INFO-F-422 - Statistical Foundations of Machine Learning

Abhisheik Krishnagiri Tupil Ravikanth - abhisheik.krishnagiri.tupil.ravikanth@vub.ac.be - **Student ID 0575002**

Sajjad Mahmoudi - sajjad.mahmoudi@vub.be - **Student ID 0573106**

Tripat Kaur - tripat.kaur@vub.be - **Student ID 0572962**

Video presentation: https://youtu.be/Q164qI5EOVQ

Data Driven - Pump It Up

Introduction

Installing the required packages for the code execution.

In [1]:
```
install.packages('randomForest')
library(randomForest)
install.packages('tidyverse')
library(tidyverse)
install.packages("caret")
library(caret)
install.packages("class")
library(class)
install.packages("rpart.plot")
library(rpart)
library(rpart.plot)
install.packages("corrplot")
library(corrplot)
```

```
Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)

package 'randomForest' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\downloaded_packages
randomForest 4.6-14

Type rfNews() to see new features/changes/bug fixes.

Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)

package 'tidyverse' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\downloaded_packages
-- Attaching packages ------------------------------------------------------------------------------ tidyverse 1.3.1 --

v ggplot2 3.3.3     v purrr   0.3.4
v tibble  3.1.0     v dplyr   1.0.5
v tidyr   1.1.3     v stringr 1.4.0
v readr   1.4.0     v forcats 0.5.1

-- Conflicts --------------------------------------------------------------------------------- tidyverse_conflicts() --
x dplyr::combine()  masks randomForest::combine()
x dplyr::filter()   masks stats::filter()
x dplyr::lag()      masks stats::lag()
x ggplot2::margin() masks randomForest::margin()

Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)

  There is a binary version available but the source version is later:
      binary source needs_compilation
caret 6.0-86 6.0-88              TRUE

  Binaries will be installed
package 'caret' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\downloaded_packages
Loading required package: lattice


Attaching package: 'caret'


The following object is masked from 'package:purrr':

    lift


Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)

package 'class' successfully unpacked and MD5 sums checked
Warning message:
"cannot remove prior installation of package 'class'"
Warning message in file.copy(savedcopy, lib, recursive = TRUE):
"problem copying C:\Users\HP\Documents\R\win-library\4.0\00LOCK\class\libs\x64\class.dll to C:\Users\HP\Documents\R\win-library\4.0\class\libs\x64\class.dll: Permission denied"
Warning message:
"restored 'class'"
The downloaded binary packages are in
        C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\downloaded_packages
Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)

package 'rpart.plot' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\downloaded_packages
Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)

package 'corrplot' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\downloaded_packages
corrplot 0.88 loaded
```

In [2]:
```
#defining functions for caluclating precision, recall, and f1-score
calc_precision <- function(confusion_matrix){
    precision <- rep(0,length(confusion_matrix[,1]))
    for (col in seq(1,length(confusion_matrix[,1]))){
       precision[col] <-confusion_matrix[col,col]/sum(confusion_matrix[1:3,col])*100
    }
    return (precision)
}

calc_recall <- function(confusion_matrix){
    recall <- rep(0,length(confusion_matrix[,1]))
    for (row in seq(1,length(confusion_matrix[,1]))){
       recall[row] <- confusion_matrix[row,row]/sum(confusion_matrix[row,1:3])*100
    }
```

```
    return (recall)
}

calc_f1 <- function(precision,recall){
    f1 <- rep(0,length(precision))
    for (i in seq(1, length(precision))){
        f1[i] <- (2*((precision[i]*recall[i])/(precision[i]+recall[i])))
    }
    return (f1)
}
```

Reading the training input and output data and combining them into one drataframe and getting summary

In [3]:
```
# replacing all blanks values with NA
training_data_x <- read.csv('Training_Data.csv',na.strings="")
training_data_y <- read.csv('Training_Set_Labels.csv',na.strings="")
testing_data_x <- read.csv('Test_Data.csv',na.strings="")

training_data_set <- merge(x=training_data_x, y=training_data_y,by="id",all=TRUE)
training_data_set_catboost <- data.frame(training_data_set)
lapply(training_data_set,function(x) cbind(summary(x)))
```

$id

A matrix: 6 × 1 of type dbl

Min.	0.00
1st Qu.	18519.75
Median	37061.50
Mean	37115.13
3rd Qu.	55656.50
Max.	74247.00

$amount_tsh

A matrix: 6 × 1 of type dbl

Min.	0.0000
1st Qu.	0.0000
Median	0.0000
Mean	317.6504
3rd Qu.	20.0000
Max.	350000.0000

$date_recorded

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$funder

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$gps_height

A matrix: 6 × 1 of type dbl

Min.	-90.0000
1st Qu.	0.0000
Median	369.0000
Mean	668.2972
3rd Qu.	1319.2500
Max.	2770.0000

$installer

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$longitude

A matrix: 6 × 1 of type dbl

Min.	0.00000
1st Qu.	33.09035
Median	34.90874
Mean	34.07743
3rd Qu.	37.17839
Max.	40.34519

$latitude

A matrix: 6 × 1 of type dbl

Min.	-11.64944018
1st Qu.	-8.54062131
Median	-5.02159665
Mean	-5.70603266
3rd Qu.	-3.32615564
Max.	-0.00000002

$wpt_name

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$num_private

A matrix: 6 × 1 of type dbl

Min.	0.0000000
1st Qu.	0.0000000
Median	0.0000000
Mean	0.4741414
3rd Qu.	0.0000000
Max.	1776.0000000

$basin

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$subvillage

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$region

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$region_code

A matrix: 6 × 1 of type dbl

Min.	1.000
1st Qu.	5.000
Median	12.000
Mean	15.297
3rd Qu.	17.000
Max.	99.000

$district_code

A matrix: 6 × 1 of type dbl

Min.	0.000000
1st Qu.	2.000000
Median	3.000000
Mean	5.629747
3rd Qu.	5.000000
Max.	80.000000

$lga

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$ward

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$population

A matrix: 6 × 1 of type dbl

Min.	0.00
1st Qu.	0.00
Median	25.00
Mean	179.91
3rd Qu.	215.00
Max.	30500.00

$public_meeting

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$recorded_by

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$scheme_management

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$scheme_name

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$permit

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

$construction_year

A matrix: 6 × 1 of type dbl

Min.	0.000
1st Qu.	0.000
Median	1986.000
Mean	1300.652
3rd Qu.	2004.000

	Max.	2013.000
$extraction_type	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$extraction_type_group	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$extraction_type_class	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$management	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$management_group	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$payment	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$payment_type	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$water_quality	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$quality_group	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$quantity	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$quantity_group	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$source	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$source_type	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$source_class	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$waterpoint_type	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character
$waterpoint_type_group	A matrix: 3 × 1 of type chr	
	Length	59400
	Class	character
	Mode	character

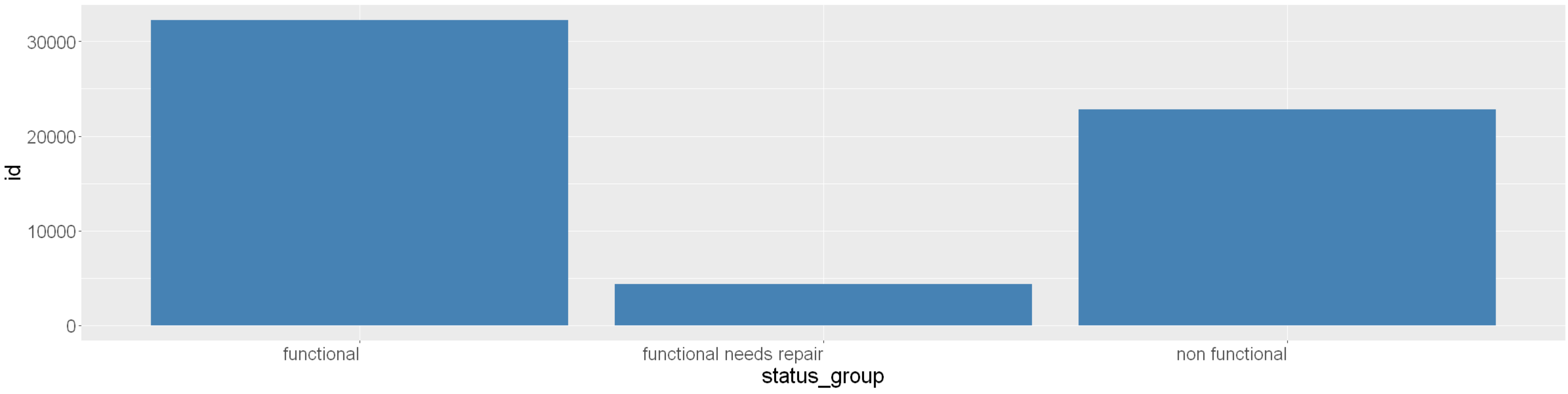
$status_group

A matrix: 3 × 1 of type chr

Length	59400
Class	character
Mode	character

Visualizing how the data is distributed over the target variable

In [4]:
```
aggregat_count_status_group <- aggregate(id~status_group,training_data_set,function(x){length(unique(x))})
options(repr.plot.width=20, repr.plot.height=5)
ggplot(aggregat_count_status_group, aes(x=status_group, y=id)) +
geom_bar(stat="identity",fill="steelblue")+theme(text = element_text(size=20),
        axis.text.x = element_text( hjust=1))
```



As we see there is hight amount of imbalance in the data. Accuracy thus might not be a good measure of how well a model performs.

Task 1 - Data Preprocessing and Feature Selection

Starting the process of feature selecting by doing analysis over the various columns

1. We have taken the summary of all columns. Looking at the mean and median of num_private it is clear that it contains mostly 0 values. So we can drop that column.

In [5]:
```
training_data_set <- subset(training_data_set,select=-num_private)
```

2. Next we look at those columns which have only one constant value. We can remove those columns since they will not provide any information to the model

In [6]:
```
unique_value_list <- apply(training_data_set, 2, function(x) length(unique(x)))
print(sort(unique_value_list))
```

```
          recorded_by        public_meeting                permit
                    1                     3                     3
         source_class          status_group      management_group
                    3                     3                     5
             quantity        quantity_group         quality_group
                    5                     5                     6
waterpoint_type_group extraction_type_class               payment
                    6                     7                     7
         payment_type           source_type       waterpoint_type
                    7                     7                     7
        water_quality                 basin                source
                    8                     9                    10
           management     scheme_management extraction_type_group
                   12                    13                    13
      extraction_type         district_code                region
                   18                    20                    21
          region_code     construction_year            amount_tsh
                   27                    55                    98
                  lga         date_recorded            population
                  125                   356                  1049
               funder                  ward             installer
                 1898                  2092                  2146
           gps_height           scheme_name           subvillage
                 2428                  2697                 19288
             wpt_name             longitude              latitude
                37400                 55366                 57517
                   id
                59400
```

Conclusion from 2

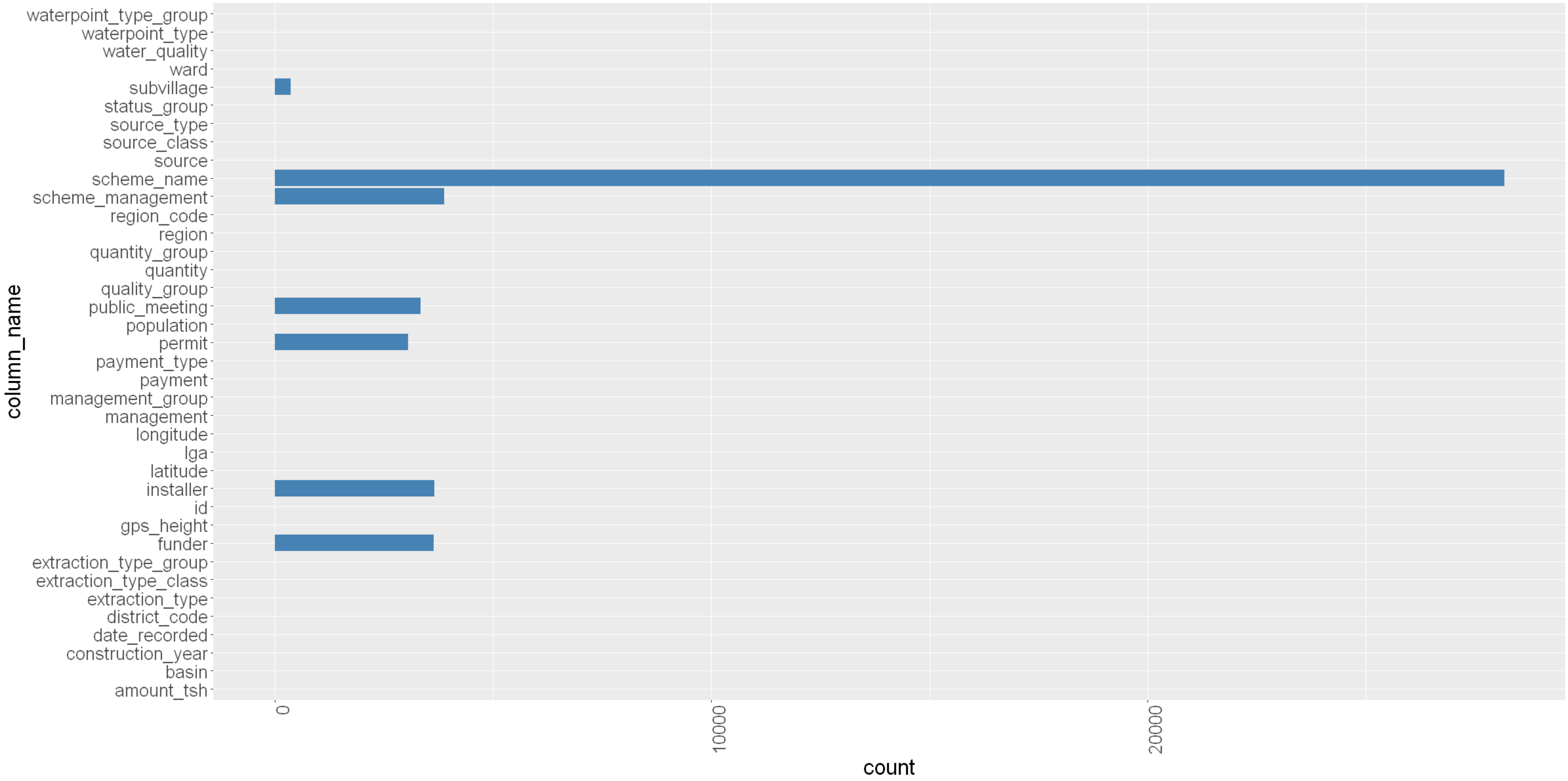
a) recorded_by can be removed as it is constant throughout the dataset.\ b) wpt_name is the name of the waterpump. Since it has 37,400 unique values it can be implied that it does not really affect the functionality of a water pump. So this feature can be removed.

In [7]:
```
training_data_set <- subset(training_data_set, select = -c(wpt_name,recorded_by))
```

In [8]:
```
# getting the number of distinct values per feature to identify which featurs can be removed because they have only one unqieu value throughout
col_name <- colnames(training_data_set)
training_data_set <- training_data_set[sapply(training_data_set,function(x)length(unique(x))>1)]
```

3. Next we take a look at the columns with missing values. We have already replaced the blank spaces in the dataset with NA while reading the csv file.

