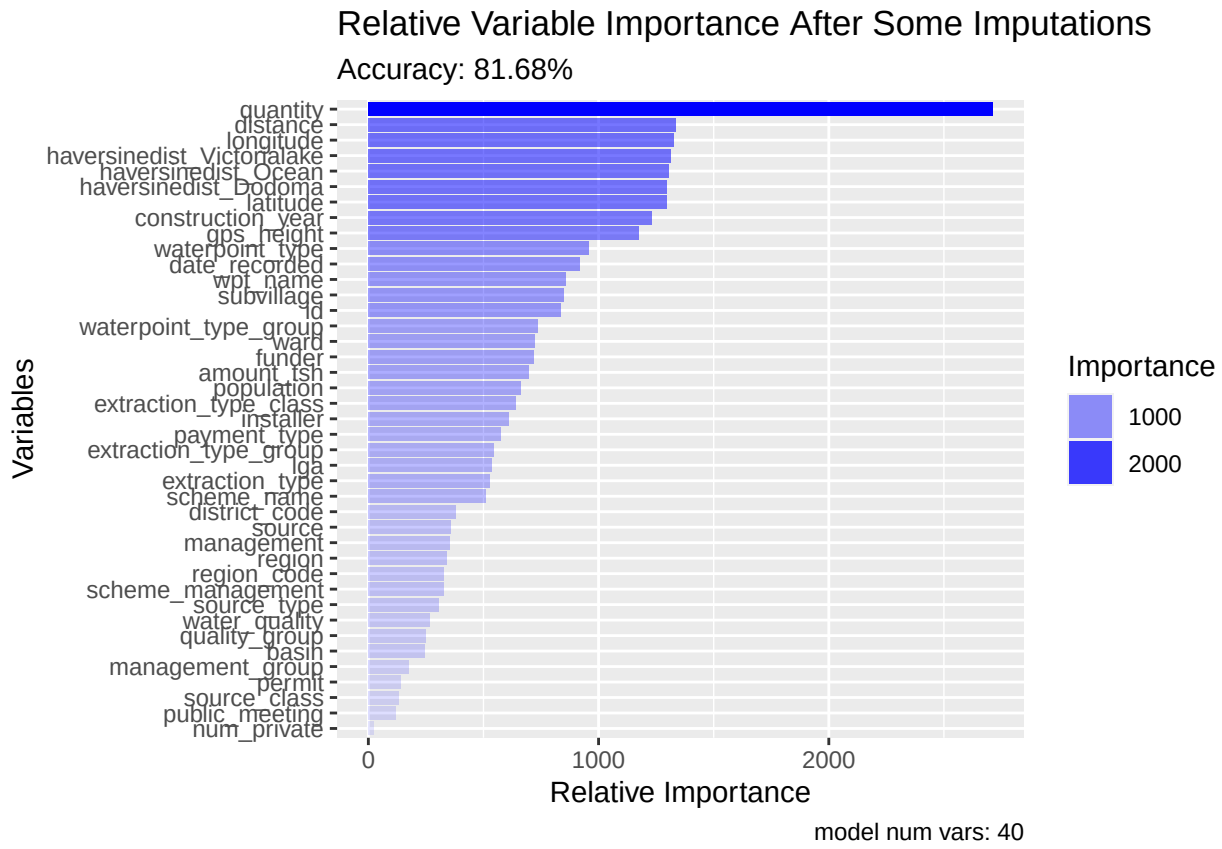
```
train2_Imputed <- merge(x = train2_Imputed, y = train_labels0, by = "id", all = TRUE)
```

```
#target should be of type factor
train2_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train2_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor
```

```
mymodel <- ranger(
                   status_group ~ .,
                   data      = mydata  ,
                   num.trees = 500,
                   mtry      = 3,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()
```

```r
#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Some Imputations",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

The local score has improved a bit. It seems like the model "likes" the new distance related variables since they all rank pretty high. The "distance" variable now ranks second, just behid the "quantity" variable.

#— Confusion Matrix. #– Predicted Values vs. Actual values

```
mymodel$confusion.matrix
```

Count-wise,the model tends to miss-classify the *functional* category more than the other two categories. Although when looked as a percentage the model shows the highest percent of miss-classification in the *functional needs repair* class where for every 7 correct predictions it almost 35 wrong predictions are done, so that's way worse than tossing a coin (50% chance). At a first sight, It could be said that the current model lacks sensitivity and that's probably the reason it predicts so bad the category with the least frequency.

#— Submission #– Prediction

```
pred_val <- predict( mymodel , data = test2_Imputed)$predictions
head(pred_val)
```

#– Prepare submission

```
sub_df <- data.frame(
  id = df_test0$id,
  status_group = pred_val
)
```

#– Save submission

```
fwrite(sub_df, "./MySubmissions/ranger_8168.csv", nThread = 3 )
```

-Local score: 0.8168 -Submission : 0.8096

Unfortunately there is overfitting and the validation score has not improved.

#Date Variable

Thanks to the *lubridate* library, a new variable "operation_years" can be obtained by substracting the "construction_year" from the "date_recorded" year. Although since the construction year variable has a lot of zeros I will first impute the *NA* s that I've got till now.

```
#su enero es verano


#convert date to year,month,day format using the lubridate
#library

df2_Imputed<- df2_Imputed %>% mutate(date_recorded = ymd(date_recorded)) %>%

  #extract the year portion from the date variable and substract it from the construction year
  mutate(operation_years = lubridate::year(date_recorded) - construction_year) %>%

  #if the operation year is a neagtive number convert it to NA
  mutate(operation_years = ifelse(operation_years < 0, NA, operation_years))
```

In addition, I usually like to have a season variable since it catches some granularity ommited by year and month.

```
df2_Imputed <- df2_Imputed %>%
  mutate(day_of_year = yday(date_recorded)) %>%
  mutate(month_recorded = lubridate::month(date_recorded)) %>%
  mutate(year_recorded= lubridate::year(date_recorded))%>%
  mutate(season = ifelse( month_recorded <= 3, "Summer",
                          ifelse( month_recorded <= 6 , "Autumn",
                                  ifelse(month_recorded <= 9, "Winter", "Spring"))))



#Create variable that determines antiquity in days, and also in years
# and then drop the original date variable

df2_Imputed <-  df2_Imputed %>%
  mutate( antiquity_days = max(date_recorded) - date_recorded)%>%
  select(-date_recorded)



# #contains only the imputed features+ the new date variables
train2_Imputed<- df2_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test2_Imputed<- df2_Imputed%>% filter(subset=='test')
```

Our current model has a lot of variables, and we even haven't started creating new ones yet. Probably some of them will be very similar so leaving out one of them won't do any harm.

It's worth mentioning, that this manual depuration is a privilege only allowed as far as our dataframe has a reasonable number of variables.

extraction_type, extraction_type_group, extraction_type_class

This three variables were already discussed in the visualization stage but no decision was made about them.

note: tally() is a convenient wrapper for summaries that will either call n or sum(n)

```
df2_Imputed%>%
  group_by(extraction_type_class, extraction_type_group, extraction_type) %>% tally()
```

```
## # A tibble: 18 x 4
## # Groups:   extraction_type_class, extraction_type_group [13]
##    extraction_type_class extraction_type_group extraction_type              n
##    <chr>                 <chr>                 <chr>                     <int>
##  1 gravity               gravity               gravity                   33263
##  2 handpump              afridev               afridev                    2208
##  3 handpump              india mark ii         india mark ii              3029
##  4 handpump              india mark iii        india mark iii              135
##  5 handpump              nira/tanira           nira/tanira               10205
##  6 handpump              other handpump        other - mkulima/shinyanga     2
##  7 handpump              other handpump        other - play pump           101
##  8 handpump              other handpump        other - swn 81              284
##  9 handpump              other handpump        walimi                       60
## 10 handpump              swn 80                swn 80                     4588
## 11 motorpump             mono                  mono                       3628
## 12 motorpump             other motorpump       cemo                        108
## 13 motorpump             other motorpump       climax                       41
## 14 other                 other                 other                      8102
## 15 rope pump             rope pump             other - rope pump           572
## 16 submersible           submersible           ksb                        1790
## 17 submersible           submersible           submersible                5982
## 18 wind-powered          wind-powered          windmill                    152
```

Both the three variables, have the "gravity" category as their most frequent one.

The main discussion is between "extraction_type_group" and "extraction_type". They share some categories with the only difference that it seems like the categories in the "extraction_type" variable refer to the brand or model.

In addition, they both have almost the same importance in ther Relative Importance chart obtained by means of running a random forest model.

So, some of the categories in the "extraction_type" will be recategorized and some of them will be renamed and then the variable "extraction_type_group" will be dropped from the dataset.

We bind the train and test data for the lumping process, because otherwise in the validation step the model will find some "strange" categories and it will not know how to proceed.

```
df2_Imputed <- df2_Imputed  %>%
    mutate(extraction_type = revalue(extraction_type,
                                     c("cemo" = "other motorpump",
                                       "climax" = "other motorpump",
                                       "other - mkulima/shinyanga" = "other handpump",
                                       "other - play pump" = "other handpump",
                                       "walimi" = "other handpump",
```

```
                                      "other - swn 81" = "swn",
                                      "swn 80" = "swn",
                                      "india mark ii" = "india mark",
                                      "india mark iii" = "india mark"))) %>%
    select( - extraction_type_group )
```

Tally provides a nice view of our data so why don't use it again?