In [9]:
```
count_missing_values <- data.frame("column_name"=c(colnames(training_data_set)),"count"=apply(training_data_set,2,function(x){sum(is.na(x))}))
# print(count_missing_values[1,2])
options(repr.plot.width=20, repr.plot.height=10)
ggplot(count_missing_values, aes(x=column_name, y=count)) +
geom_bar(stat="identity",fill="steelblue")+coord_flip()+theme(text = element_text(size=20),
        axis.text.x = element_text(angle=90, hjust=1))
```



Conclusion from 3

As we can see from the above chart scheme_name is the column with the maximum missing values. So we can drop that column. Also it will not have much effect because we get the same information from the scheme_management column. For all columns that contain "Other" as a value the missing values can be replaced by the same.

In [10]:
```
training_data_set <- subset(training_data_set,select=-scheme_name)
training_data_set$scheme_management[training_data_set$scheme_management==NA]<-'Other'
training_data_set$funder[training_data_set$funder==NA]<-'Others'
```

4. Next we can look at some categorical variables and use the chi square test to identify the relation between them

In [11]:
```
training_data_set[1:5,]
```

A data.frame: 5 × 37

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	basin	subvillage	...	water_quality	quality_group	quantity	quantity_group	source	source_type	source_class	waterpoint_type
	<int>	<dbl>	<chr>	<chr>	<int>	<chr>	<dbl>	<dbl>	<chr>	<chr>	...	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>
1	0	0	2012-11-13	Tasaf	0	TASAF	33.12583	-5.118154	Lake Tanganyika	Majengo	...	milky	milky	enough	enough	shallow well	shallow well	groundwater	hand pump
2	1	0	2011-03-05	Shipo	1978	SHIPO	34.77072	-9.395642	Rufiji	Magoda C	...	soft	good	enough	enough	shallow well	shallow well	groundwater	hand pump
3	2	0	2011-03-27	Lvia	0	LVIA	36.11506	-6.279268	Wami / Ruvu	Songambele	...	soft	good	insufficient	insufficient	machine dbh	borehole	groundwater	communal standpipe multiple
4	3	10	2013-06-03	Germany Republi	1639	CES	37.14743	-3.187555	Pangani	Urereni	...	soft	good	enough	enough	spring	spring	groundwater	communal standpipe
5	4	0	2011-03-22	Cmsr	0	CMSR	36.16489	-6.099289	Wami / Ruvu	Maata A	...	soft	good	dry	dry	shallow well	shallow well	groundwater	hand pump

---

4.a) We look at the water quality and quality group

In [12]:
```
tbl <- table(training_data_set$water_quality,training_data_set$quality_group)
tbl
chisq.test(tbl)
```

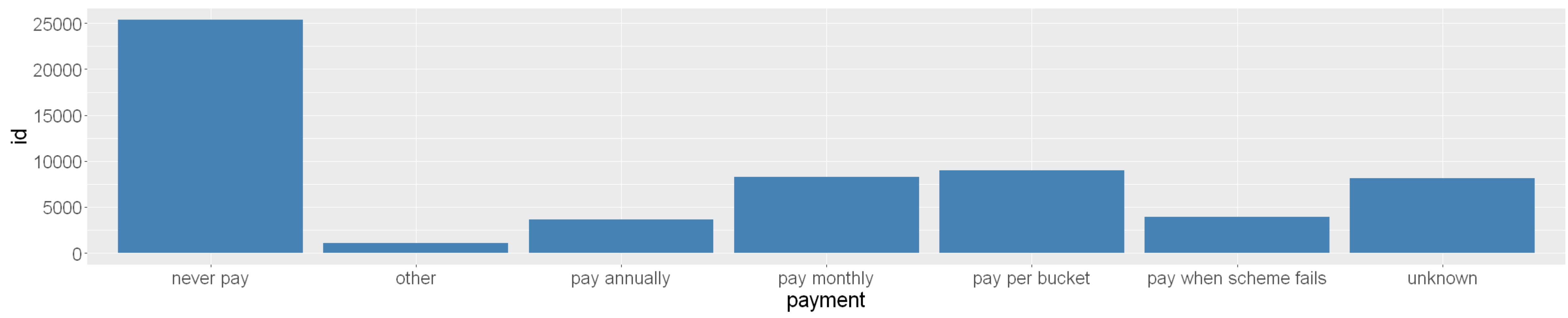
```
                    colored fluoride  good milky salty unknown
  coloured              490        0     0     0     0       0
  fluoride                0      200     0     0     0       0
  fluoride abandoned      0       17     0     0     0       0
  milky                   0        0     0   804     0       0
  salty                   0        0     0     0  4856       0
  salty abandoned         0        0     0     0   339       0
  soft                    0        0 50818     0     0       0
  unknown                 0        0     0     0     0    1876
Warning message in chisq.test(tbl):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  tbl
X-squared = 297000, df = 35, p-value < 2.2e-16
```

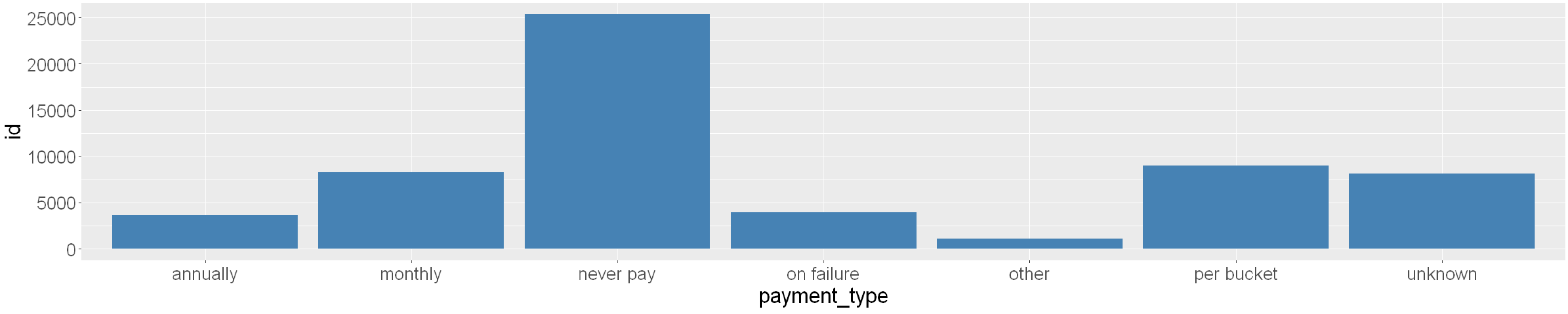
Conclusion from 4.a

As we see in the table above water_quality and quality_group are higly related to one another. Also because of such a low p-values we can imply that water_quality is a sub category of quality_group. So we can again eliminate one of the two.

4.b) We can do a similar test on payment and payment_type. This time we are visualising the data in both columns to arrive to a conclusion.

In [13]:
```
# training_data_set["count"] <- 1
options(repr.plot.width=20, repr.plot.height=4)
group_by_payment_count <- aggregate(id~payment,training_data_set, function(x) length(unique(x)))
group_by_payment_type_count <- aggregate(id~payment_type,training_data_set, function(x) length(unique(x)))
ggplot(group_by_payment_count,aes(x=payment,y=id))+geom_bar(stat="identity", fill="steelblue")+theme(text = element_text(size=20))
ggplot(group_by_payment_type_count,aes(x=payment_type,y=id))+geom_bar(stat="identity", fill="steelblue")+theme(text = element_text(size=20))
```





Conclusion from 4.b

From the above charts we can see that payment and payment type provide the same information. So we can eliminate one column from our training dataset

---

4.c) Next we look at management and management_group

In [14]:
```
tbl_manage <- table(training_data_set$management,training_data_set$management_group)
print(chisq.test(tbl_manage))
```

```
Warning message in chisq.test(tbl_manage):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  tbl_manage
X-squared = 237600, df = 44, p-value < 2.2e-16
```

4.d) We look at quantity and quantity group

In [15]:
```
tbl_quantity <- table(training_data_set$quantity,training_data_set$quantity_group)
print(tbl_quantity)
print(chisq.test(tbl_quantity))
```

```
                dry enough insufficient seasonal unknown
  dry          6246      0            0        0       0
  enough          0  33186            0        0       0
  insufficient    0      0        15129        0       0
  seasonal        0      0            0     4050       0
  unknown         0      0            0        0     789

        Pearson's Chi-squared test

data:  tbl_quantity
X-squared = 237600, df = 16, p-value < 2.2e-16
```

4.e) We look at waterpoint type and waterpoint type group

In [16]:
```
tbl_waterpoint <- table(training_data_set$waterpoint_type,training_data_set$waterpoint_type_group)
print(tbl_waterpoint)
print(chisq.test(tbl_waterpoint))
```

```
                             cattle trough communal standpipe   dam hand pump
  cattle trough                        116                  0     0         0
  communal standpipe                     0              28522     0         0
  communal standpipe multiple            0               6103     0         0
  dam                                    0                  0     7         0
  hand pump                              0                  0     0     17488
  improved spring                        0                  0     0         0
  other                                  0                  0     0         0

                             improved spring other
  cattle trough                            0     0
  communal standpipe                       0     0
  communal standpipe multiple              0     0
  dam                                      0     0
  hand pump                                0     0
  improved spring                        784     0
  other                                    0  6380
Warning message in chisq.test(tbl_waterpoint):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  tbl_waterpoint
X-squared = 297000, df = 30, p-value < 2.2e-16
```

4.f) Source, Source class and source type

In [17]:
```
print("Chi-square test for source_type and source_class")
tbl_source_type_class <- table(training_data_set$source_type,training_data_set$source_class)
# print(tbl_waterpoint)
print(chisq.test(tbl_source_type_class))

print("Chi-square test for source and source_class")
tbl_source <- table(training_data_set$source,training_data_set$source_class)
print(chisq.test(tbl_source))

print("Chi-square test for source_type and source")
table_source_type <- table(training_data_set$source,training_data_set$source_type)
print(chisq.test(table_source_type))
```

```
[1] "Chi-square test for source_type and source_class"
Warning message in chisq.test(tbl_source_type_class):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  tbl_source_type_class
X-squared = 118800, df = 12, p-value < 2.2e-16

[1] "Chi-square test for source and source_class"
Warning message in chisq.test(tbl_source):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  tbl_source
X-squared = 118800, df = 18, p-value < 2.2e-16

[1] "Chi-square test for source_type and source"
Warning message in chisq.test(table_source_type):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  table_source_type
X-squared = 356400, df = 54, p-value < 2.2e-16
```

Conclusion

(source_type and source_class) and (source and source_class) have a lower values of X^2 compared to (source and source_type). So we will choose either one or two.

4.g) extraction, extraction_type_group and extraction_type_class

In [18]:
```
print("Chi-square test for extraction and extraction_type_group")
tbl_extraction_type_group <- table(training_data_set$extraction_type,training_data_set$extraction_type_group)
# print(tbl_waterpoint)
print(chisq.test(tbl_extraction_type_group))

print("Chi-square test for extraction and extraction_type_class")
tbl_extraction  <- table(training_data_set$extraction_type,training_data_set$extraction_type_class)
print(chisq.test(tbl_extraction))

print("Chi-square test for extraction_type_group and extraction_type_class")
table_extraction_type <- table(training_data_set$extraction_type_group,training_data_set$extraction_type_class)
print(chisq.test(table_extraction_type))
```

```
[1] "Chi-square test for extraction and extraction_type_group"
Warning message in chisq.test(tbl_extraction_type_group):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  tbl_extraction_type_group
X-squared = 712800, df = 204, p-value < 2.2e-16

[1] "Chi-square test for extraction and extraction_type_class"
Warning message in chisq.test(tbl_extraction):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  tbl_extraction
X-squared = 356400, df = 102, p-value < 2.2e-16

[1] "Chi-square test for extraction_type_group and extraction_type_class"
Warning message in chisq.test(table_extraction_type):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  table_extraction_type
X-squared = 356400, df = 72, p-value < 2.2e-16
```

4.h) Next we take a look at funder and installer. But since both of them are in different cases, first we need to convert them to the same case and then use chi-square test to identify if there is a relation between the two.

In [19]:
```
training_data_set$funder <- tolower(training_data_set$funder)
training_data_set$installer <- tolower(training_data_set$installer)

print(chisq.test(table(training_data_set$funder,training_data_set$installer)))
```

```
Warning message in chisq.test(table(training_data_set$funder, training_data_set$installer)):
"Chi-squared approximation may be incorrect"
        Pearson's Chi-squared test

data:  table(training_data_set$funder, training_data_set$installer)
X-squared = NaN, df = 3666864, p-value = NA
```

We get NaN as X-Squared because the value is too small. But we infer that funder and installer are correlated by the high df value and also by observing the dataset

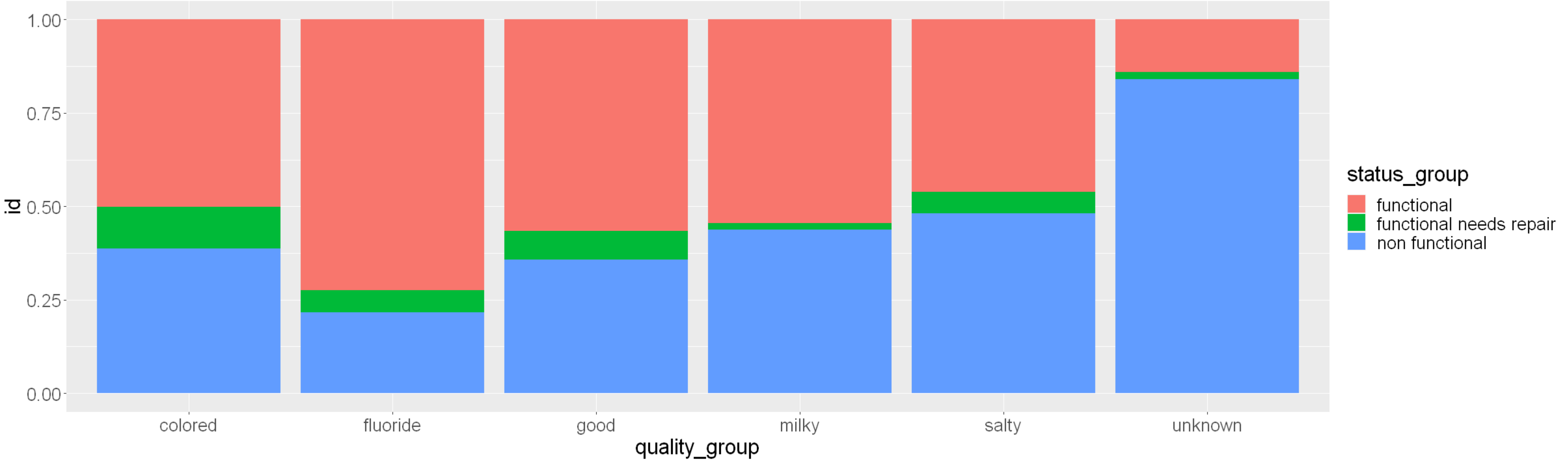
---

Removing all the columns that have been till now discussed that won't impart any extra information

In [20]:
```
training_data_set <- subset(training_data_set,select = -c(water_quality,payment_type,management_group,waterpoint_type,source_type,extraction_type,installer,quantity,source_class,extraction_type_g
```

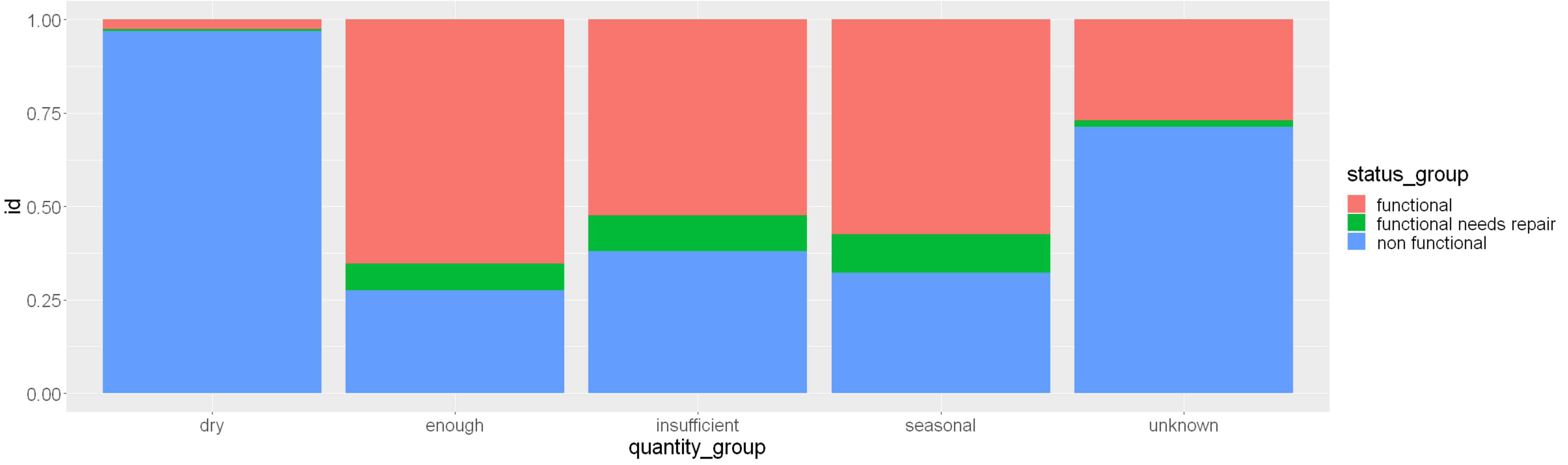
We also have a look at how the left over columns are affecting the status of a water pump.

In [21]:
```
group_by_quality_group <- aggregate(id~quality_group+status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_quality_group, aes(fill=status_group, y=id, x=quality_group)) +
    geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```

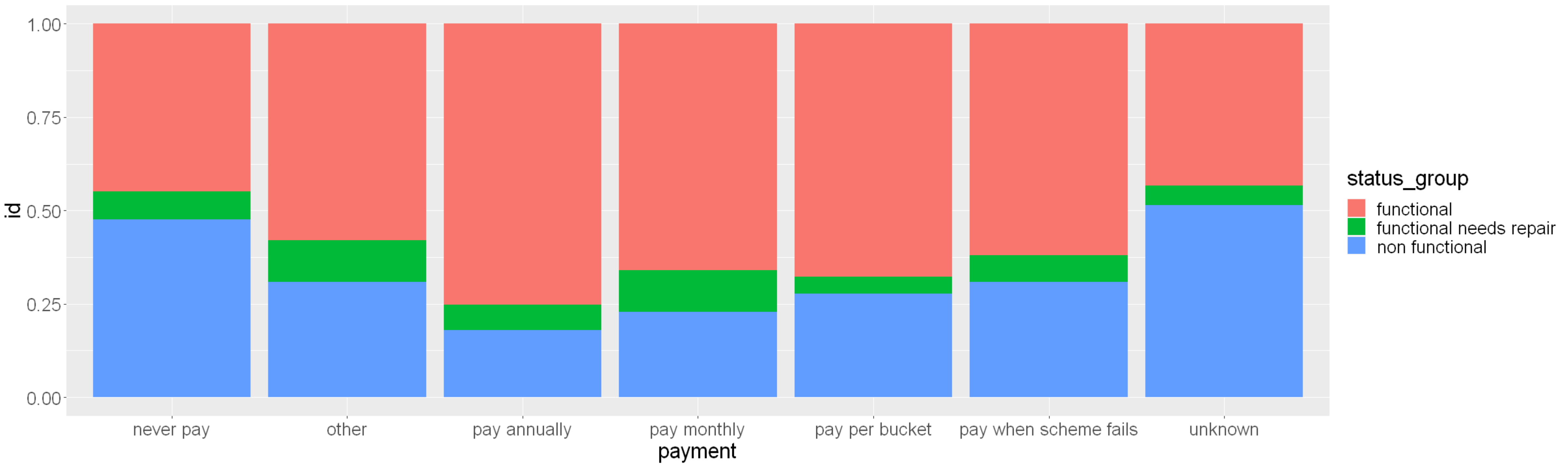


We can see that depending on the quality of water the status of the water pump changes. If the water has a lot of fluoride the pump is functional. But for salty water the pump becomes non functional.

In [22]:
```
group_by_quantity_group <- aggregate(id~quantity_group+status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_quantity_group, aes(fill=status_group, y=id, x=quantity_group)) +
    geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```

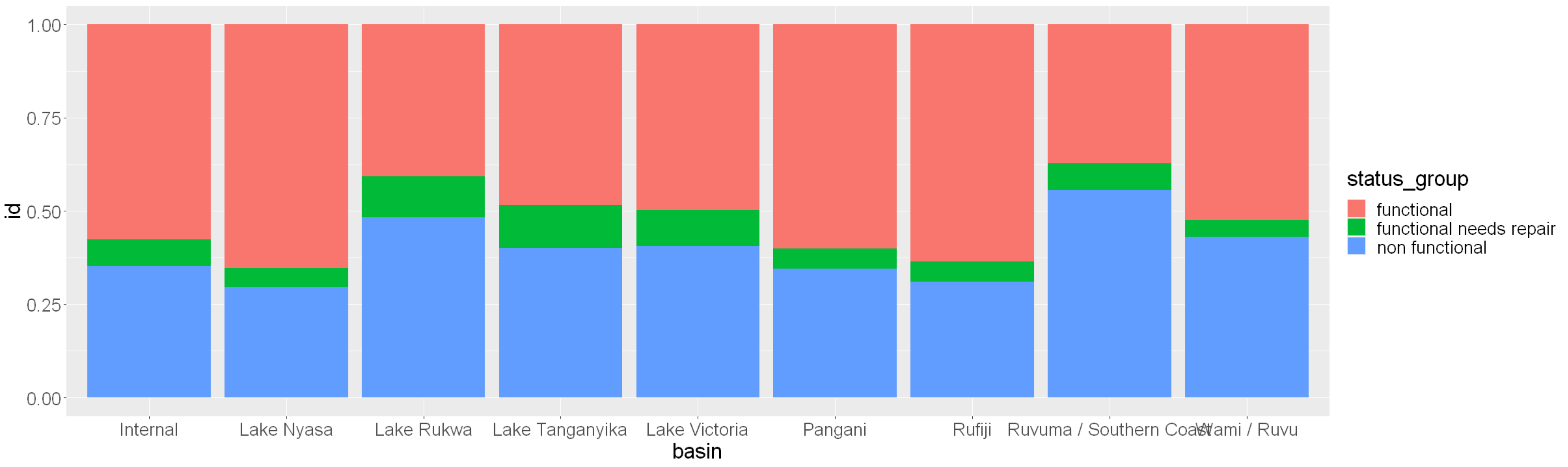


In [23]:
```
group_by_payment <- aggregate(id~payment +status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_payment , aes(fill=status_group, y=id, x=payment )) + geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```



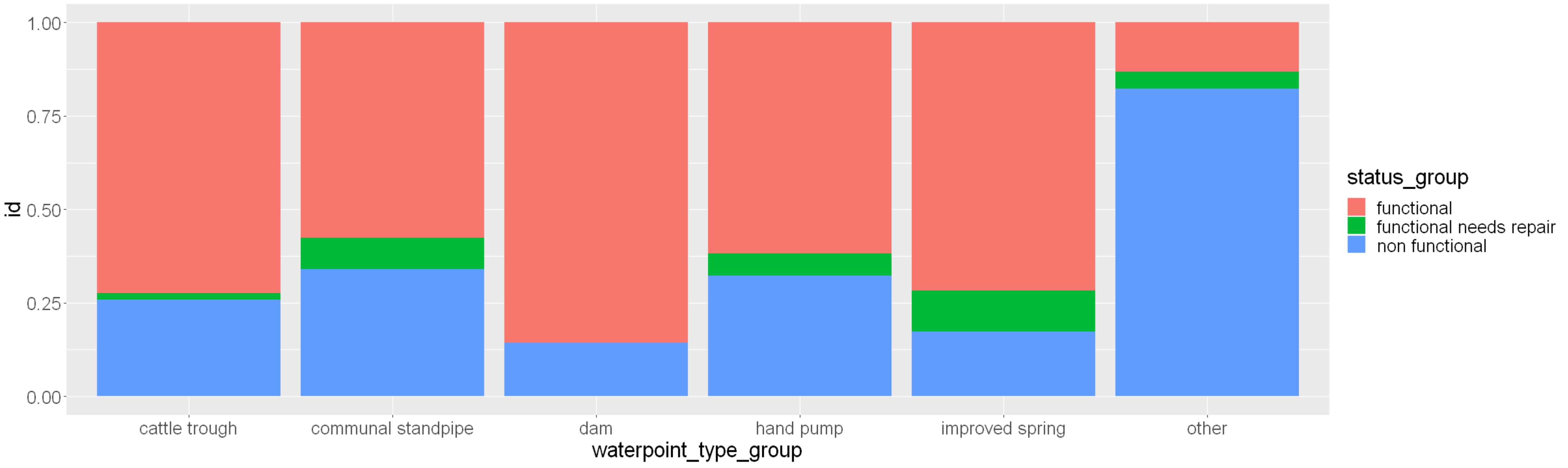
When payments are not done more pumps are non functional compared to when payments are done annually.

In [24]:
```
group_by_basin <- aggregate(id~basin +status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_basin , aes(fill=status_group, y=id, x=basin)) + geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```



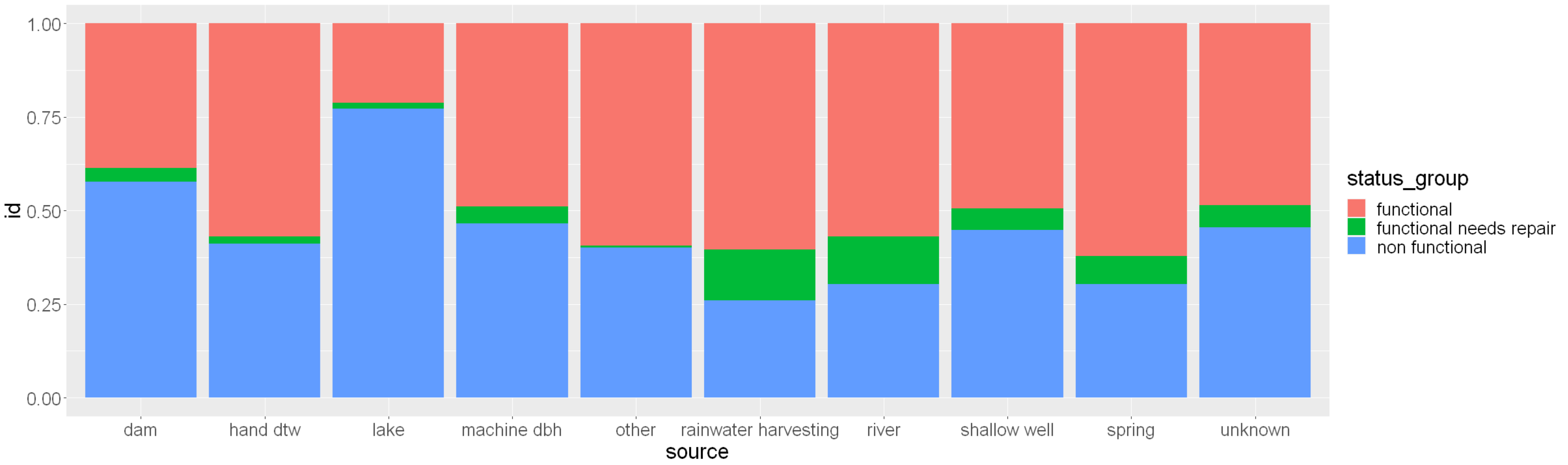
The quantity of water also clearly affects the status. Dry pumps are mostly non functional whereas pumps where there is enough water are functional.

In [25]:
```
group_by_waterpoint_group <- aggregate(id~waterpoint_type_group+status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_waterpoint_group, aes(fill=status_group, y=id, x=waterpoint_type_group)) +
    geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```



The type of water pump also clearly affects the status. If it is a dam chances are the water pump will be functional. For other categories water pumps are mostly non functional.

In [26]:
```
group_by_source <- aggregate(id~source+status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_source, aes(fill=status_group, y=id, x=source)) +
    geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```



The source of water pump also affects the status. If the source is lake and dam most of the pumps are non functional. For the rest of the categories there is not a large differentiation. So we keep this feature.

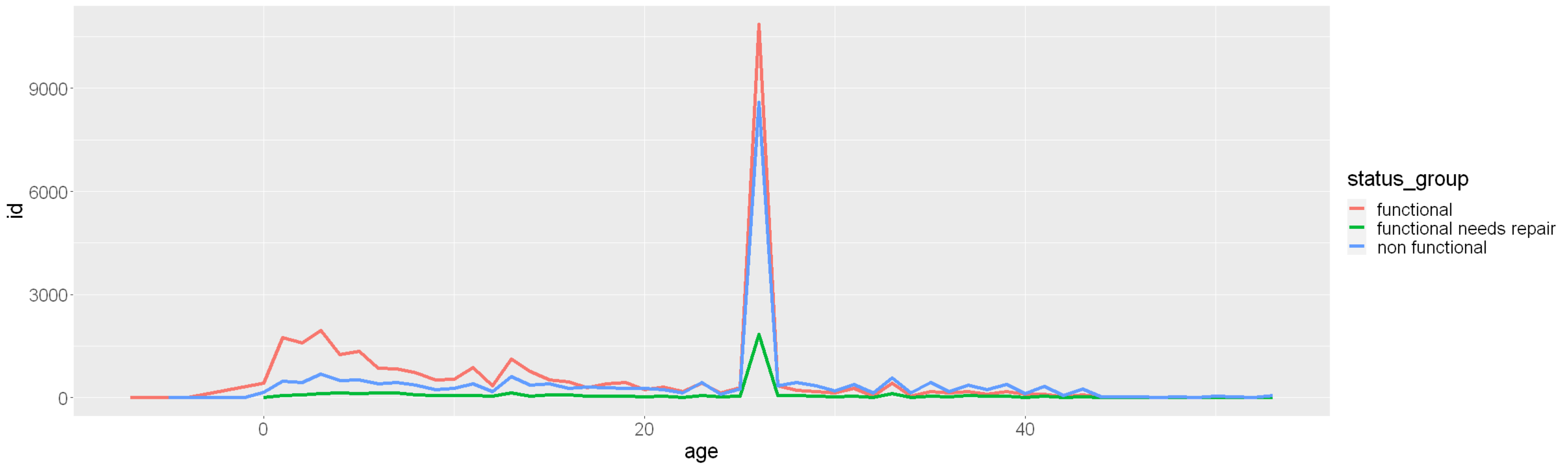
5. Since we have the constructed year and the date recorded, we can find out the age of the pump and check if it has an effect on its functionality

In [27]:
```
# relacing all 0s with median of construction year
# training_data_set$construction_year[training_data_set["construction_year"]==0] <-median(training_data_set$construction_year)
date_recorded_in_years <- as.numeric(format(as.Date(training_data_set$"date_recorded"),"%Y"))
# print(training_data_set$construction_year[training_data_set$construction_year<1986])
training_data_set["age"] <-   date_recorded_in_years - training_data_set["construction_year"]
training_data_set$age[training_data_set["age"]==as.numeric(format(as.Date(training_data_set$"date_recorded"),"%Y"))] <- median(training_data_set$age)
```

Since there is 0 in construction year we can replace that with the median to be able to calculate the age of the pump by subtracting the date_recorded and construction_year to get one feature called age.