```
df2_Imputed%>%
  group_by(extraction_type_class, extraction_type) %>% tally()
```

```
## # A tibble: 13 x 3
## # Groups:   extraction_type_class [7]
##    extraction_type_class extraction_type       n
##    <chr>                 <chr>             <int>
##  1 gravity               gravity           33263
##  2 handpump              afridev            2208
##  3 handpump              india mark         3164
##  4 handpump              nira/tanira       10205
##  5 handpump              other handpump      163
##  6 handpump              swn                4872
##  7 motorpump             mono               3628
##  8 motorpump             other motorpump     149
##  9 other                 other              8102
## 10 rope pump             other - rope pump   572
## 11 submersible           ksb                1790
## 12 submersible           submersible        5982
## 13 wind-powered          windmill            152
```

Now this looks much more descriptive, neat and beautiful. Gravity is still the same in both but now we can see to which type of extraction do the different brands of pump belong.

Rest of Geographic Variables: region, region_code, district_code, ward, subvillage, lga

```
df2_Imputed %>% group_by(region, region_code, district_code) %>% tally()
```

The "region" variable will be kept as it is. The same "district_code" appears in different region names, so it will be assumed that this variable indicates a smaller unit within each region

```
sapply(subset(df2_Imputed, select = c(region, subvillage,ward)),function(x) length(unique(x)))
#data_Imputed %>% group_by(region, subvillage,ward ) %>% tally()
```

So, probably in increasing degree of precision, the geographic information is given by: - region (or region_code) - district_code within region - ward - subvillage - longitude x latitude

{Since there are no missing values in the region and district columns, I tried to substitute missing individual long/lat values using the KNN means but an error of : "Not sufficient complete cases for computing neighbors." appeared}

#lga

```
df2_Imputed %>% group_by(lga) %>% tally()
```

Some of the values in this variable are divided into Rural and Urban.

I will create a new variable where "lga_3" is transformed into a new feature that takes three categories: rural, urban and other.

```
df2_Imputed <- df2_Imputed %>% mutate(lga_3 = ifelse( grepl(" rural", lga), "rural",ifelse( grepl(" urba
```

---

scheme_management, scheme_name

```
df2_Imputed %>% group_by(scheme_management) %>% tally()
```

```
## # A tibble: 13 x 2
##    scheme_management      n
##    <chr>              <int>
##  1 ""                  4846
##  2 "company"           1341
##  3 "none"                 1
##  4 "other"              996
##  5 "parastatal"        2124
##  6 "private operator"  1326
##  7 "swc"                123
##  8 "trust"               92
##  9 "vwc"              45917
## 10 "water authority"   3975
## 11 "water board"       3462
## 12 "wua"               3551
## 13 "wug"               6496
```

```
levels(as.factor(df2_Imputed $scheme_management))
```

```
##  [1] ""                 "company"          "none"             "other"
##  [5] "parastatal"       "private operator" "swc"              "trust"
##  [9] "vwc"              "water authority"  "water board"      "wua"
## [13] "wug"
```

The variable "scheme_management" has 13 distinct levels. There is a level called "" which will be replaced by *NA*. Apart from that, there are some under-represented categories which we will recategorize to "other".

keywords: VWC: village water committee, SWC: State Water Committee

```
#Substitute empty strings by "NA"

df2_Imputed <- df2_Imputed %>%
  mutate(scheme_management = ifelse(scheme_management == "", NA, scheme_management))



#Join the two categories with very low representations
# to Other

df2_Imputed %<>% mutate(scheme_management= revalue(scheme_management,
  c("none"="other",
```

```
    "trust"="other"
)))
```

```
df2_Imputed %>% group_by(scheme_name) %>% tally()
```

This variables will be dropped since *NA*s make up for most of the incidences, and the rest of categories have very low frequency.

```
#drop variable scheme_name
df2_Imputed %<>%select(-scheme_name)
```

---

water_quality, quality_group

```
df2_Imputed %>% group_by(quality_group, water_quality) %>% tally()
```

Both have the same same exact levels but I prefer the naming of the "water_quality" group better since it seems more descriptive, therefore the "quality_group" variable will be dropped.

```
df2_Imputed%<>%select(-quality_group)
```

source, source_type, source_class

```
df2_Imputed %>% group_by(source_class, source_type, source) %>% tally()
```

The three variables seems to catch different levels of granularity, becoming more precise from left to right in the table. Curiously, the importance increases in the reverse order.

Anyway, the *unknown* category in the "source_class" variable will be relabeled as *NA*.

```
df2_Imputed <- df2_Imputed %>%
  mutate(source_class = ifelse(source_class == "unknown", NA, source_class))


df2_Imputed <- df2_Imputed %>%
  mutate(source = ifelse(source  == "unknown", NA, source))
```

#waterpoint_type, waterpoint_type_group

```
df2_Imputed %>% group_by(waterpoint_type_group, waterpoint_type) %>% tally()
```

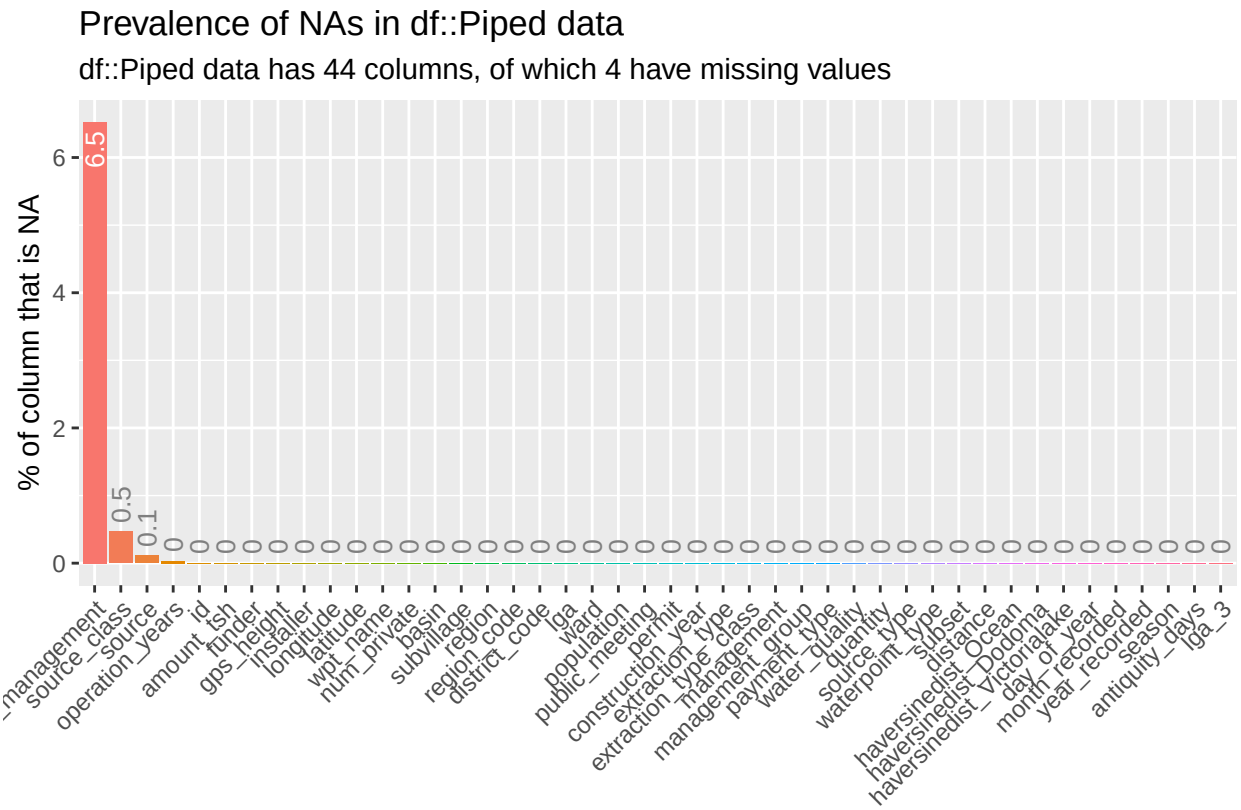
These two variables have the exact same categories but with different names. So "waterpoint_type_group" will be dropped from the dataset.

This leaves us with 34 predictors.