In [28]:
```
group_by_age <- aggregate(id~age+status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_age)
```

```
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_age, aes(x = age, y = id)) +
  geom_line(aes(color = status_group),size=1.5)+theme(text = element_text(size=20))
```



Conclusion from 5

As the age increases after around 30 the number of functioning pumps decreases. Also we see some values are below 0 indicating that at certain places the age is negative. The constructed year is after the recorded date. So we can filter out those records and continue to keep age as a variable and remove date_recorded and construction_year

In [29]:
```
training_data_set <- training_data_set[!(training_data_set["age"]<0),]
training_data_set <- subset(training_data_set,select=-c(date_recorded,construction_year))
```

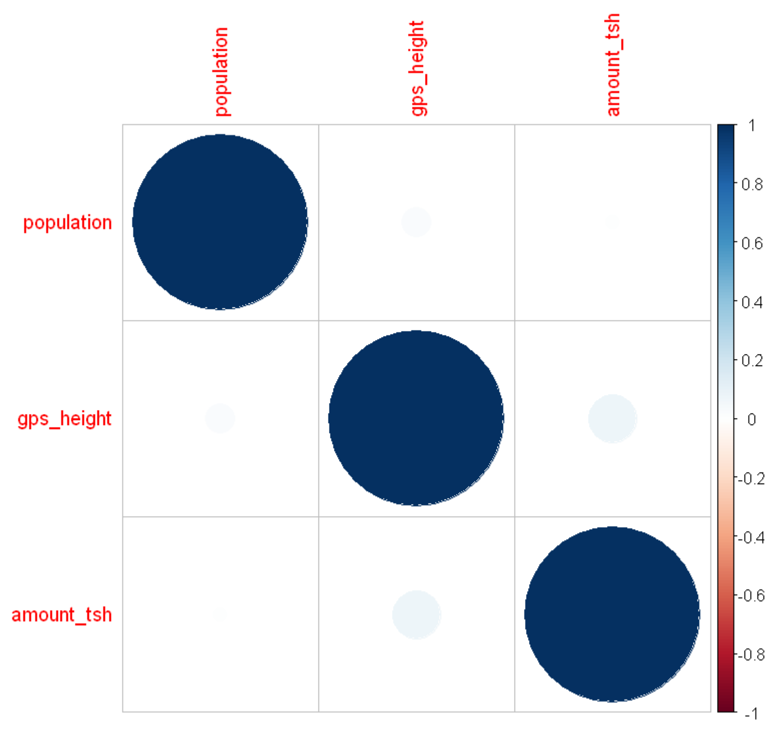
In [30]:
```
colnames(training_data_set)
```

'id' · 'amount_tsh' · 'funder' · 'gps_height' · 'longitude' · 'latitude' · 'basin' · 'subvillage' · 'region' · 'region_code' · 'district_code' · 'lga' · 'ward' · 'population' · 'public_meeting' · 'scheme_management' · 'permit' · 'extraction_type_class' · 'management' · 'payment' · 'quality_group' · 'quantity_group' · 'source' · 'waterpoint_type_group' · 'status_group' · 'age'

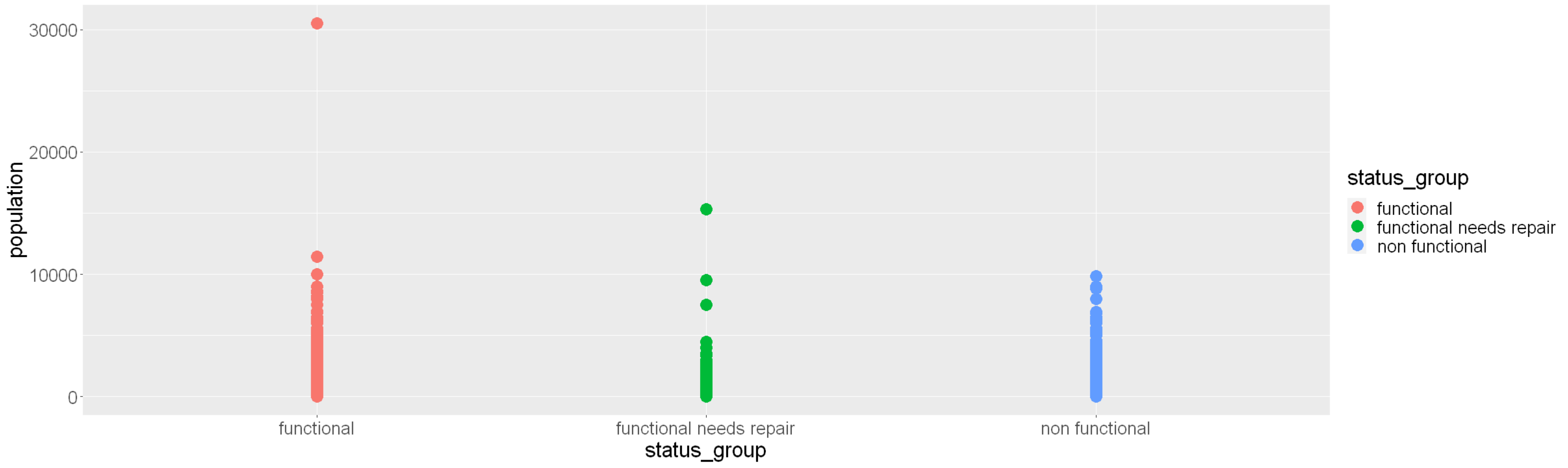
6. Next we look at the numerical features and see if there are any missing values. Also we find the correlation between the numerical values to see if any features could be eliminated.

In [31]:
```
# print(unique(training_data_set$population))
# 0 population does not make sense so we replace it with the mean of the population
training_data_set$population[training_data_set$population==0] <- mean(training_data_set$population)
# for gps_height and amount_tsh 0 values are acceptable
```

In [32]:
```
corrplot(cor(training_data_set[,c("population","gps_height","amount_tsh")], method = "pearson"), method="circle")
```

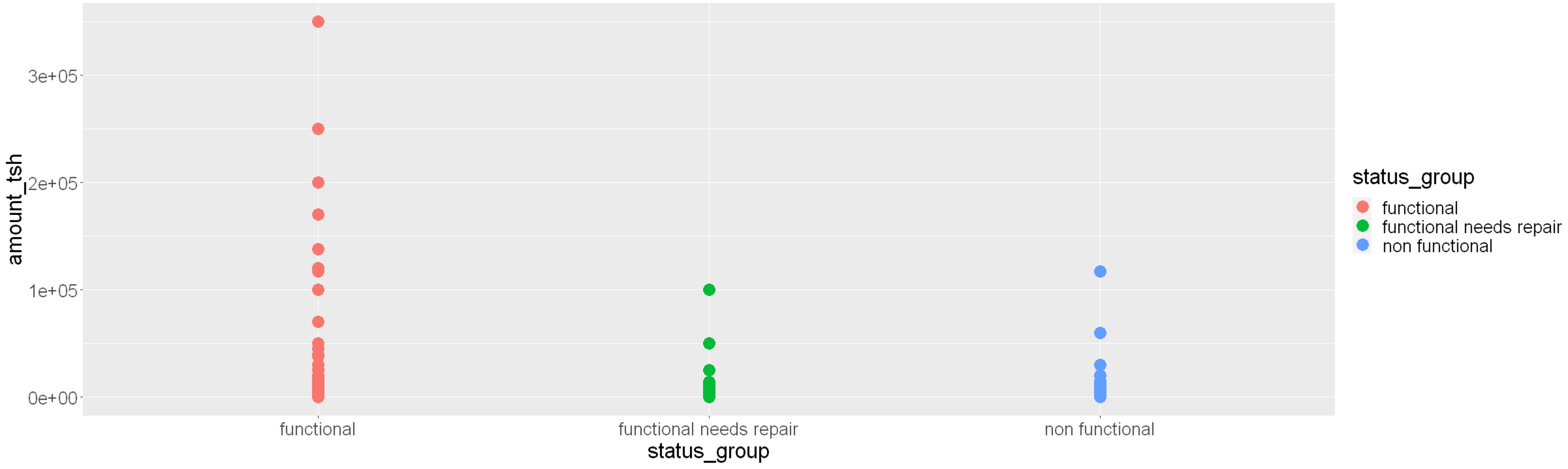


In [33]:
```
ggplot(training_data_set, aes(x=status_group, y=population,color=status_group)) + geom_point(size=5)+theme(text = element_text(size=20))
```



In [34]:
```
ggplot(training_data_set, aes(x=status_group, y=gps_height,color=status_group)) + geom_point(size=5)+theme(text = element_text(size=20))
```

In [35]:
```
ggplot(training_data_set, aes(x=status_group, y=amount_tsh,color=status_group)) + geom_point(size=5)+theme(text = element_text(size=20))
```



As the amount of water increases in the pump the pump seems to be more functional.

Conclusion from 6

As we can see from the above correlation matrix none of the three features are highly related. So we keep all of them.

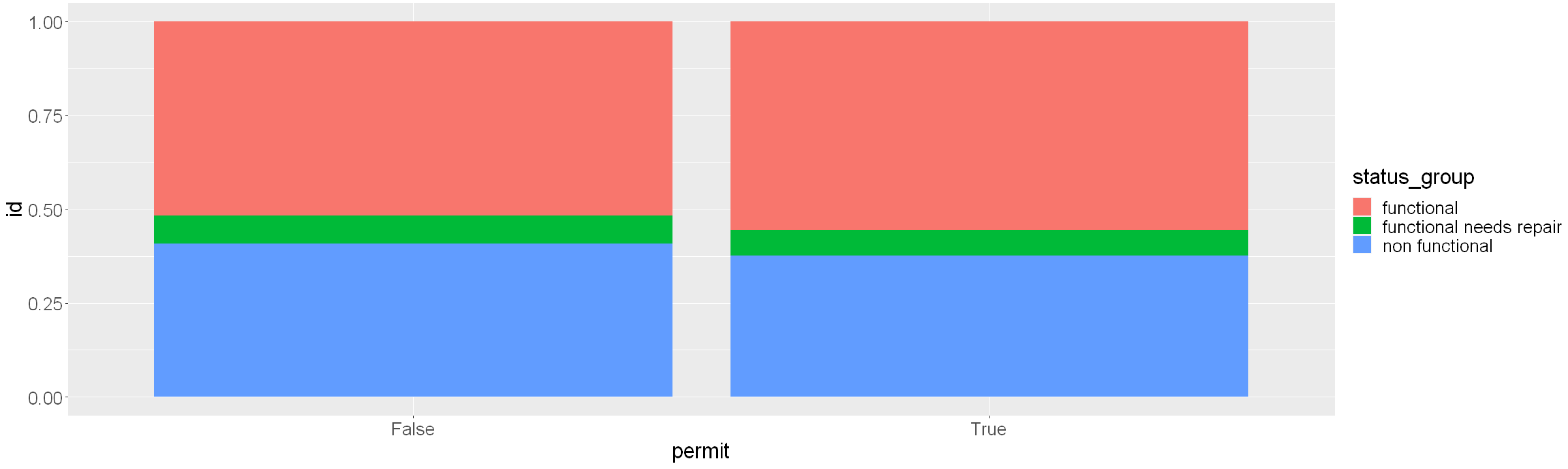
7. We now replace all the missing values from the features.

In [36]:
```
names(which(colSums(is.na(training_data_set)) > 0))
```

'funder' · 'subvillage' · 'public_meeting' · 'scheme_management' · 'permit'

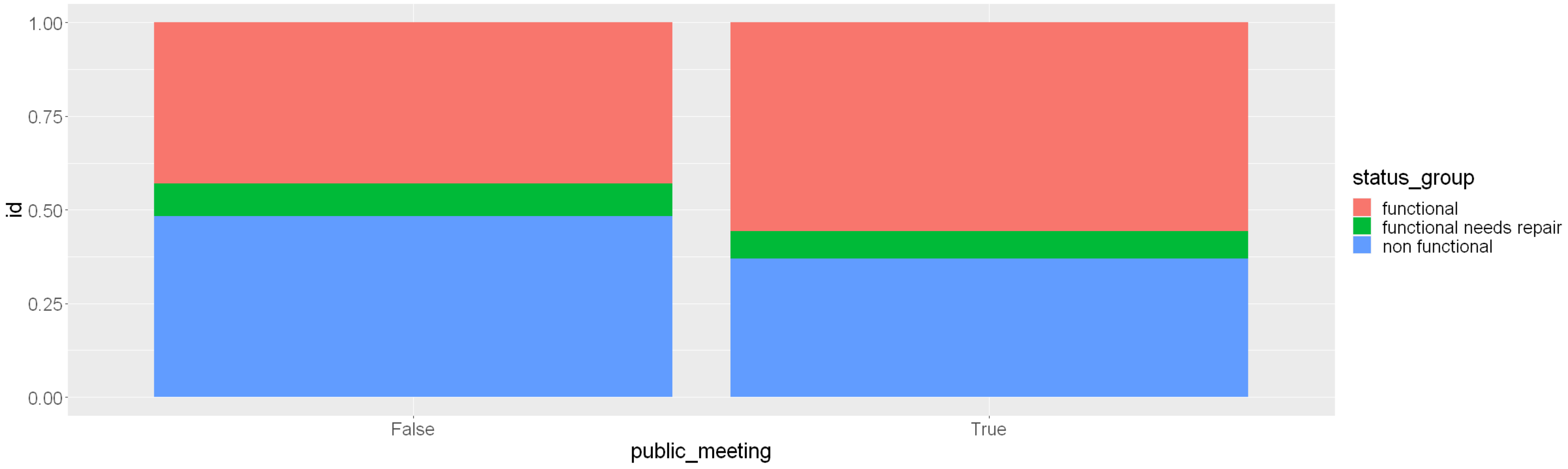
We do it only for scheme_management because we will be eliminating public_meeting and permit as per below charts.