```
df2_Imputed %<>% select(-waterpoint_type_group)
```

Check for *NA*s to decide if we need to run an imputation process again

```
df2_Imputed %>%inspect_na()%>%show_plot()
```

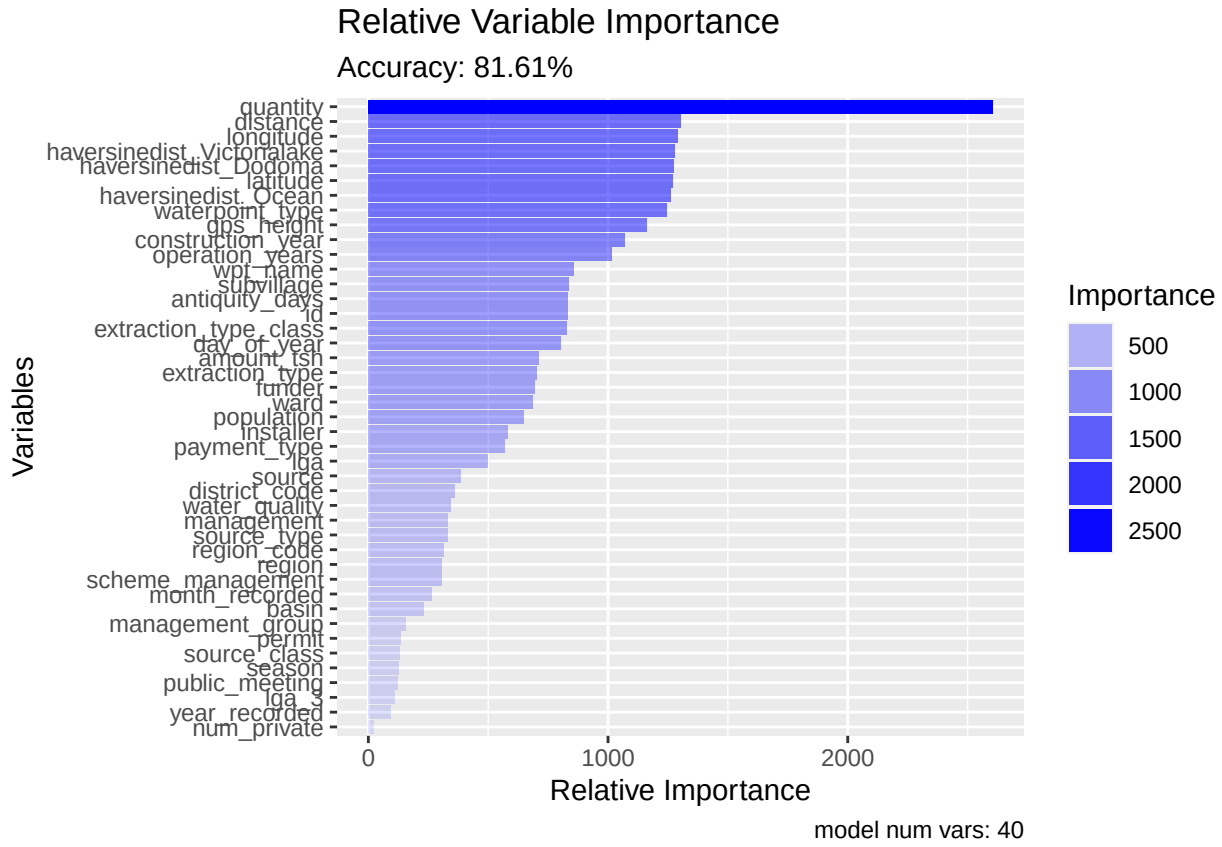


There are some *NA*s in the model so we run an imputation before running the model.

```
df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)



# #contains only the imputed features+ the new date variables
train3_Imputed<- df3_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed<- df3_Imputed%>% filter(subset=='test')



train3_Imputed <- merge(x = train3_Imputed, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor
```

```
mymodel <- ranger(
                    status_group ~ .,
                    data      = mydata  ,
                    num.trees = 500,
                    mtry      = 3,
                    importance="impurity",
                    seed=123
                  )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

Strangely, the local scored has decreased after the depuration of some variables.

It can also be seen that the "operation_years" variable has more importance than the other date-related variables.

Again, the geo-variables make up a good part of the top ten.

#— Confusion Matrix. #– Predicted Values vs. Actual values

```
mymodel$confusion.matrix
```

Nothing has changed drastically.

#—— Submission #– Prediction

```
pred_val <- predict( mymodel , data = test3_Imputed)$predictions
head(pred_val)
```

#– Save submission

```
fwrite(sub_df, "./MySubmissions/ranger_08152.csv", nThread = 3 )
```

- Local Score: 0.8152
- Submission Score:

#funder, installer, wpt_name

These three variables have a lot of different categories. In this case joining categories manually is just impossible. So here I will do a lumping in which I will join categories with less that a number of occurrences as "other". The question is: "how many levels should I decide to keep for each one?" Unfortunately the

answer is that there is no exact answer and it's a matter of trial and error and see which one improves the model more.

In addition I am gonna recategorize the empty strings and zero values by "other". I could also substitute them by *NA*s but that will add the necessity for yet another imputation step.

I'm gonna try a couple of possibilities and see which one works better.

```
cbind(
  df3_Imputed%>% group_by(funder) %>% tally() %>% arrange(desc(n)) %>% slice(1:50),
  df3_Imputed %>% group_by(installer) %>% tally() %>% arrange(desc(n)) %>% slice(1:50),
  df3_Imputed %>% group_by(wpt_name) %>% tally() %>% arrange(desc(n)) %>% slice(1:50)
)
```

Reduce the number of levels of a categorical variable by grouping the smaller levels into "other".

#try 1: 50 levels

```
#We had already converted all levels to lowercase

df3_Imputed<-df3_Imputed%>%
  #mutate( funder = stri_trans_tolower(funder)) %>%
  #funder variable
  # remove spaces
  mutate( funder = stri_replace_all_fixed(funder, " ", "")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( funder = stri_replace_all_fixed(funder,
                                          c("0", "notknown") ,
                                          c('unknown'),
                                          vectorise_all = FALSE ) ) %>%
  mutate( funder = ifelse(nchar(funder) == 0, "unknown", funder)) %>%

  # installer variable
    mutate( installer= stri_replace_all_fixed(installer, " ", "")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( installer = stri_replace_all_fixed(installer,
                                          c("0", "notknown") ,
                                          c('unknown'),
                                          vectorise_all = FALSE ) ) %>%
  mutate( installer = ifelse(nchar(installer) == 0, "unknown", installer)) %>%

    # wpt_name variable
  mutate( wpt_name= stri_replace_all_fixed(wpt_name, " ","")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( wpt_name = stri_replace_all_fixed(wpt_name,
                                          c("0", "notknown") ,
                                          c('unknown'),
                                          vectorise_all = FALSE ) ) %>%
  mutate( wpt_name = ifelse(nchar(wpt_name) == 0, "unknown", wpt_name)) %>%


 # keep only 50 categories

   mutate( funder_redux50 = fct_lump_n(as.factor(funder), n = 50)) %>%
  mutate( installer_redux50 = fct_lump_n(as.factor(installer), n = 50)) %>%
```

```
 mutate( wpt_name_redux50 = fct_lump_n(as.factor(wpt_name), n = 50)) #%>%
```

Of these funder and installer have a few large categories (more than 500 instances), so I keep those and group their smaller categories under other. I decide to keep only 20

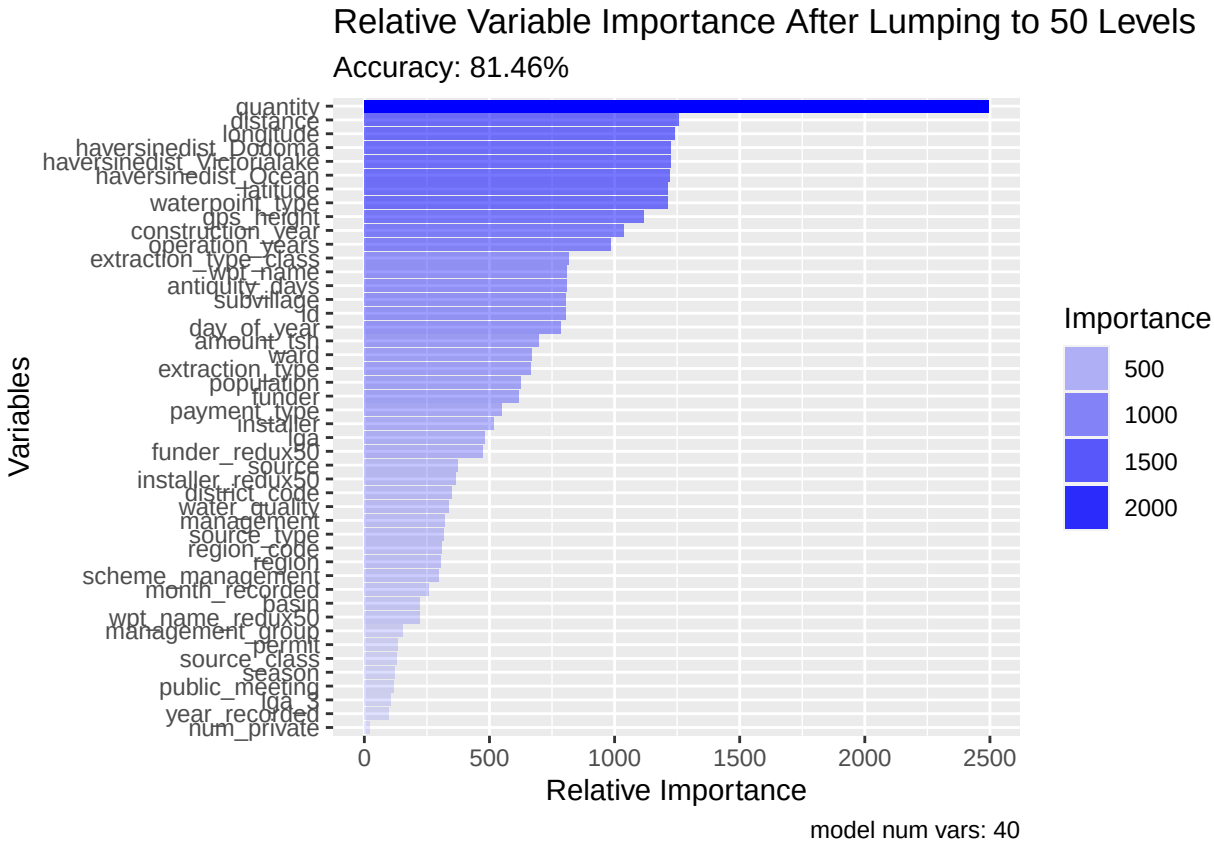
```
cbind(
  df3_Imputed%>% group_by(funder_redux50) %>% tally() %>% arrange(desc(n)) %>% slice(1:10),
  df3_Imputed %>% group_by(installer_redux50) %>% tally() %>% arrange(desc(n)) %>% slice(1:10),
  df3_Imputed %>% group_by(wpt_name_redux50) %>% tally() %>% arrange(desc(n)) %>% slice(1:10)
)
```

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed<- df3_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed<- df3_Imputed%>% filter(subset=='test')



train3_Imputed <- merge(x = train3_Imputed, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                   status_group ~ .,
                   data      = mydata  ,
                   num.trees = 500,
                   mtry      = 3,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

The lumping with 50 levels has slightly decreased the local score.