In [37]:
```
group_by_permit <- aggregate(id~permit+status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_permit, aes(fill=status_group, y=id, x=permit)) +
    geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```



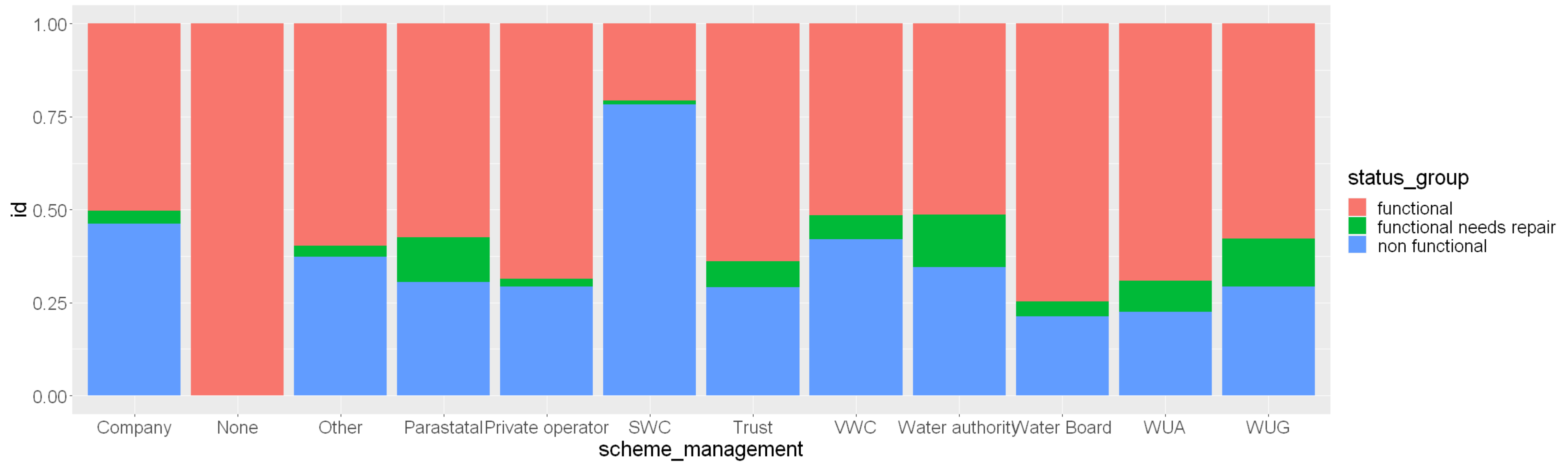
From the above graph we see that permit does not really affect the status of a pump. For both True and False the distribution of pumps based on their functional status continues to be the same.

In [38]:
```
group_by_public_meeting <- aggregate(id~public_meeting+status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_public_meeting, aes(fill=status_group, y=id, x=public_meeting)) +
    geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```



Same as for permit, we observe that public_meeting might not affect the target variable so we can eliminate it

In [39]:
```
group_by_scheme_management <- aggregate(id~scheme_management+status_group,training_data_set, function(x) length(unique(x)))
# print(group_by_permit)
options(repr.plot.width=20, repr.plot.height=6)
ggplot(group_by_scheme_management, aes(fill=status_group, y=id, x=scheme_management)) +
    geom_bar(position="fill", stat="identity")+theme(text = element_text(size=20))
```



Based on the scheme the functionality varies. This is observed from the stacked percentage chart above. Also we have a "Other" category. We can map all missing values to it.

In [40]:
```
training_data_set["scheme_management"][is.na(training_data_set["scheme_management"])]<- "Other"
training_data_set <- subset(training_data_set,select=-c(permit,public_meeting))
```

8. Since now we have our list of features we try and reduce the levels in our categorical variables.

In [41]:
```
unique_value_list <- apply(training_data_set, 2, function(x) length(unique(x)))
print(sort(unique_value_list))
```

```
        status_group       quantity_group        quality_group 
                   3                    5                    6 
waterpoint_type_group extraction_type_class              payment 
                   6                    7                    7 
               basin               source    scheme_management 
                   9                   10                   12 
          management        district_code               region 
                  12                   20                   21 
         region_code                  age           amount_tsh 
                  27                   54                   98 
                 lga           population               funder 
                 125                 1048                 1897 
                ward           gps_height           subvillage 
                2092                 2428                19287 
           longitude             latitude                   id 
               55358                57508                59391 
```

We replace the missing values from funder with NA

In [42]:
```
training_data_set["funder"][is.na(training_data_set["funder"])] <- "Other"
# print((unique(training_data_set["funder"])))
```

9. Since we have latitude and longitude along with region we can remove the rest of the geographical features.

In [43]:
```
training_data_set <- subset(training_data_set, select = -c(subvillage,region_code,district_code,lga,ward,region))
print(colnames(training_data_set))
```

```
 [1] "id"                    "amount_tsh"            "funder"               
 [4] "gps_height"            "longitude"             "latitude"             
 [7] "basin"                 "population"            "scheme_management"    
[10] "extraction_type_class" "management"            "payment"              
[13] "quality_group"         "quantity_group"        "source"               
[16] "waterpoint_type_group" "status_group"          "age"                  
```

10. Next we replace all blank numerical data with their corresponding median values

In [44]:
```
training_data_set["amount_tsh"] <- lapply((training_data_set["amount_tsh"]),as.numeric)
training_data_set["amount_tsh"][is.na(training_data_set["amount_tsh"])]<- mean((training_data_set$amount_tsh))
training_data_set["gps_height"][is.na(training_data_set["gps_height"])]<- mean(as.numeric(training_data_set$gps_height))
training_data_set["latitude"][is.na(training_data_set["latitude"])]<- mean(as.numeric(training_data_set$latitude))
training_data_set["longitude"][is.na(training_data_set["longitude"])]<- mean(as.numeric(training_data_set$longitude))
# 0 longitude probably implies missing values
training_data_set$longitude[training_data_set$longitude==0]<-median(training_data_set$longitude)
# print(unique(training_data_set["public_meeting"]))
```

11. We remove id column since that is unique for every row. Also we remove the count column that we generated above

In [45]:
```
# remove_column <- "id"

training_data_set <- subset(training_data_set,select=-c(id))
print(colnames(training_data_set))
```

```
 [1] "amount_tsh"            "funder"                "gps_height"           
 [4] "longitude"             "latitude"              "basin"                
 [7] "population"            "scheme_management"     "extraction_type_class"
[10] "management"            "payment"               "quality_group"        
[13] "quantity_group"        "source"                "waterpoint_type_group"
[16] "status_group"          "age"                  
```

12. We factorize the target variable to be able to use it in our classification models next.

In [46]:
```
training_data_set$status_group<-as.factor(training_data_set$status_group)
unique(training_data_set$status_group)
```

non functional · functional · functional needs repair

▶ **Levels**:

In [47]:
```
dim(training_data_set)
```

59391 · 17

Task 2 - Model Selection and Implementation

Explanation

The first and foremost model that we decided to impelement is Random Forest. Random Forest is an ensemble technique which means that it uses bootstrapping and bagging techniques to improve its predictions. Out of the given list Random Forest is the one that can handle categorical variables directly so we could obtain a baseline accuracy from this model directly from our selected features. Also Random Forests work well with large datasets and large features.

1. Random Forest

Tuning of hyperparameters - ntree

We tried executing random forest model with different parameters to understand how well the model performs. This part of the code has been commented to reduce the execution time. After running all models we decided to choose ntree as 250 and mtry as 3. Please uncomment the code in order to execute it (Cells 37-43).

In [48]:
```
# with default parameters ntree = 500
model_1_rf = randomForest(status_group~., data = training_data_set, importance = TRUE)
```

In [49]:
```
print(model_1_rf)
```

```
Call:
 randomForest(formula = status_group ~ ., data = training_data_set,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

        OOB estimate of  error rate: 18.74%
Confusion matrix:
                        functional functional needs repair non functional
functional                   29117                     667           2471
functional needs repair       2316                    1398            603
non functional                4783                     291          17745
                        class.error
functional               0.09728724
functional needs repair  0.67616400
non functional           0.22235856
```

In [50]:
```
model_2_rf = randomForest(status_group~., data = training_data_set, importance = TRUE,ntree=100)
print(model_2_rf)
```

```
Call:
 randomForest(formula = status_group ~ ., data = training_data_set,      importance = TRUE, ntree = 100)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 4

        OOB estimate of  error rate: 19.07%
Confusion matrix:
                        functional functional needs repair non functional
functional                   29012                     709           2534
functional needs repair       2324                    1399            594
non functional                4858                     305          17656
                        class.error
functional                0.1005426
functional needs repair   0.6759324
non functional            0.2262588
```

In [51]:
```
model_3_rf = randomForest(status_group~., data = training_data_set, importance = TRUE,ntree=200)
print(model_3_rf)
```

```
Call:
 randomForest(formula = status_group ~ ., data = training_data_set,      importance = TRUE, ntree = 200)
               Type of random forest: classification
                     Number of trees: 200
No. of variables tried at each split: 4

        OOB estimate of  error rate: 18.85%
Confusion matrix:
                        functional functional needs repair non functional
functional                   29046                     680           2529
functional needs repair       2323                    1390            604
non functional                4763                     298          17758
                        class.error
functional               0.09948845
functional needs repair  0.67801714
non functional           0.22178886
```

Conclusion from hyperparameter tuning

We can conclude that for very few number of tress the error rate is high. As the number of trees are less there will be more bias in the data because all not rows will be taken into consideration while generating the samples. As the number of tress increase the error rate decreases but after a certain point it brings only a marginal improvement. So we choose an optimum point where error rate is low and so is execution time.

Tuning of hyperparameter - mtry

In [52]:
```
model_4_rf = randomForest(status_group~., data = training_data_set, importance = TRUE,ntree=250,mtry=6)
print(model_4_rf)
```

```
Call:
 randomForest(formula = status_group ~ ., data = training_data_set,      importance = TRUE, ntree = 250, mtry = 6)
               Type of random forest: classification
                     Number of trees: 250
No. of variables tried at each split: 6

        OOB estimate of  error rate: 18.89%
Confusion matrix:
                        functional functional needs repair non functional
functional                   28784                     799           2672
functional needs repair       2204                    1476            637
non functional                4557                     350          17912
                        class.error
functional                0.1076112
functional needs repair   0.6580959
non functional            0.2150401
```

In [53]:
```
model_5_rf = randomForest(status_group~., data = training_data_set, importance = TRUE,ntree=250,mtry=12)
print(model_5_rf)
```

```
Call:
 randomForest(formula = status_group ~ ., data = training_data_set,      importance = TRUE, ntree = 250, mtry = 12)
               Type of random forest: classification
                     Number of trees: 250
No. of variables tried at each split: 12

        OOB estimate of  error rate: 19.44%
Confusion matrix:
                        functional functional needs repair non functional
functional                   28373                     986           2896
functional needs repair       2147                    1517            653
non functional                4451                     412          17956
                        class.error
functional                0.1203534
functional needs repair   0.6485986
non functional            0.2131119
```

In [54]:
```
model_6_rf = randomForest(status_group~., data = training_data_set, importance = TRUE,ntree=250,mtry=1)
print(model_6_rf)
```

```
Call:
 randomForest(formula = status_group ~ ., data = training_data_set,      importance = TRUE, ntree = 250, mtry = 1)
               Type of random forest: classification
                     Number of trees: 250
No. of variables tried at each split: 1

        OOB estimate of  error rate: 22.94%
Confusion matrix:
                        functional functional needs repair non functional
functional                   30612                      87           1556
functional needs repair       3612                     217            488
non functional                7846                      35          14938
                        class.error
functional               0.05093784
functional needs repair  0.94973361
non functional           0.34537009
```

In [55]:
```
model_final_rf = randomForest(status_group~., data = training_data_set, importance = TRUE,ntree=250,mtry=6)
model_final_rf
```

```
Call:
 randomForest(formula = status_group ~ ., data = training_data_set,      importance = TRUE, ntree = 250, mtry = 6)
               Type of random forest: classification
                     Number of trees: 250
No. of variables tried at each split: 6

        OOB estimate of  error rate: 18.91%
Confusion matrix:
                        functional functional needs repair non functional
functional                   28794                     819           2642
functional needs repair       2214                    1478            625
non functional                4570                     359          17890
                        class.error
functional                0.1073012
```

```
functional needs repair   0.6576326
non functional            0.2160042
```

In [56]:
```
accuracy_rf <- sum(diag(model_final_rf$confusion))/sum(model_final_rf$confusion)*100
accuracy_rf
model_final_rf$confusion
```

81.0917555524272

A matrix: 3 × 4 of type dbl

	functional	functional needs repair	non functional	class.error
functional	28794	819	2642	0.1073012
functional needs repair	2214	1478	625	0.6576326
non functional	4570	359	17890	0.2160042

In [57]:
```
print("Recall Per Class - Random Forest")
recall_rf <- calc_recall(model_final_rf$confusion)
print(recall_rf)
```

```
[1] "Recall Per Class - Random Forest"
[1] 89.26988 34.23674 78.39958
```

In [58]:
```
print("Precision Per Class - Random Forest")
precision_rf <- calc_precision(model_final_rf$confusion)
print(precision_rf)
```

```
[1] "Precision Per Class - Random Forest"
[1] 80.93204 55.64759 84.55830
```

In [59]:
```
print("F1 - SCore - Random Forest")
f1_rf <- calc_f1(precision_rf,recall_rf)
print(f1_rf)
```

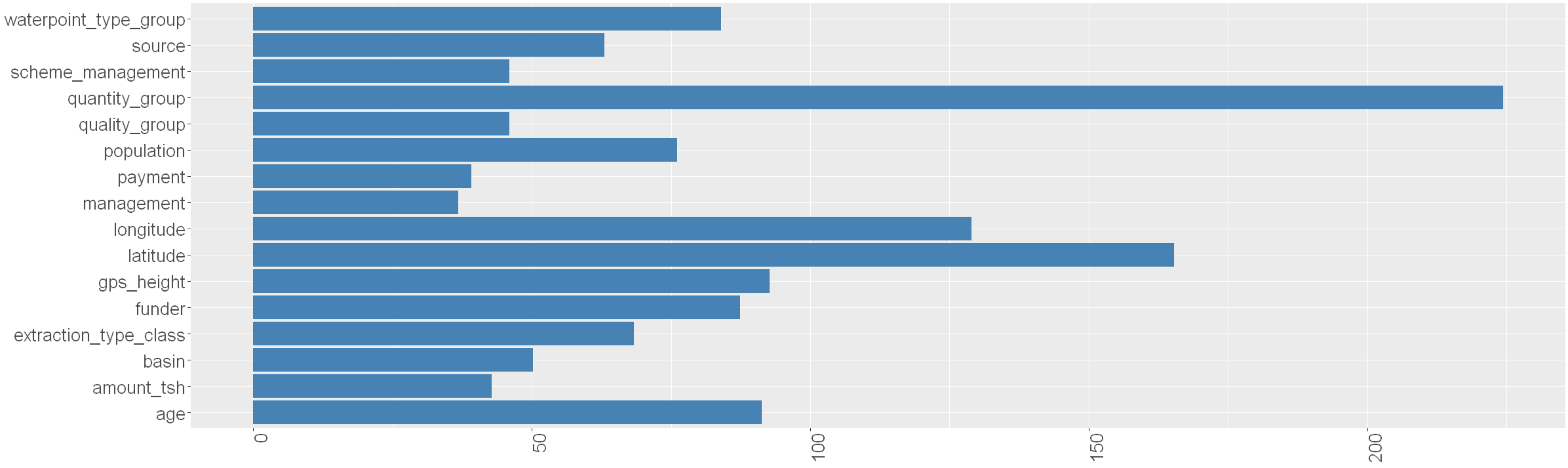
```
[1] "F1 - SCore - Random Forest"
[1] 84.89673 42.39208 81.36256
```

From the class error it is clear that the maximum error is in predicting "functional needs repair". This is because it is the class with the lowest distribution (imbalance in data). And that is also visible in the precision and recall for "functional needs repair"

Increasing the value of mtry and decreasing it cause an increase in the error rate. So we choose the value 6 for which we get the optimum rate. Using mtry=6 and ntree =250 we will generate the predictions.

Getting feature importance from random forest

In [60]:
```
feature_importance_rf <- data.frame(colName = colnames(subset(training_data_set,select=-status_group)),
                                    importance = importance(model_final_rf,1)
                                    )
options(repr.plot.width=20, repr.plot.height=6)
ggplot(feature_importance_rf[,1:2], aes(y=feature_importance_rf[,2], x=feature_importance_rf[,1])) +
   geom_bar(stat="identity",fill="steelblue")+coord_flip()+theme(text = element_text(size=20),
        axis.text.x = element_text(angle=90, hjust=1),axis.title.x = element_blank(),axis.title.y = element_blank())
```



Each feature that we have selected gives a good score in random forest.