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()
```

```r
#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Lumping to 50 Levels",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

The "funder_redux50" and "installer_redux50" variables are pretty close in importance to the original one although "wpt_name_redux50" is doing way worse that its original counterpart.

#try 2: 150 levels

```
df3_Imputed<-df3_Imputed%>%

  #funder variable
  # remove spaces
  mutate( funder = stri_replace_all_fixed(funder, " ", "")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( funder = stri_replace_all_fixed(funder,
                                          c("0", "notknown") ,
                                          c('unknown'),
                                          vectorise_all = FALSE ) ) %>%
  mutate( funder = ifelse(nchar(funder) == 0, "unknown", funder)) %>%

  # installer variable
    mutate( installer= stri_replace_all_fixed(installer, " ", "")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( installer = stri_replace_all_fixed(installer,
                                          c("0", "notknown") ,
                                          c('unknown'),
                                          vectorise_all = FALSE ) ) %>%
```

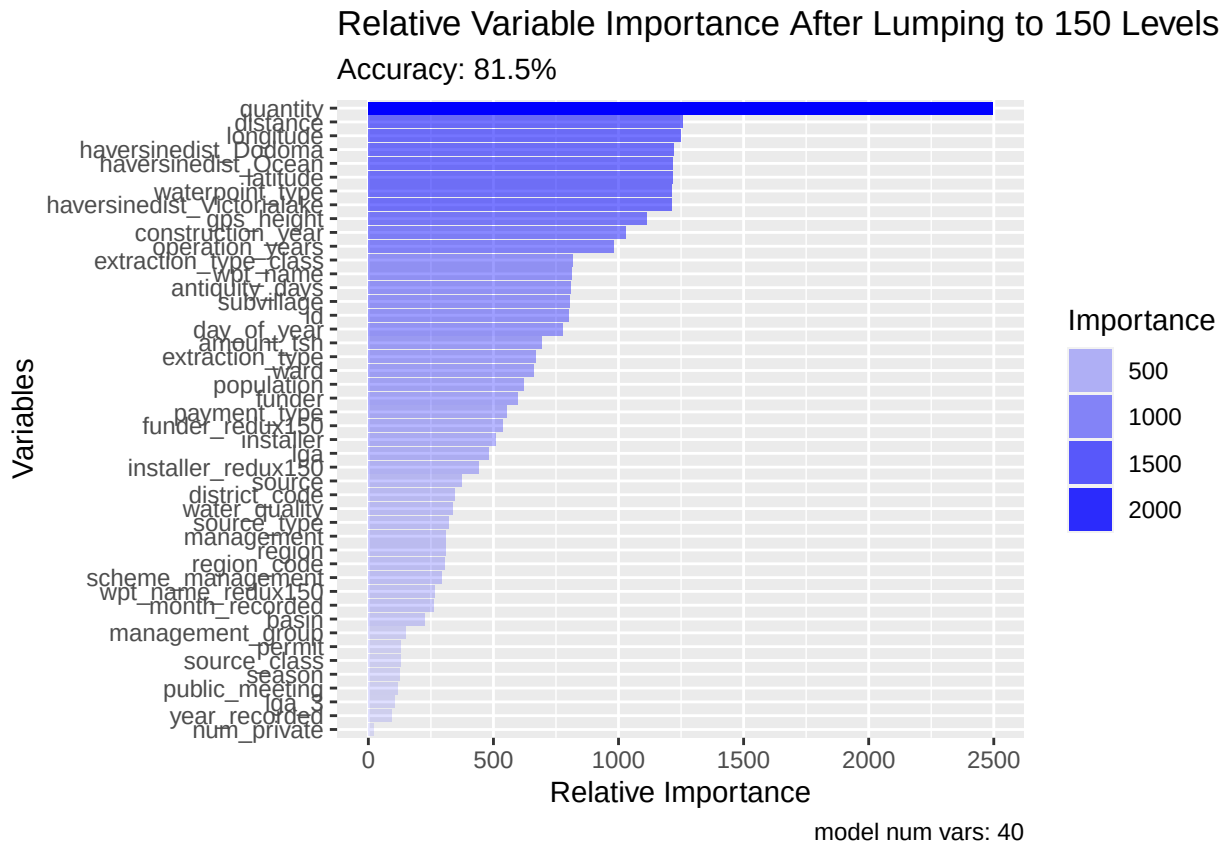
```
  mutate( installer = ifelse(nchar(installer) == 0, "unknown", installer)) %>%

    # wpt_name variable
  mutate( wpt_name= stri_replace_all_fixed(wpt_name, " ","")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( wpt_name = stri_replace_all_fixed(wpt_name,
                                            c("0", "notknown") ,
                                            c('unknown'),
                                            vectorise_all = FALSE ) ) %>%
  mutate( wpt_name = ifelse(nchar(wpt_name) == 0, "unknown", wpt_name)) %>%


 # keep only 50 categories

   mutate( funder_redux150 = fct_lump_n(as.factor(funder), n = 150)) %>%
  mutate( installer_redux150 = fct_lump_n(as.factor(installer), n = 150)) %>%
 mutate( wpt_name_redux150 = fct_lump_n(as.factor(wpt_name), n = 150)) %>%
 select(-c(funder_redux50,installer_redux50,wpt_name_redux50))
```

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed<- df3_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed<- df3_Imputed%>% filter(subset=='test')



train3_Imputed <- merge(x = train3_Imputed, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                  status_group ~ .,
                  data      = mydata  ,
                  num.trees = 500,
                  mtry      = 3,
                  importance="impurity",
                  seed=123
                )

error_val <- mymodel$prediction.error
error_val
```

```
accu_val <- 1 - error_val
accu_val
```

Better than with 50 levels but still worse than before lumping.

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Lumping to 150 Levels",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

Still none of the *redux150* varaibles is doing better than the original ones.

#try 3: 500 levels

```
df3_Imputed<-df3_Imputed%>%

  #funder variable
  # remove spaces
  mutate( funder = stri_replace_all_fixed(funder, " ", "")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( funder = stri_replace_all_fixed(funder,
                                          c("0", "notknown") ,
                                          c('unknown'),
                                          vectorise_all = FALSE ) ) %>%
  mutate( funder = ifelse(nchar(funder) == 0, "unknown", funder)) %>%

  # installer variable
    mutate( installer= stri_replace_all_fixed(installer, " ", "")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( installer = stri_replace_all_fixed(installer,
                                          c("0", "notknown") ,
                                          c('unknown'),
                                          vectorise_all = FALSE ) ) %>%
  mutate( installer = ifelse(nchar(installer) == 0, "unknown", installer)) %>%
```

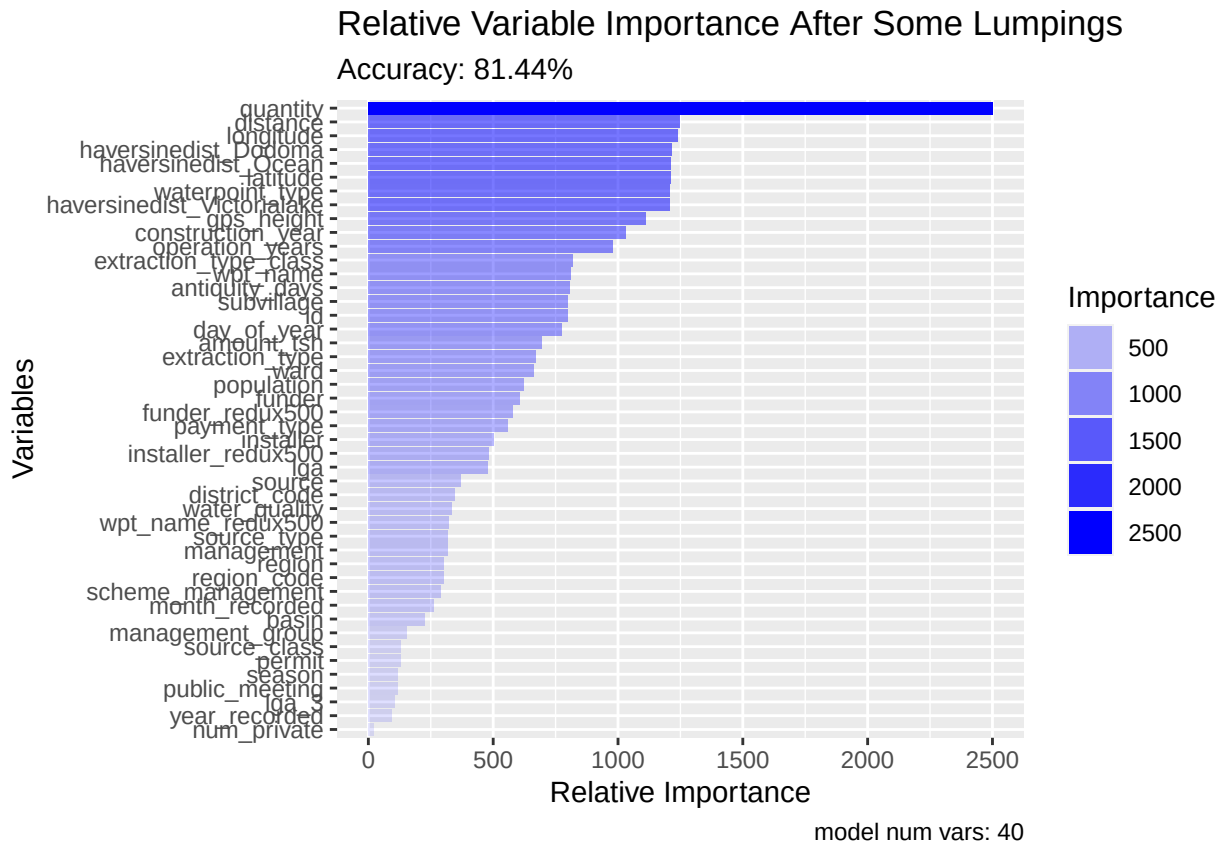
```
    # wpt_name variable
  mutate( wpt_name= stri_replace_all_fixed(wpt_name, " ","")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( wpt_name = stri_replace_all_fixed(wpt_name,
                                            c("0", "notknown") ,
                                            c('unknown'),
                                            vectorise_all = FALSE ) ) %>%
  mutate( wpt_name = ifelse(nchar(wpt_name) == 0, "unknown", wpt_name)) %>%


 # keep only 50 categories

   mutate( funder_redux500 = fct_lump_n(as.factor(funder), n = 500)) %>%
  mutate( installer_redux500 = fct_lump_n(as.factor(installer), n = 500)) %>%
 mutate( wpt_name_redux500 = fct_lump_n(as.factor(wpt_name), n = 500)) %>%
 select(-c(funder_redux150,installer_redux150,wpt_name_redux150))
```

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed<- df3_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed<- df3_Imputed%>% filter(subset=='test')



train3_Imputed <- merge(x = train3_Imputed, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                  status_group ~ .,
                  data      = mydata  ,
                  num.trees = 500,
                  mtry      = 3,
                  importance="impurity",
                  seed=123
                )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
```

```
accu_val
```

Better than with 50 levels but still worse than before lumping.

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Some Lumpings",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

At least we have reached pre-lumping accuracy and now we've got less levels which leads toa simpler model. Anyway I will keep the 500 level "funder" and "installer" variables while dropping their counterparts.

```
df3_Imputed%<>%select(-c(funder,installer))
```

#try 3: 1000 levels for wpt_name

```
df3_Imputed<-df3_Imputed%>%

  # wpt_name variable
  mutate( wpt_name= stri_replace_all_fixed(wpt_name, " ","")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( wpt_name = stri_replace_all_fixed(wpt_name,
                                            c("0", "notknown") ,
                                            c('unknown'),
                                            vectorise_all = FALSE ) ) %>%
  mutate( wpt_name = ifelse(nchar(wpt_name) == 0, "unknown", wpt_name)) %>%


 # keep only 10000 categories
 mutate( wpt_name_redux1000 = fct_lump_n(as.factor(wpt_name), n = 1000)) %>%

 select(-c(wpt_name_redux500))
```

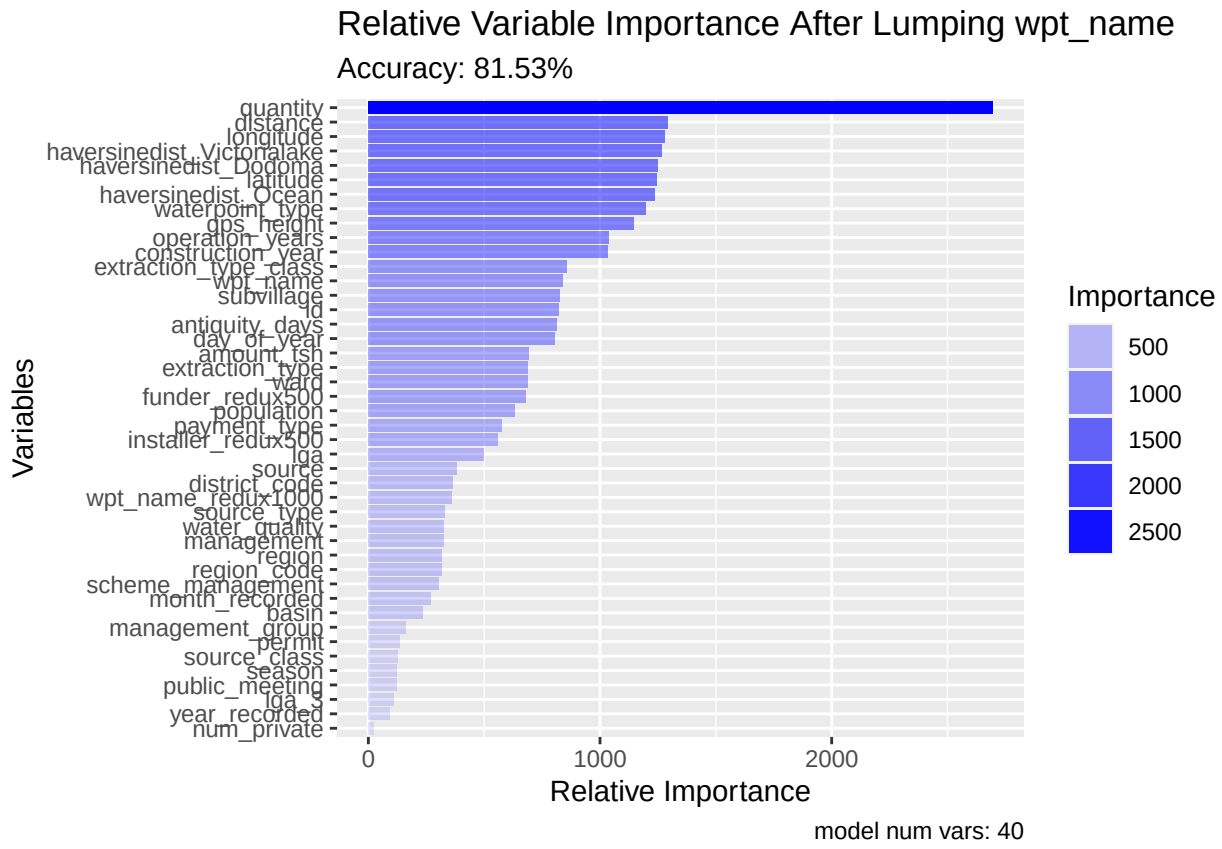
```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)
```

```
# #contains only the imputed features+ the new date variables
train3_Imputed<- df3_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed<- df3_Imputed%>% filter(subset=='test')



train3_Imputed <- merge(x = train3_Imputed, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                  status_group ~ .,
                  data      = mydata  ,
                  num.trees = 500,
                  mtry      = 3,
                  importance="impurity",
                  seed=123
                )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

Better than with 50 levels but still worse than before lumping.

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"
```

```
#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Lumping wpt_name",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

It was not possible to create a better variable from "wpt_name" using lumping.

Also it can be seen that by substituting the new lump variables the local score has improved.

(I keep both wpt_name variables cause if I remove the lumped one the score drops.)