For the next models categorical variables need to be encoded

Encoding for string columns

In [61]:
```
#creating a copy of categorical features
training_data_set_without_encoded <- data.frame(training_data_set)
```

From the above data pre-processing it is clear that the levels of "funder" variable need to be reduced. So we count the number of instances per funder and match all those occuring less than 500 times to Other

In [62]:
```
aggregate_funder_count<-data.frame(aggregate(.~funder,training_data_set, function(x)length(x) ))
aggregate_funder_count <- subset(aggregate_funder_count,select=c(1,2))
colnames(aggregate_funder_count)[2]<-"count_funder"

aggregate_funder_count<- subset(aggregate_funder_count,select=c(funder,count_funder))
training_data_set <- merge(x=training_data_set, y=aggregate_funder_count,by="funder",all=TRUE)
training_data_set["funder"][training_data_set["count_funder"]<500] <- "Other"
training_data_set <- subset(training_data_set,select=-count_funder)
training_data_set_reduced_funder <-  data.frame(training_data_set)
```

In [63]:
```
# Function which performs Ordinal Encoding
encode_ordinal <- function(x, order = unique(x)) {
  x <- as.numeric(factor(x, levels = order, exclude = NULL))
  x
}

# Performing Ordinal Encoding on two features
training_data_set[["quality_group"]] <- encode_ordinal(training_data_set[["quality_group"]])
training_data_set[["quantity_group"]] <- encode_ordinal(training_data_set[["quantity_group"]])

# Viewing the data_set to check if the encoding took place.
head(training_data_set)

# Before performing One-Hot encoding , we have to factorize the features which are specifically strings.
columns <- c("funder","basin","payment","scheme_management","extraction_type_class","management","source","waterpoint_type_group")
training_data_set[columns] <- lapply(training_data_set[columns],as.factor)

# Performing One hot Encoding and assigning it to a variable
dv <- caret::dummyVars("~ funder + basin + payment + scheme_management + extraction_type_class + management + source + waterpoint_type_group", data = training_data_set)
training_data_set_encoded <- data.frame(predict(dv, newdata = training_data_set))


# Removing the Encoded features
training_data_set <- subset(training_data_set, select = -c(funder,basin,payment,scheme_management,extraction_type_class,source,waterpoint_type_group,source,management))

# Copying the values from the encoded data frame into the existing training_data_set.
training_data_set <- cbind(training_data_set,training_data_set_encoded)
```

A data.frame: 6 × 17

	funder	amount_tsh	gps_height	longitude	latitude	basin	population	scheme_management	extraction_type_class	management	payment	quality_group	quantity_group	source	waterpoint_type_group	status_group	age
	<chr>	<dbl>	<int>	<dbl>	<dbl>	<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>	<dbl>	<dbl>	<chr>	<chr>	<fct>	<dbl>
1	0	0	-50	39.39950	-6.865737	Wami / Ruvu	90	SWC	motorpump	other - school	never pay	1	1	machine dbh	communal standpipe	non functional	13
2	0	50	-7	39.19571	-6.903115	Wami / Ruvu	150	Private operator	submersible	private operator	pay per bucket	2	2	machine dbh	communal standpipe	functional	3
3	0	100	90	39.13444	-6.700663	Wami / Ruvu	256	VWC	submersible	vwc	pay per bucket	2	3	river	communal standpipe	functional	3
4	0	50	63	39.12356	-6.919401	Wami / Ruvu	120	Private operator	submersible	private operator	pay per bucket	1	2	machine dbh	communal standpipe	functional	3
5	0	50	-12	39.35191	-6.863978	Wami / Ruvu	30	WUG	submersible	wug	pay per bucket	1	3	machine dbh	communal standpipe	functional	13
6	0	0	95	39.15866	-6.832383	Wami / Ruvu	120	WUG	submersible	wug	never pay	2	2	machine dbh	communal standpipe	non functional	3

Explanation

The next model that we choose is KNN. KNN is a good model for multiclassification. It provides good performance for large datasets. It is one of the simplest classification algorithms.K-nearest neighbours is a non-parametric classification and regression technique. KNN's basic logic is to look around your neighbourhood, assume the test datapoint is identical to them, and extract the result.

We search for k neighbours and make a prediction using KNN. KNN classification uses majority voting over the k closest datapoints, while KNN regression uses the mean of the k closest datapoints as the output. We choose odd numbers as k as a rule of thumb.

We chose KNN because it is an easy and simple machine learning model and has few hyperparameters to tune.

2. KNN

Splitting the test and train dataset

In [64]:
```
df1 <- as.data.frame(sapply(training_data_set, as.numeric)) # Converting the complete dataset into numeric type.
p <- 0.8 # Using only 80% for training and the rest 20% for validation.
train_index <- sample.int(nrow(df1),nrow(df1)*p) # Generating random integers for 80% of the data set
data_train <- as.data.frame(df1[train_index,]) # New data frame with the 80% random indices for training
data_val <- as.data.frame(df1[-train_index,]) # New data frame with the rest 20% random indices for testing
d_label <- data_train$status_group # label which is used for classification
```

Tuning of hyperparameter - count of seed k

Applying different values for the seeds to check how the accuracy score will differ (Hyperparameter Tuning). This part of the code has been commented to reduce the execution time of the report. Please uncomment below cell to execute it.

In [65]:
```
for (i in 3:6) {
  set.seed(5) # For reproducibility purpose because we are using random indices.
  Ypred_knn = knn(train = data_train , test = data_val, cl = d_label, k = i) # perform the KNN prediction
  confusion_matrix <- table(data_val$status_group,Ypred_knn) # Confusion Matrix of the prediction
  accuracy <- ((sum(diag(confusion_matrix)))/sum(confusion_matrix))*100 # Printing the accuracy of the model
  Ypred_knn <- as.numeric(as.character(Ypred_knn)) # Converting the prediction to numeric
  cross_validation <-data.frame(RMSE = RMSE(Ypred_knn,data_train$status_group),MAE= MAE(Ypred_knn,data_train$status_group)) # Performing Cross Validation
  print((paste0('The accuracy of the KNN model when number of neighbors = ',i,' is ', accuracy ,' % '))) # Printing the result
  print(cross_validation) # Printing the result
}
```

```
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
[1] "The accuracy of the KNN model when number of neighbors = 3 is 76.5721020287903 % "
      RMSE       MAE
1 1.344105 0.9621148
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
[1] "The accuracy of the KNN model when number of neighbors = 4 is 76.2353733479249 % "
      RMSE       MAE
1 1.348123 0.9641354
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
[1] "The accuracy of the KNN model when number of neighbors = 5 is 76.7909756713528 % "
      RMSE       MAE
1 1.344402 0.9587473
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
[1] "The accuracy of the KNN model when number of neighbors = 6 is 76.647865981985 % "
      RMSE       MAE
1 1.347123 0.9613992
```

Conclusion from hyperparameter tuning

After looking at various values of k from the above code, we see that the maximum accuracy is obtained from k=5 and so we use that for the final implementation.

In [66]:
```
set.seed(5) # For reproducibility purpose because we are using random indices.
Ypred_knn = knn(train = data_train , test = data_val, cl = d_label, k = 5) # perform the KNN prediction
confusion_matrix <- table(data_val$status_group,Ypred_knn) # Confusion Matrix of the prediction
accuracy <- ((sum(diag(confusion_matrix)))/sum(confusion_matrix))*100 # Printing the accuracy of the model
Ypred_knn <- as.numeric(as.character(Ypred_knn)) # Converting the prediction to numeric
cross_validation <-data.frame(RMSE = RMSE(Ypred_knn,data_train$status_group),MAE= MAE(Ypred_knn,data_train$status_group)) # Performing Cross Validation
print(confusion_matrix)
print((paste0('The accuracy of the KNN model when number of neighbors = 5',' is ', accuracy ,' % '))) # Printing the result
print(cross_validation) # Printing the result
```

```
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
Warning message in pred - obs:
"longer object length is not a multiple of shorter object length"
   Ypred_knn
       1    2    3
  1 5473   94  898
  2  351  342  168
  3 1160   86 3307
[1] "The accuracy of the KNN model when number of neighbors = 5 is 76.7909756713528 % "
      RMSE       MAE
1 1.344402 0.9587473
```

In [67]:
```
print("Recall Per Class  - KNN")
recall_knn <- calc_recall(confusion_matrix)
print(recall_knn)
```

```
[1] "Recall Per Class  - KNN"
[1] 84.65584 39.72125 72.63343
```

In [68]:
```
print("Precision Per Class - KNN")
precision_knn <- calc_precision(confusion_matrix)
print(precision_knn)
```

```
[1] "Precision Per Class - KNN"
[1] 78.36483 65.51724 75.62314
```

In [69]:
```
print("F1 Per Class - KNN")
f1_knn <- calc_f1(precision_knn,recall_knn)
print(f1_knn)
```

```
[1] "F1 Per Class - KNN"
[1] 81.38895 49.45770 74.09814
```

A decision tree is a type of machine learning algorithm that divides data into groups. Partitioning begins with a binary split and continues until no further splits are possible. Various branches of varying lengths are developed. A decision tree's aim is to condense the training data into the smallest possible tree.

A decision tree forces all possible consequences of a decision to be considered, and it tracks each direction to a conclusion. It generates a detailed overview of the implications along each branch and defines decision nodes that need additional investigation.

Decision trees are fast executors. However they do not work well with imbalanced data as observed below.

3. Decision Tree

For decision trees also, we tried executing the same dataset as for Random Forest but it takes too long to execute. So we have reduced the number of values for the funder value and mapped all funders with less than 500 observations as Other

In [70]:
```
# generate a random list of index
shuffle_index <- sample(1:nrow(training_data_set_reduced_funder))

# shuffle the dataset
training_data_set_dt <- training_data_set[shuffle_index, ]
```

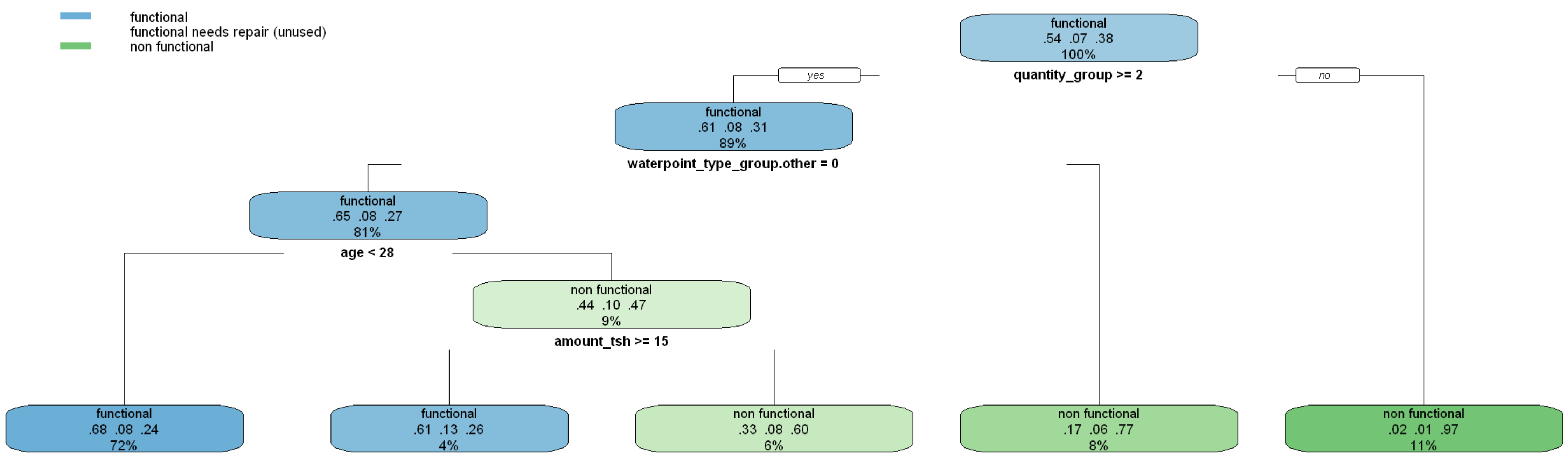
In [71]:
```
# create a function to split the data set to train and test parts
create_train_test <- function(data, size = 0.8, train = TRUE) {
    n_row = nrow(data)
    total_row = size * n_row
    train_sample <- 1: total_row
    if (train == TRUE) {
        return (data[train_sample, ])
    } else {
        return (data[-train_sample, ])
    }
}
```

In [72]:
```
# create train and test data sets and check their dimensions
data_train <- create_train_test(training_data_set_dt, 0.8, train = TRUE)
data_test <- create_train_test(training_data_set_dt, 0.8, train = FALSE)
dim(data_train)
dim(data_test)
```

47512 · 92

11879 · 92

In [73]:
```
# Build the model
fit <- rpart(status_group~., data = data_train, method = 'class')
rpart.plot(fit, extra = 104, varlen = 0)
```



In [74]:
```
# function to get the accuracy
accuracy <- function(fit) {
    predict_unseen <- predict(fit, data_test, type = 'class')
    table_mat <- table(data_test$status_group, predict_unseen)
    accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
    accuracy_Test
}
```

In [75]:
```
# make a prediction from test set
predict_status <-predict(fit, data_test, type = 'class')

# accuracy of test data set
print(paste('Accuracy for test', accuracy(fit)*100))
```

```
[1] "Accuracy for test 70.3678760838454"
```

In [76]:
```
confusion_matrix_decision_tree <- table(data_test$status_group,predict_status)
confusion_matrix_decision_tree
```

```
                         predict_status
                          functional functional needs repair non functional
  functional                    5971                       0            414
  functional needs repair        735                       0            116
  non functional                2255                       0           2388
```

In [77]:
```
print("Recall Per Class - Decision Tree")
recall_dt <- calc_recall(confusion_matrix_decision_tree)
print(recall_dt)
```

```
[1] "Recall Per Class - Decision Tree"
[1] 93.51605  0.00000 51.43226
```

In [78]:
```
print("Precision Per Class - Decision Tree")
precision_dt <- calc_precision(confusion_matrix_decision_tree)
print(precision_dt)
```

```
[1] "Precision Per Class - Decision Tree"
[1] 66.63319      NaN 81.83687
```

In [79]:
```
print("F1 - SCore - Decision Tree")
f1_dt <- calc_f1(precision_dt,recall_dt)
print(f1_dt)
```

```
[1] "F1 - SCore - Decision Tree"
[1] 77.81832      NaN 63.16625
```

Decision Tree gives no prediction as "functional needs repair" because of the large imbalance in the data.

Tuning of hyperparameter

The rpart object has the following default arguments: rpart(minsplit = 20, maxdepth = 30, cp = 0.01). Again the hyperparameter tuning part has been commented for faster execution of the report. Please uncomment the below cells to execute them.

In [80]:
```
cp = 0.1
control <- rpart.control(minsplit = 20, maxdepth = 30, cp = 0.1)
```

```
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 0.688778516710161"
[1] "Difference with default-argument rpart= -0.0149002441282936"
```

In [81]:
```
cp = 0.001
control <- rpart.control(minsplit = 20, maxdepth = 30, cp = 0.001)
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 0.728259954541628"
[1] "Difference with default-argument rpart= 0.0245811937031737"
```

In [82]:
```
minsplit = 30
control <- rpart.control(minsplit = 30, maxdepth = 30, cp = 0.001)
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 0.728259954541628"
[1] "Difference with default-argument rpart= 0.0245811937031737"
```

In [83]:
```
minsplit = 60
control <- rpart.control(minsplit = 60, maxdepth = 30, cp = 0.001)
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 0.728512501052277"
[1] "Difference with default-argument rpart= 0.0248337402138228"
```

In [84]:
```
minsplit = 5
control <- rpart.control(minsplit = 5, maxdepth = 30, cp = 0.001)
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 0.728259954541628"
[1] "Difference with default-argument rpart= 0.0245811937031737"
```

In [85]:
```
maxdepth = 20
control <- rpart.control(minsplit = 5, maxdepth = 20, cp = 0.001)
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 0.728259954541628"
[1] "Difference with default-argument rpart= 0.0245811937031737"
```

In [86]:
```
maxdepth = 10
control <- rpart.control(minsplit = 5, maxdepth = 10, cp = 0.001)
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 0.722956477817998"
[1] "Difference with default-argument rpart= 0.0192777169795437"
```

In [87]:
```
minsplit = 5
maxdepth = 20
cp = 0.0001
control <- rpart.control(minsplit = 5, maxdepth = 20, cp = 0.0001)
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 0.77523360552235"
[1] "Difference with default-argument rpart= 0.0715548446838959"
```

In [88]:
```
minsplit = 5
maxdepth = 20
cp = 0.00001
control <- rpart.control(minsplit = 5, maxdepth = 20, cp = 0.00001)
tune_fit <- rpart(status_group~., data = data_train, method = 'class', control = control)
print(paste('Accuracy for tuning', accuracy(tune_fit)*100))
print(paste('Difference with default-argument rpart=', accuracy(tune_fit) - accuracy(fit)))
```

```
[1] "Accuracy for tuning 76.4963380755956"
[1] "Difference with default-argument rpart= 0.0612846199175016"
```

So, the best accuracy (77.07%) is obtained by applying minsplit, maxdepth, and cp arguments equals 5, 20, and 0.0001, respectively.

We also tried using the Lasso Model but got pretty low accuracy.

4. Lasso Model

The code has been commented for Lasso Model because it isn't one of our top 3 selections. We executed it but the accuracy was not as good as the other models so we removed it.

In [89]:
```
install.packages("glmnet")
library(glmnet)
```

```
Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)

package 'glmnet' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\downloaded_packages
Loading required package: Matrix


Attaching package: 'Matrix'


The following objects are masked from 'package:tidyr':

    expand, pack, unpack


Loaded glmnet 4.1-1
```

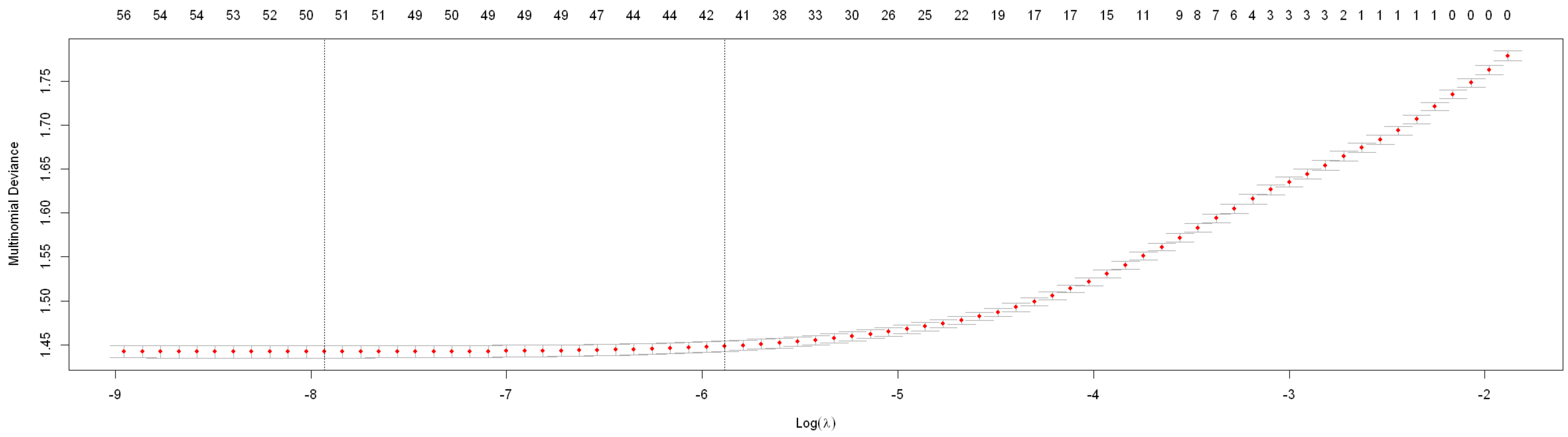
In [91]:
```
# LASSO Model
X <- as.matrix(sapply(subset(training_data_set,select=-status_group),as.numeric)) # Converting the data set as matrix to pass in the LASSO model.
Y <- as.numeric(training_data_set$status_group)  # Converting the labels as numeric.
lasso_fit <- cv.glmnet(X,Y,family="multinomial",nfolds=3) # Fitting the lasso model with our dataset.
```

In [92]:
```
# print(colnames(test_data_lasso))
lasso_predict <- predict(lasso_fit, newx = as.matrix(sapply(subset(training_data_set,select=-status_group),as.numeric)),type="class") # Prediction process of the lasso model takes place in this s
```

In [93]:
```
confusion_matrix1 <- ftable(as.numeric(training_data_set$status_group),lasso_predict)
print(confusion_matrix1)
accuracy1 <- 100* (sum(diag(confusion_matrix1)) / length(training_data_set$status_group)) # Printing the Accuracy of the Model
print(accuracy1)
```

```
  lasso_predict     1     2     3

1               28063    21  4171
2                3365     7   945
3                9903    23 12893
[1] 68.97173
```

In [94]:
```
plot(lasso_fit)
```



Task 3 - Implementing a new model

Explanation

Since from our top 3 models above Random Forest provided a good accuracy we decided to choose an extension of the same to see if we get a better accuracy. Different algorithms such as Adaptive Boosting and Categorical Boosting have been tried. In the end we decided to go with CatBoost because it gave a better performance and accuracy as compared to Adaptive Boosting. Also CatBoost can easily deal with categorical variables and does not require encoding.

CatBoost Algorithm

In [95]:

```
install.packages('devtools')
devtools::install_url('https://github.com/catboost/catboost/releases/download/v0.25.1/catboost-R-Windows-0.25.1.tgz', INSTALL_opts = c("--no-multiarch"))
```

```
Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)

package 'devtools' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\downloaded_packages
WARNING: Rtools is required to build R packages, but is not currently installed.

Please download and install Rtools 4.0 from https://cran.r-project.org/bin/windows/Rtools/.

Downloading package from url: https://github.com/catboost/catboost/releases/download/v0.25.1/catboost-R-Windows-0.25.1.tgz


WARNING: Rtools is required to build R packages, but is not currently installed.

Please download and install Rtools 4.0 from https://cran.r-project.org/bin/windows/Rtools/.

v  checking for file 'C:\Users\HP\AppData\Local\Temp\RtmpaOmOn0\remotes4c3041753334\catboost/DESCRIPTION' (654ms)
-  preparing 'catboost': (726ms)
v  checking DESCRIPTION meta-information
-  checking for LF line-endings in source and make files and shell scripts
-  checking for empty or unneeded directories
-  building 'catboost_0.25.1.tar.gz'


Installing package into 'C:/Users/HP/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
```

In [96]:

```
library(catboost)
```

In [97]:

```
indexes=createDataPartition(training_data_set_without_encoded$status_group, p=.70, list = F)
train_data_catboost <- training_data_set_without_encoded[indexes,]
test_data_catboost <- training_data_set_without_encoded[-indexes,]
train_data_label <- as.integer(train_data_catboost$status_group)-1
test_data_label <- as.integer(test_data_catboost$status_group)-1


col_factor <- c("funder","basin","payment","scheme_management","extraction_type_class","management","source","waterpoint_type_group","quality_group","quantity_group")
train_data_catboost[,col_factor] <- lapply(train_data_catboost[,col_factor],as.factor)
test_data_catboost[,col_factor] <- lapply(test_data_catboost[,col_factor],as.factor)
```

In [98]:

```
train_pool <- catboost.load_pool(data=subset(train_data_catboost,select=-status_group),
                                 label = train_data_label,cat_features=c(2,6,8,9,10,11,12,13,14,15),)
test_pool <- catboost.load_pool(data=subset(test_data_catboost,select=-status_group),
                                label = test_data_label,cat_features=c(2,6,8,9,10,11,12,13,14,15),)
```

Since we have Nvidia in our systems we decided to execute it using GPU for faster execution. Please change GPU to CPU if Nvidia 418.xx or greater is not present.