```
sapply(Filter(is.character,df3_Imputed),function(x) length(unique(x)))
```

```
##              wpt_name                 basin             subvillage
##                 45095                     9                 21426
##                region                   lga                  ward
##                    21                   125                  2098
##    scheme_management       extraction_type extraction_type_class
##                    10                    13                     7
##            management      management_group          payment_type
##                    12                     5                     7
##         water_quality              quantity                source
##                     8                     5                     9
##           source_type          source_class       waterpoint_type
##                     7                     2                     7
##                subset                season                lga_3
##                     2                     4                     3
```

```
sapply(Filter(is.factor,df3_Imputed),function(x) length(unique(x)))
```

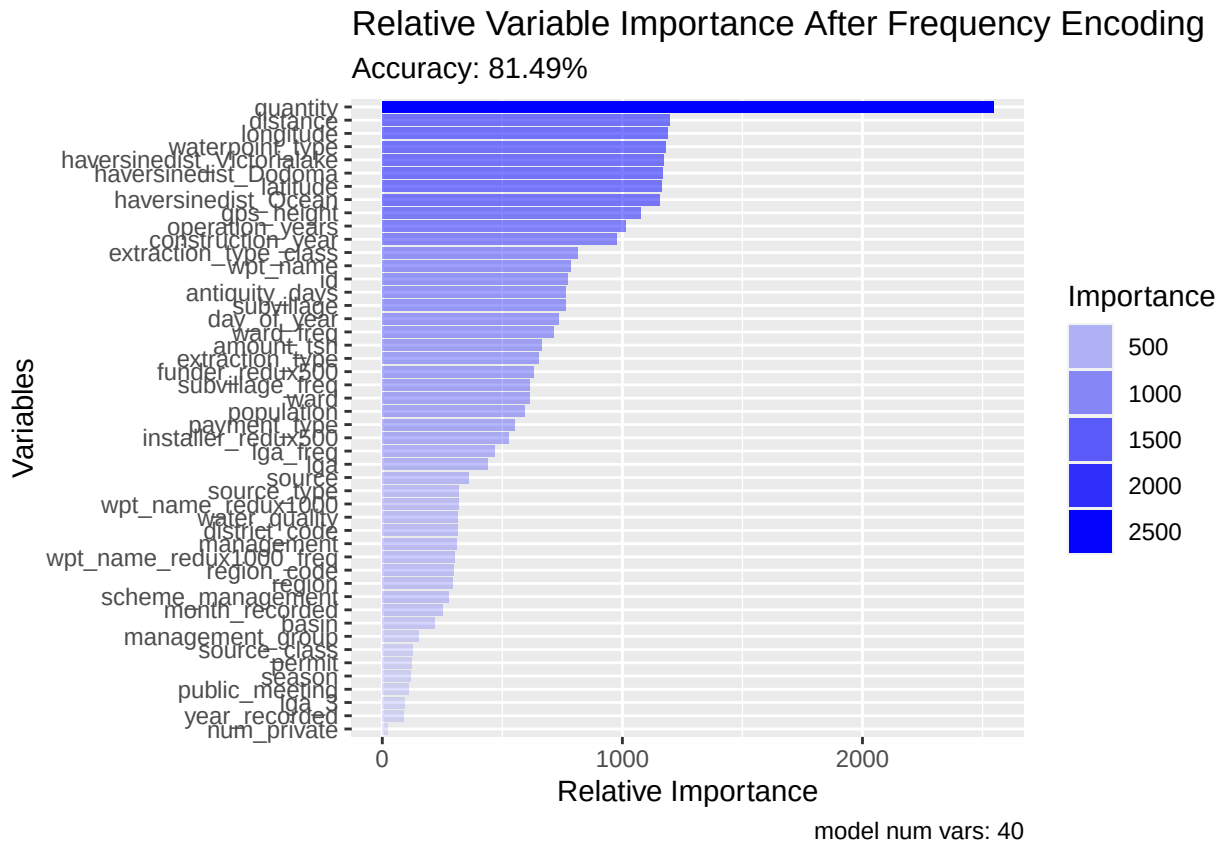
```
##    funder_redux500 installer_redux500 wpt_name_redux1000
##                501                512               1020
```

```
#-- FE - Variable nuevas....
numlga_dc <- df3_Imputed %>%
   group_by(lga) %>%
   dplyr::summarise( lga_freq = n() )

df3_Imputed %<>% left_join( numlga_dc )


numward_dc <- df3_Imputed %>%
   group_by(ward) %>%
   dplyr::summarise( ward_freq = n() )

df3_Imputed %<>% left_join( numward_dc )



numsubvillage_dc <- df3_Imputed %>%
   group_by(subvillage) %>%
   dplyr::summarise( subvillage_freq = n() )

df3_Imputed %<>% left_join( numsubvillage_dc )




numwpt_name_redux1000_dc <- df3_Imputed %>%
   group_by(wpt_name_redux1000 ) %>%
   dplyr::summarise( wpt_name_redux1000_freq = n() )

df3_Imputed %<>% left_join( numwpt_name_redux1000_dc )
```

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed<- df3_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed<- df3_Imputed%>% filter(subset=='test')



train3_Imputed <- merge(x = train3_Imputed, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                   status_group ~ .,
                   data      = mydata  ,
                   num.trees = 500,
                   mtry      = 3,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"
```

```
#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Frequency Encoding",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

"ward_frequency" has improved compared to "ward" but subvillage has not experienced the same effect(i will try a luming with subvillage).

Also "lga_freq" is a bit better than it's original.

#Lumping for lga and subvillage

```
df3_Imputed<-df3_Imputed%>%

  #funder variable
  # remove spaces
  mutate( lga = stri_replace_all_fixed(lga, " ", "")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( lga = stri_replace_all_fixed(lga,
                                       c("0", "notknown") ,
                                       c('unknown'),
                                       vectorise_all = FALSE ) ) %>%
  mutate( lga = ifelse(nchar(lga) == 0, "unknown", lga)) %>%

  # installer variable
    mutate( subvillage= stri_replace_all_fixed(subvillage, " ", "")) %>%
  # Convert: 0, "unknown" or "" -> "NA"
  mutate( subvillage = stri_replace_all_fixed(subvillage,
                                       c("0", "notknown") ,
                                       c('unknown'),
                                       vectorise_all = FALSE ) ) %>%
  mutate( subvillage = ifelse(nchar(subvillage) == 0, "unknown",subvillage)) %>%



   mutate( lga_redux25 = fct_lump_n(as.factor(lga), n =25)) %>%
  mutate( subvillage_redux500 = fct_lump_n(as.factor(subvillage), n = 500))
# select(-c(funder_redux150,installer_redux150,wpt_name_redux150))
```

Lump the newly created frequency variables

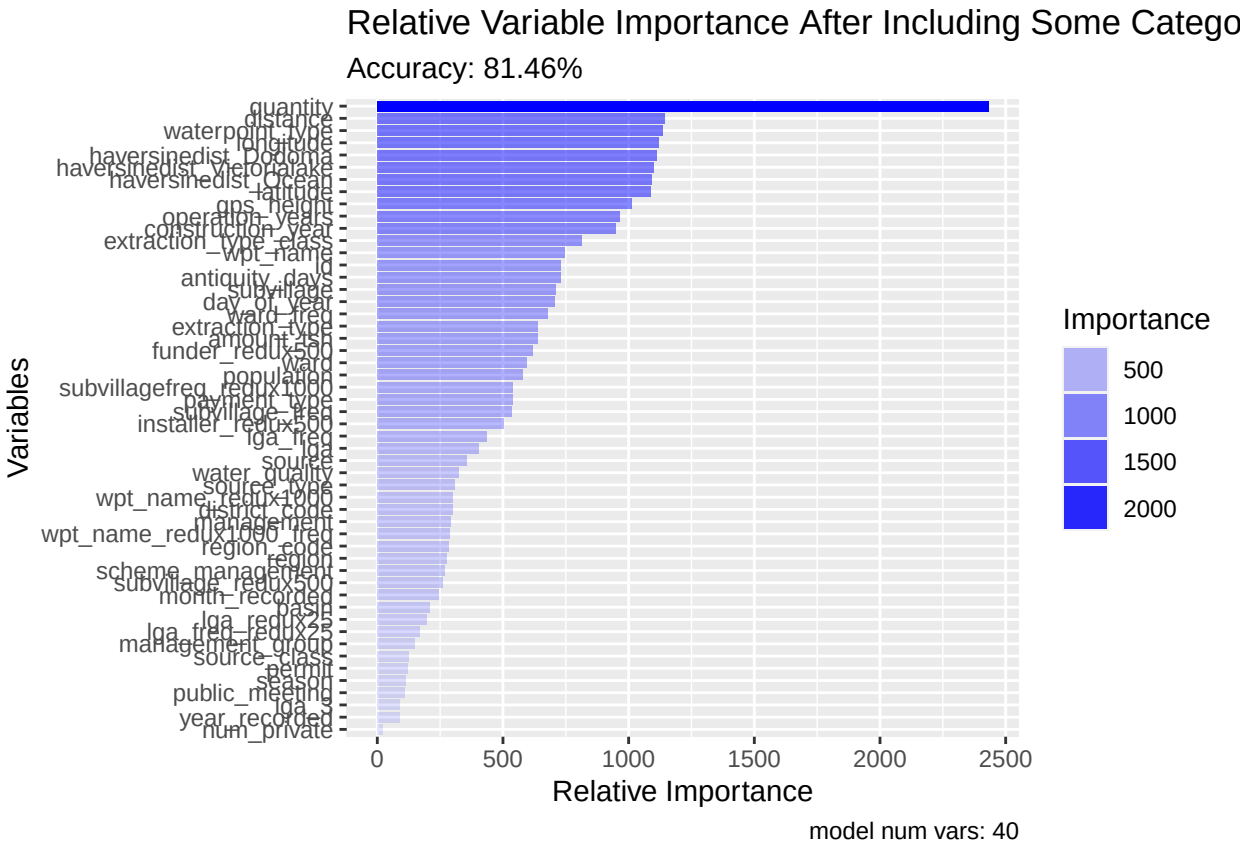
```
df3_Imputed<-df3_Imputed%>%


 # keep only 50 categories

   mutate( lga_freq_redux25 = fct_lump_n(as.factor(lga_freq), n =25)) %>%
  mutate( subvillagefreq_redux1000 = fct_lump_n(as.factor(subvillage_freq), n = 1000))
```

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed<- df3_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed<- df3_Imputed%>% filter(subset=='test')



train3_Imputed <- merge(x = train3_Imputed, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed%<>%mutate(status_group = as.factor(status_group))
```

```
mydata<-train3_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                   status_group ~ .,
                   data      = mydata  ,
                   num.trees = 500,
                   mtry      = 3,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Including Some Categorical Variables",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
```

```
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train) ,sep = "")
  )
```




```
    theme_classic()
```

It can be seen than lumping the frequency variable has no effect on its importance.

#Dropping the non-relevant new variables

Some of the variables created have worsened the model so they will be dropped.

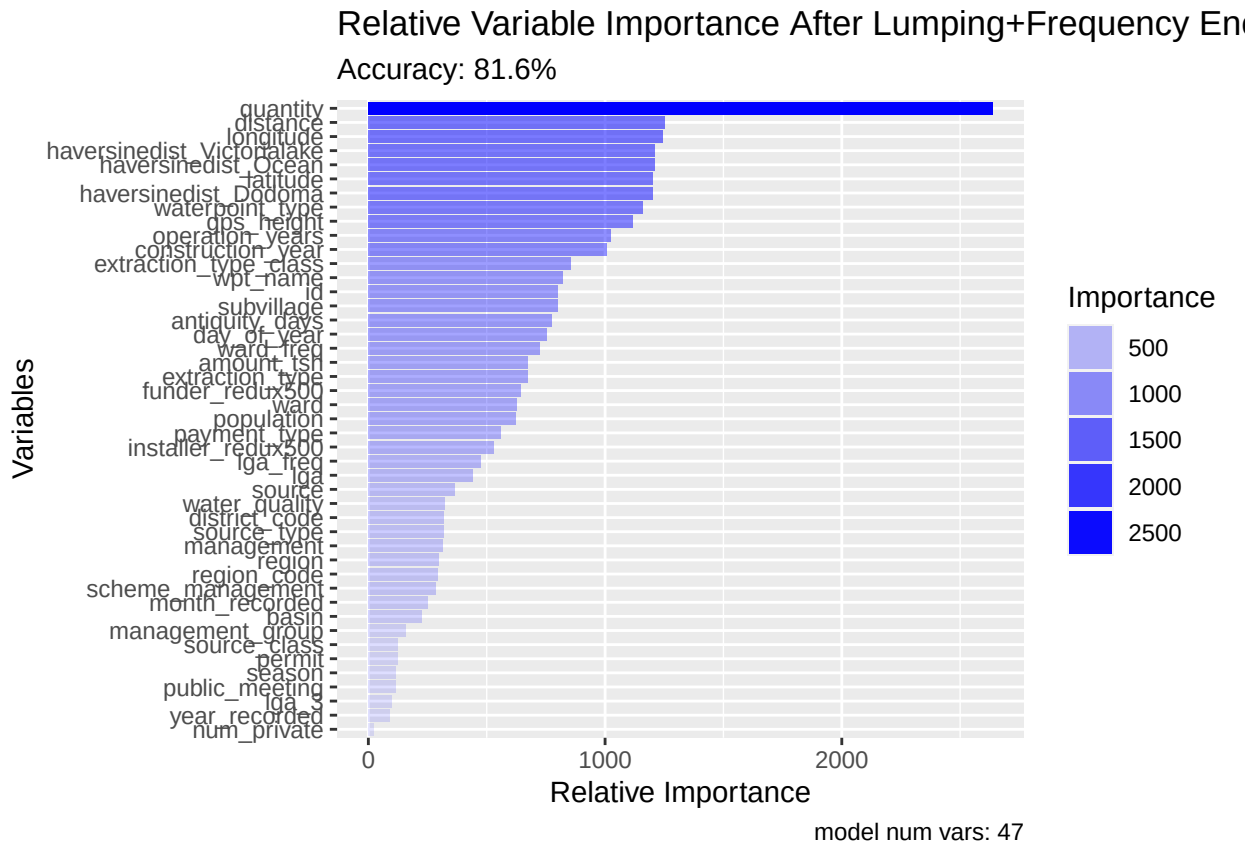
```
df3_Imputed%<>% select(-c(subvillagefreq_redux1000,lga_freq_redux25,subvillage_redux500,
                          lga_redux25,wpt_name_redux1000_freq,subvillage_freq , wpt_name_redux1000))
```

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed<- df3_Imputed%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed<- df3_Imputed%>% filter(subset=='test')



train3_Imputed <- merge(x = train3_Imputed, y = train_labels0, by = "id", all = TRUE)
```

```
#target should be of type factor
train3_Imputed%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                   status_group ~ .,
                   data      = mydata  ,
                   num.trees = 500,
                   mtry      = 3,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
```

```
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Lumping+Frequency Encoding",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train3_Imputed) ,sep = "")
  )
```




```
    theme_classic()
```

```
# df3_Imputed%<>% select(-c(season,lga_3,num_private,year_recorded))
#
varImp %<>%
  arrange(-Importance)

varImp_30_names<-
  varImp %>%
  arrange(-Importance)%>%head(40)%>%select(vars)


top30vars<- c(varImp_30_names$vars)


top30vars<-c(top30vars,"subset")
```

```
df3_Imputed_reduced <- subset(df3_Imputed, select = c(top30vars) )

names(df3_Imputed_reduced )
```

```
##  [1] "quantity"                  "distance"
##  [3] "longitude"                 "haversinedist_Victorialake"
##  [5] "haversinedist_Ocean"       "latitude"
##  [7] "haversinedist_Dodoma"      "waterpoint_type"
##  [9] "gps_height"                "operation_years"
## [11] "construction_year"         "extraction_type_class"
## [13] "wpt_name"                  "id"
## [15] "subvillage"                "antiquity_days"
## [17] "day_of_year"               "ward_freq"
## [19] "amount_tsh"                "extraction_type"
## [21] "funder_redux500"           "ward"
## [23] "population"                "payment_type"
## [25] "installer_redux500"        "lga_freq"
## [27] "lga"                       "source"
## [29] "water_quality"             "district_code"
## [31] "source_type"               "management"
## [33] "region"                    "region_code"
## [35] "scheme_management"         "month_recorded"
## [37] "basin"                     "management_group"
## [39] "source_class"              "permit"
## [41] "subset"
```

#amount_tsh

This variable has a lot of zero's which seem to be *NA*s.

I'm gonna convert it to a frequency variable and it will be the last step in the feature engineering process.

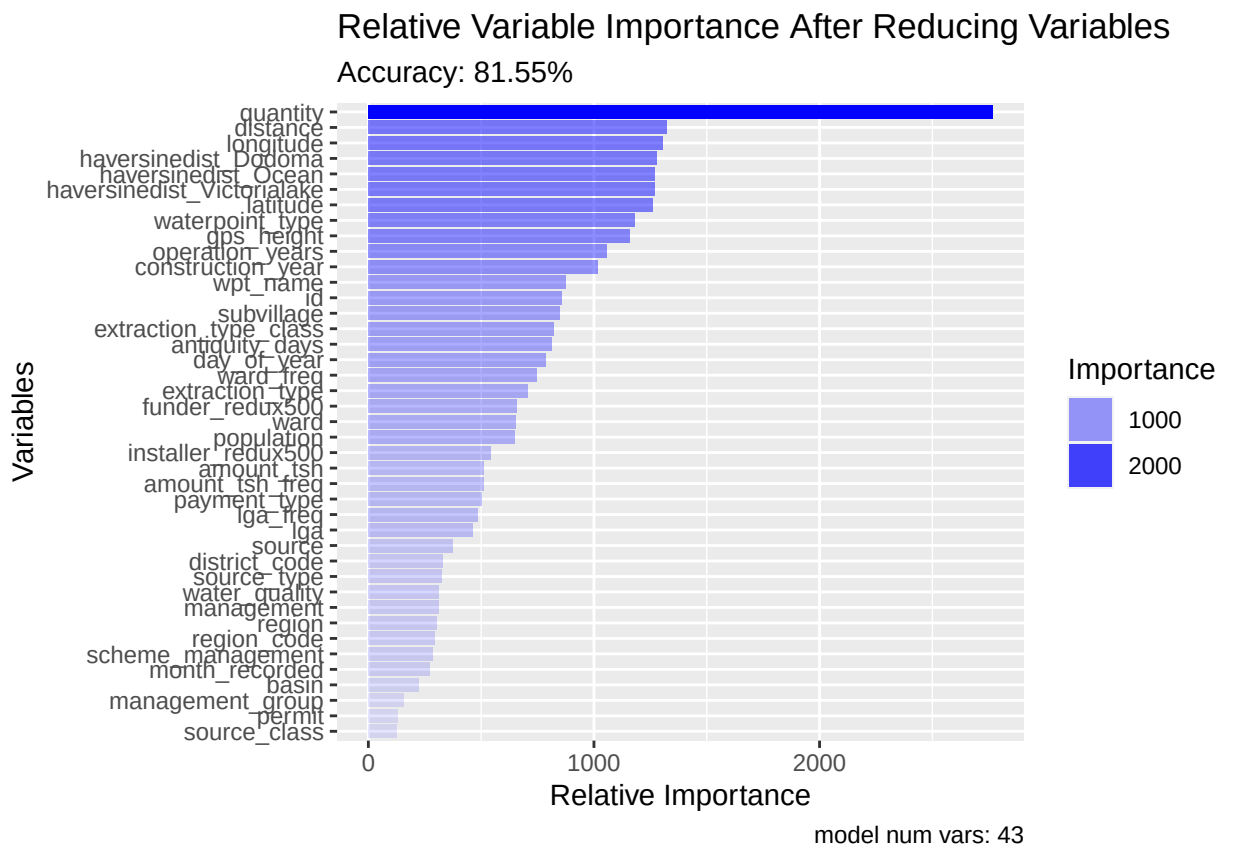
```
numtamount_tsh_dc <- df3_Imputed_reduced %>%
   group_by(amount_tsh) %>%
   dplyr::summarise( amount_tsh_freq = n() )

df3_Imputed_reduced %<>% left_join( numtamount_tsh_dc )
```

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed_reduced<- df3_Imputed_reduced%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed_reduced<- df3_Imputed_reduced%>% filter(subset=='test')



train3_Imputed_reduced <- merge(x = train3_Imputed_reduced, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed_reduced%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed_reduced%>%select(-subset)
```

```
# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                   status_group ~ .,
                   data      = mydata  ,
                   num.trees = 500,
                   mtry      = 3,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Reducing Variables",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train3_Imputed_reduced) ,sep = "")
```

```
  )
```




```
    theme_classic()
```

Compared to the full dataset the local score is almost the same.

#— Confusion Matrix. #– Predicted Values vs. Actual values

```
mymodel$confusion.matrix
```

Count-wise,the model tends to miss-classify the *functional* category more than the other two categories. Although when looked as a percentage the model shows the highest percent of miss-classification in the *functional needs repair* class where for every 7 correct predictions it almost 35 wrong predictions are done, so that's way worse than tossing a coin (50% chance). At a first sight, It could be said that the current model lacks sensitivity and that's probably the reason it predicts so bad the category with the least frequency.

#—– Submission #– Prediction

```
pred_val <- predict( mymodel , data = test3_Imputed_reduced)$predictions
head(pred_val)
```

#– Prepare submission

```
sub_df <- data.frame(
  id = df_test0$id,
  status_group = pred_val
)
```

#– Save submission

```
fwrite(sub_df, "./MySubmissions/ranger_reduced_8153.csv", nThread = 3 )
```

#Gridesearch

Let's tune the model using grid search.

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed_reduced<- df3_Imputed_reduced%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed_reduced<- df3_Imputed_reduced%>% filter(subset=='test')



train3_Imputed_reduced <- merge(x = train3_Imputed_reduced, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed_reduced%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed_reduced%>%select(-subset)




mytrees<-c(500,600,750,900,1250)
mymtry<- c(4,5,6,8)

migrid<- expand.grid(mytrees=mytrees, mymtry=mymtry)

res_val<- data.frame(trees=0,mtry=0,acc=0)
for (i in 1:nrow(migrid)){

mymodel <- ranger(
                  status_group ~ .,
                  data       = mydata  ,
                  num.trees = migrid$mytrees[i],
                  mtry       = migrid$mymtry[i],
                  importance="impurity",
                  seed=123
                )

           # Calculate error and accuracy
              error_val <- mymodel$prediction.error
              accu_val <- 1 - error_val
               #Store results
              res_temp <- data.frame(
              trees= migrid$mytrees[i],
              mtry= migrid$mymtry[i],
              acc= accu_val
              )
```

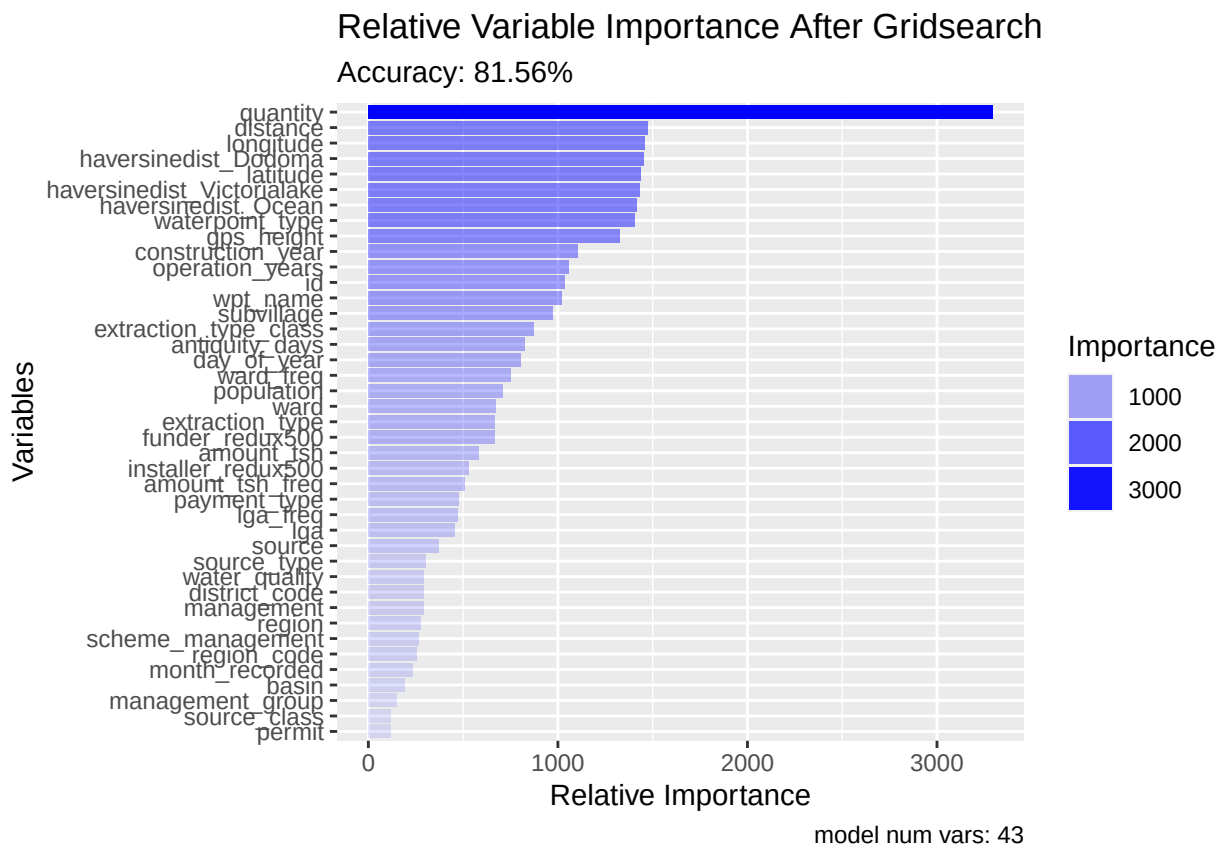
```
              res_val<- rbind(res_val,res_temp)

}
```

mytry=5 and 1250 trees showed a very minuscule improvement in the local score

```
# df3_Imputed <- missRanger(df2_Imputed, pmm.k = 3, num.trees = 100)

# #contains only the imputed features+ the new date variables
train3_Imputed_reduced<- df3_Imputed_reduced%>% filter(subset=='train')

#test dataframe with imputed values and new date variables
test3_Imputed_reduced<- df3_Imputed_reduced%>% filter(subset=='test')



train3_Imputed_reduced <- merge(x = train3_Imputed_reduced, y = train_labels0, by = "id", all = TRUE)

#target should be of type factor
train3_Imputed_reduced%<>%mutate(status_group = as.factor(status_group))

mydata<-train3_Imputed_reduced%>%select(-subset)


# If it gives you an error, make sure that your target i
#is a factor




mymodel <- ranger(
                   status_group ~ .,
                   data      = mydata  ,
                   num.trees = 1250,
                   mtry      = 5,
                   importance="impurity",
                   seed=123
                 )

error_val <- mymodel$prediction.error
error_val
accu_val <- 1 - error_val
accu_val
```

```
varImp <- mymodel$variable.importance %>%
  as.data.frame()


#Create a new column from the row names
varImp   %<>%
  mutate( vars = rownames(varImp)) %>%
  arrange(-.)

# Drop the rownames column
```

```
rownames(varImp) <- NULL

#rename the column containing the importance values
names(varImp)[1] <- "Importance"


#Order by importance in descending order
varImp %<>%
  arrange(-Importance)


#fct_reorder: orders the variable "vars" according to
#the variable "Importance"

varImp%>%
    ggplot(aes(x = fct_reorder(vars,Importance), y = Importance,alpha = Importance)) +
    geom_col(fill = "blue")+
    coord_flip()+
  labs(
    title = "Relative Variable Importance After Gridsearch",
    subtitle = paste("Accuracy: ", round(100*accu_val,2), "%", sep = ""),
    x = "Variables",
    y = "Relative Importance",
    caption = paste("model num vars: ", ncol(train3_Imputed_reduced) ,sep = "")
  )
```



```
    theme_classic()
```

#—— Submission #– Prediction

```
pred_val <- predict( mymodel , data = test3_Imputed_reduced)$predictions
head(pred_val)
```

#– Prepare submission

```
sub_df <- data.frame(
  id = df_test0$id,
  status_group = pred_val
)
```

#– Save submission

```
fwrite(sub_df, "./MySubmissions/ranger_reduced_8156.csv", nThread = 3 )
```

-local score: 0.81536 - Submission score: 0.8189

Not bad, I couldn't reach 0.82 but almost :))

Further Recommmendations:

- Create new variables based on the interaction between variables.
- Use XGBoost Algorithm