In [99]:

```
print(Sys.time())
model <- catboost.train(train_pool, test_pool=test_pool,params = list(
task_type = "GPU" ,
loss_function = "MultiClass",
eval_metric="Accuracy",
random_seed = 10,
use_best_model = TRUE
))
print(Sys.time())
```

```
[1] "2021-05-19 00:11:58 IST"
Learning rate set to 0.153226
0:      learn: 0.5995189        test: 0.6056354 best: 0.6056354 (0)     total: 69.7ms   remaining: 1m 9s
1:      learn: 0.6617438        test: 0.6639537 best: 0.6639537 (1)     total: 122ms    remaining: 1m
2:      learn: 0.7044137        test: 0.7103166 best: 0.7103166 (2)     total: 173ms    remaining: 57.4s
3:      learn: 0.7062898        test: 0.7098675 best: 0.7103166 (2)     total: 226ms    remaining: 56.3s
4:      learn: 0.7096813        test: 0.7148069 best: 0.7148069 (4)     total: 274ms    remaining: 54.5s
5:      learn: 0.7131690        test: 0.7174450 best: 0.7174450 (5)     total: 315ms    remaining: 52.3s
6:      learn: 0.7155021        test: 0.7200269 best: 0.7200269 (6)     total: 357ms    remaining: 50.6s
7:      learn: 0.7242574        test: 0.7291199 best: 0.7291199 (7)     total: 396ms    remaining: 49.1s
8:      learn: 0.7237041        test: 0.7286147 best: 0.7291199 (7)     total: 439ms    remaining: 48.3s
9:      learn: 0.7252916        test: 0.7304109 best: 0.7304109 (9)     total: 481ms    remaining: 47.6s
10:     learn: 0.7266146        test: 0.7331051 best: 0.7331051 (10)    total: 514ms    remaining: 46.2s
11:     learn: 0.7305592        test: 0.7365851 best: 0.7365851 (11)    total: 544ms    remaining: 44.8s
12:     learn: 0.7294528        test: 0.7355186 best: 0.7365851 (11)    total: 576ms    remaining: 43.7s
13:     learn: 0.7333013        test: 0.7372025 best: 0.7372025 (13)    total: 606ms    remaining: 42.7s
14:     learn: 0.7340469        test: 0.7387180 best: 0.7387180 (14)    total: 636ms    remaining: 41.8s
15:     learn: 0.7330607        test: 0.7375393 best: 0.7387180 (14)    total: 666ms    remaining: 40.9s
16:     learn: 0.7345761        test: 0.7387180 best: 0.7387180 (14)    total: 700ms    remaining: 40.5s
17:     learn: 0.7361636        test: 0.7405703 best: 0.7405703 (17)    total: 751ms    remaining: 41s
18:     learn: 0.7378232        test: 0.7423103 best: 0.7423103 (18)    total: 799ms    remaining: 41.3s
19:     learn: 0.7385448        test: 0.7420296 best: 0.7423103 (18)    total: 838ms    remaining: 41.1s
20:     learn: 0.7400842        test: 0.7437135 best: 0.7437135 (20)    total: 883ms    remaining: 41.2s
21:     learn: 0.7402766        test: 0.7436574 best: 0.7437135 (20)    total: 923ms    remaining: 41.1s
22:     learn: 0.7411185        test: 0.7446677 best: 0.7446677 (22)    total: 960ms    remaining: 40.8s
23:     learn: 0.7406133        test: 0.7446677 best: 0.7446677 (22)    total: 1000ms   remaining: 40.6s
24:     learn: 0.7419603        test: 0.7452290 best: 0.7452290 (24)    total: 1.04s    remaining: 40.4s
25:     learn: 0.7414793        test: 0.7457903 best: 0.7457903 (25)    total: 1.07s    remaining: 40s
26:     learn: 0.7428503        test: 0.7466884 best: 0.7466884 (26)    total: 1.1s     remaining: 39.5s
27:     learn: 0.7427541        test: 0.7469129 best: 0.7469129 (27)    total: 1.13s    remaining: 39.1s
28:     learn: 0.7438845        test: 0.7478110 best: 0.7478110 (28)    total: 1.16s    remaining: 38.7s
29:     learn: 0.7442934        test: 0.7482039 best: 0.7482039 (29)    total: 1.2s     remaining: 38.7s
30:     learn: 0.7441972        test: 0.7476426 best: 0.7482039 (29)    total: 1.24s    remaining: 38.7s
31:     learn: 0.7449669        test: 0.7482039 best: 0.7482039 (29)    total: 1.28s    remaining: 38.7s
32:     learn: 0.7454961        test: 0.7479793 best: 0.7482039 (29)    total: 1.32s    remaining: 38.7s
33:     learn: 0.7460493        test: 0.7482600 best: 0.7482600 (33)    total: 1.36s    remaining: 38.6s
```

```
34:	learn: 0.7466747	test: 0.7485406 best: 0.7485406 (34)	total: 1.4s	remaining: 38.6s
35:	learn: 0.7473722	test: 0.7482039 best: 0.7485406 (34)	total: 1.44s	remaining: 38.6s
36:	learn: 0.7480938	test: 0.7481477 best: 0.7485406 (34)	total: 1.48s	remaining: 38.6s
37:	learn: 0.7489838	test: 0.7495510 best: 0.7495510 (37)	total: 1.52s	remaining: 38.6s
38:	learn: 0.7496572	test: 0.7498877 best: 0.7498877 (38)	total: 1.56s	remaining: 38.5s
39:	learn: 0.7499459	test: 0.7502806 best: 0.7502806 (39)	total: 1.6s	remaining: 38.5s
40:	learn: 0.7501143	test: 0.7507858 best: 0.7507858 (40)	total: 1.65s	remaining: 38.5s
41:	learn: 0.7501383	test: 0.7506174 best: 0.7507858 (40)	total: 1.69s	remaining: 38.5s
42:	learn: 0.7510283	test: 0.7515716 best: 0.7515716 (42)	total: 1.73s	remaining: 38.4s
43:	learn: 0.7524714	test: 0.7532555 best: 0.7532555 (43)	total: 1.77s	remaining: 38.4s
44:	learn: 0.7528563	test: 0.7535923 best: 0.7535923 (44)	total: 1.81s	remaining: 38.4s
45:	learn: 0.7533614	test: 0.7539852 best: 0.7539852 (45)	total: 1.85s	remaining: 38.4s
46:	learn: 0.7541311	test: 0.7544903 best: 0.7544903 (46)	total: 1.89s	remaining: 38.3s
47:	learn: 0.7541792	test: 0.7543781 best: 0.7544903 (46)	total: 1.93s	remaining: 38.3s
48:	learn: 0.7544438	test: 0.7542097 best: 0.7544903 (46)	total: 1.97s	remaining: 38.2s
49:	learn: 0.7551173	test: 0.7546587 best: 0.7546587 (49)	total: 2.01s	remaining: 38.1s
50:	learn: 0.7559351	test: 0.7549394 best: 0.7549394 (50)	total: 2.04s	remaining: 37.9s
51:	learn: 0.7562237	test: 0.7555007 best: 0.7555007 (51)	total: 2.06s	remaining: 37.6s
52:	learn: 0.7570896	test: 0.7560620 best: 0.7560620 (52)	total: 2.1s	remaining: 37.4s
53:	learn: 0.7577390	test: 0.7562304 best: 0.7562304 (53)	total: 2.13s	remaining: 37.2s
54:	learn: 0.7577390	test: 0.7565671 best: 0.7565671 (54)	total: 2.16s	remaining: 37.1s
55:	learn: 0.7576428	test: 0.7563987 best: 0.7565671 (54)	total: 2.19s	remaining: 36.9s
56:	learn: 0.7581479	test: 0.7561181 best: 0.7565671 (54)	total: 2.22s	remaining: 36.7s
57:	learn: 0.7586290	test: 0.7563987 best: 0.7565671 (54)	total: 2.25s	remaining: 36.5s
58:	learn: 0.7588695	test: 0.7570162 best: 0.7570162 (58)	total: 2.28s	remaining: 36.3s
59:	learn: 0.7589176	test: 0.7567355 best: 0.7570162 (58)	total: 2.31s	remaining: 36.1s
60:	learn: 0.7595430	test: 0.7565110 best: 0.7570162 (58)	total: 2.33s	remaining: 35.9s
61:	learn: 0.7595911	test: 0.7565671 best: 0.7570162 (58)	total: 2.36s	remaining: 35.8s
62:	learn: 0.7594227	test: 0.7575775 best: 0.7575775 (62)	total: 2.39s	remaining: 35.6s
63:	learn: 0.7597595	test: 0.7578581 best: 0.7578581 (63)	total: 2.43s	remaining: 35.5s
64:	learn: 0.7603127	test: 0.7580265 best: 0.7580265 (64)	total: 2.46s	remaining: 35.4s
65:	learn: 0.7602646	test: 0.7579142 best: 0.7580265 (64)	total: 2.49s	remaining: 35.3s
66:	learn: 0.7602405	test: 0.7574091 best: 0.7580265 (64)	total: 2.52s	remaining: 35.1s
67:	learn: 0.7604089	test: 0.7584194 best: 0.7584194 (67)	total: 2.55s	remaining: 35s
68:	learn: 0.7612508	test: 0.7586439 best: 0.7586439 (68)	total: 2.58s	remaining: 34.8s
69:	learn: 0.7617078	test: 0.7586439 best: 0.7586439 (68)	total: 2.61s	remaining: 34.7s
70:	learn: 0.7618040	test: 0.7586439 best: 0.7586439 (68)	total: 2.64s	remaining: 34.6s
71:	learn: 0.7618280	test: 0.7585317 best: 0.7586439 (68)	total: 2.67s	remaining: 34.4s
72:	learn: 0.7624293	test: 0.7585317 best: 0.7586439 (68)	total: 2.7s	remaining: 34.3s
73:	learn: 0.7624534	test: 0.7589807 best: 0.7589807 (73)	total: 2.73s	remaining: 34.2s
74:	learn: 0.7625256	test: 0.7587000 best: 0.7589807 (73)	total: 2.76s	remaining: 34s
75:	learn: 0.7630547	test: 0.7598226 best: 0.7598226 (75)	total: 2.78s	remaining: 33.8s
76:	learn: 0.7629585	test: 0.7593175 best: 0.7598226 (75)	total: 2.82s	remaining: 33.8s
77:	learn: 0.7631509	test: 0.7594297 best: 0.7598226 (75)	total: 2.85s	remaining: 33.7s
78:	learn: 0.7633674	test: 0.7595981 best: 0.7598226 (75)	total: 2.88s	remaining: 33.6s
79:	learn: 0.7637041	test: 0.7604401 best: 0.7604401 (79)	total: 2.91s	remaining: 33.5s
80:	learn: 0.7637282	test: 0.7610013 best: 0.7610013 (80)	total: 2.94s	remaining: 33.3s
81:	learn: 0.7639687	test: 0.7605523 best: 0.7610013 (80)	total: 2.97s	remaining: 33.2s
82:	learn: 0.7641852	test: 0.7606646 best: 0.7610013 (80)	total: 3s	remaining: 33.1s
83:	learn: 0.7649549	test: 0.7610013 best: 0.7610013 (80)	total: 3.03s	remaining: 33s
84:	learn: 0.7651954	test: 0.7612259 best: 0.7612259 (84)	total: 3.06s	remaining: 32.9s
85:	learn: 0.7658930	test: 0.7616749 best: 0.7616749 (85)	total: 3.09s	remaining: 32.8s
86:	learn: 0.7658689	test: 0.7620117 best: 0.7620117 (86)	total: 3.14s	remaining: 32.9s
87:	learn: 0.7665183	test: 0.7623485 best: 0.7623485 (87)	total: 3.19s	remaining: 33s
88:	learn: 0.7673361	test: 0.7631904 best: 0.7631904 (88)	total: 3.23s	remaining: 33.1s
89:	learn: 0.7675526	test: 0.7629659 best: 0.7631904 (88)	total: 3.27s	remaining: 33.1s
90:	learn: 0.7686109	test: 0.7636956 best: 0.7636956 (90)	total: 3.31s	remaining: 33.1s
91:	learn: 0.7690198	test: 0.7633026 best: 0.7636956 (90)	total: 3.34s	remaining: 32.9s
92:	learn: 0.7690439	test: 0.7638639 best: 0.7638639 (92)	total: 3.36s	remaining: 32.8s
93:	learn: 0.7697895	test: 0.7640323 best: 0.7640323 (93)	total: 3.39s	remaining: 32.7s
94:	learn: 0.7698376	test: 0.7646498 best: 0.7646498 (94)	total: 3.43s	remaining: 32.7s
95:	learn: 0.7701503	test: 0.7642568 best: 0.7646498 (94)	total: 3.46s	remaining: 32.6s
96:	learn: 0.7701263	test: 0.7647620 best: 0.7647620 (96)	total: 3.49s	remaining: 32.5s
97:	learn: 0.7704871	test: 0.7653794 best: 0.7653794 (97)	total: 3.52s	remaining: 32.4s
98:	learn: 0.7707517	test: 0.7657723 best: 0.7657723 (98)	total: 3.55s	remaining: 32.3s
99:	learn: 0.7709922	test: 0.7657162 best: 0.7657723 (98)	total: 3.58s	remaining: 32.2s
100:	learn: 0.7719543	test: 0.7667265 best: 0.7667265 (100)	total: 3.61s	remaining: 32.1s
101:	learn: 0.7719062	test: 0.7670633 best: 0.7670633 (101)	total: 3.64s	remaining: 32s
102:	learn: 0.7720505	test: 0.7675685 best: 0.7675685 (102)	total: 3.67s	remaining: 31.9s
103:	learn: 0.7724113	test: 0.7671194 best: 0.7675685 (102)	total: 3.7s	remaining: 31.9s
104:	learn: 0.7725797	test: 0.7675123 best: 0.7675685 (102)	total: 3.73s	remaining: 31.8s
105:	learn: 0.7729405	test: 0.7680175 best: 0.7680175 (105)	total: 3.76s	remaining: 31.7s
106:	learn: 0.7731810	test: 0.7675685 best: 0.7680175 (105)	total: 3.79s	remaining: 31.6s
107:	learn: 0.7734937	test: 0.7682420 best: 0.7682420 (107)	total: 3.82s	remaining: 31.5s
108:	learn: 0.7736621	test: 0.7681298 best: 0.7682420 (107)	total: 3.85s	remaining: 31.4s
109:	learn: 0.7738545	test: 0.7676246 best: 0.7682420 (107)	total: 3.88s	remaining: 31.4s
110:	learn: 0.7737102	test: 0.7675123 best: 0.7682420 (107)	total: 3.91s	remaining: 31.3s
111:	learn: 0.7742153	test: 0.7674562 best: 0.7682420 (107)	total: 3.94s	remaining: 31.2s
112:	learn: 0.7743115	test: 0.7678491 best: 0.7682420 (107)	total: 3.97s	remaining: 31.1s
113:	learn: 0.7740469	test: 0.7681298 best: 0.7682420 (107)	total: 4s	remaining: 31.1s
114:	learn: 0.7745520	test: 0.7688033 best: 0.7688033 (114)	total: 4.03s	remaining: 31s
115:	learn: 0.7754660	test: 0.7701504 best: 0.7701504 (115)	total: 4.05s	remaining: 30.9s
116:	learn: 0.7756344	test: 0.7702627 best: 0.7702627 (116)	total: 4.08s	remaining: 30.8s
117:	learn: 0.7754420	test: 0.7709924 best: 0.7709924 (117)	total: 4.11s	remaining: 30.7s
118:	learn: 0.7753939	test: 0.7700943 best: 0.7709924 (117)	total: 4.14s	remaining: 30.6s
119:	learn: 0.7754420	test: 0.7704311 best: 0.7709924 (117)	total: 4.17s	remaining: 30.6s
120:	learn: 0.7755863	test: 0.7708801 best: 0.7709924 (117)	total: 4.19s	remaining: 30.5s
121:	learn: 0.7756825	test: 0.7702627 best: 0.7709924 (117)	total: 4.22s	remaining: 30.4s
122:	learn: 0.7757306	test: 0.7696453 best: 0.7709924 (117)	total: 4.26s	remaining: 30.4s
123:	learn: 0.7758509	test: 0.7700382 best: 0.7709924 (117)	total: 4.29s	remaining: 30.3s
124:	learn: 0.7761155	test: 0.7703188 best: 0.7709924 (117)	total: 4.32s	remaining: 30.2s
125:	learn: 0.7773181	test: 0.7713291 best: 0.7713291 (125)	total: 4.34s	remaining: 30.1s
126:	learn: 0.7773422	test: 0.7714414 best: 0.7714414 (126)	total: 4.37s	remaining: 30s
127:	learn: 0.7773422	test: 0.7711046 best: 0.7714414 (126)	total: 4.39s	remaining: 29.9s
128:	learn: 0.7773181	test: 0.7712169 best: 0.7714414 (126)	total: 4.42s	remaining: 29.8s
129:	learn: 0.7775827	test: 0.7714975 best: 0.7714975 (129)	total: 4.45s	remaining: 29.8s
130:	learn: 0.7779916	test: 0.7714414 best: 0.7714975 (129)	total: 4.47s	remaining: 29.7s
131:	learn: 0.7780878	test: 0.7714975 best: 0.7714975 (129)	total: 4.5s	remaining: 29.6s
132:	learn: 0.7782321	test: 0.7720027 best: 0.7720027 (132)	total: 4.53s	remaining: 29.6s
133:	learn: 0.7782081	test: 0.7722833 best: 0.7722833 (133)	total: 4.58s	remaining: 29.6s
134:	learn: 0.7783764	test: 0.7723395 best: 0.7723395 (134)	total: 4.64s	remaining: 29.8s
135:	learn: 0.7786891	test: 0.7727885 best: 0.7727885 (135)	total: 4.7s	remaining: 29.9s
136:	learn: 0.7788575	test: 0.7724517 best: 0.7727885 (135)	total: 4.77s	remaining: 30s
137:	learn: 0.7789296	test: 0.7726762 best: 0.7727885 (135)	total: 4.82s	remaining: 30.1s
138:	learn: 0.7787613	test: 0.7723956 best: 0.7727885 (135)	total: 4.87s	remaining: 30.2s
139:	learn: 0.7790018	test: 0.7725079 best: 0.7727885 (135)	total: 4.92s	remaining: 30.2s
140:	learn: 0.7791702	test: 0.7723395 best: 0.7727885 (135)	total: 4.96s	remaining: 30.2s
141:	learn: 0.7791942	test: 0.7717220 best: 0.7727885 (135)	total: 5.02s	remaining: 30.3s
142:	learn: 0.7792423	test: 0.7716098 best: 0.7727885 (135)	total: 5.08s	remaining: 30.4s
143:	learn: 0.7792423	test: 0.7717220 best: 0.7727885 (135)	total: 5.12s	remaining: 30.5s
144:	learn: 0.7793145	test: 0.7720588 best: 0.7727885 (135)	total: 5.17s	remaining: 30.5s
145:	learn: 0.7794348	test: 0.7723395 best: 0.7727885 (135)	total: 5.22s	remaining: 30.5s
146:	learn: 0.7797956	test: 0.7729008 best: 0.7729008 (146)	total: 5.27s	remaining: 30.6s
147:	learn: 0.7801563	test: 0.7729569 best: 0.7729569 (147)	total: 5.32s	remaining: 30.6s
148:	learn: 0.7804690	test: 0.7727885 best: 0.7729569 (147)	total: 5.36s	remaining: 30.6s
149:	learn: 0.7804690	test: 0.7728446 best: 0.7729569 (147)	total: 5.4s	remaining: 30.6s
150:	learn: 0.7804931	test: 0.7726201 best: 0.7729569 (147)	total: 5.44s	remaining: 30.6s
151:	learn: 0.7806374	test: 0.7728446 best: 0.7729569 (147)	total: 5.49s	remaining: 30.6s
152:	learn: 0.7805893	test: 0.7729008 best: 0.7729569 (147)	total: 5.54s	remaining: 30.6s
153:	learn: 0.7809260	test: 0.7730692 best: 0.7730692 (153)	total: 5.58s	remaining: 30.7s
154:	learn: 0.7813109	test: 0.7735182 best: 0.7735182 (154)	total: 5.62s	remaining: 30.6s
155:	learn: 0.7813590	test: 0.7741917 best: 0.7741917 (155)	total: 5.65s	remaining: 30.6s
156:	learn: 0.7820565	test: 0.7749214 best: 0.7749214 (156)	total: 5.67s	remaining: 30.5s
157:	learn: 0.7822971	test: 0.7750898 best: 0.7750898 (157)	total: 5.71s	remaining: 30.4s
158:	learn: 0.7824895	test: 0.7753705 best: 0.7753705 (158)	total: 5.74s	remaining: 30.3s
159:	learn: 0.7827060	test: 0.7753705 best: 0.7753705 (158)	total: 5.76s	remaining: 30.3s
160:	learn: 0.7828262	test: 0.7753143 best: 0.7753705 (158)	total: 5.79s	remaining: 30.2s
161:	learn: 0.7829224	test: 0.7753143 best: 0.7753705 (158)	total: 5.82s	remaining: 30.1s
162:	learn: 0.7833073	test: 0.7754266 best: 0.7754266 (162)	total: 5.87s	remaining: 30.1s
163:	learn: 0.7834997	test: 0.7757072 best: 0.7757072 (163)	total: 5.92s	remaining: 30.2s
164:	learn: 0.7841251	test: 0.7759879 best: 0.7759879 (164)	total: 5.97s	remaining: 30.2s
165:	learn: 0.7844378	test: 0.7758195 best: 0.7759879 (164)	total: 6.02s	remaining: 30.2s
166:	learn: 0.7846542	test: 0.7759317 best: 0.7759879 (164)	total: 6.07s	remaining: 30.3s
167:	learn: 0.7848467	test: 0.7758195 best: 0.7759879 (164)	total: 6.12s	remaining: 30.3s
168:	learn: 0.7848948	test: 0.7757072 best: 0.7759879 (164)	total: 6.16s	remaining: 30.3s
169:	learn: 0.7854239	test: 0.7761001 best: 0.7761001 (169)	total: 6.21s	remaining: 30.3s
170:	learn: 0.7852075	test: 0.7764930 best: 0.7764930 (170)	total: 6.25s	remaining: 30.3s
171:	learn: 0.7851834	test: 0.7758195 best: 0.7764930 (170)	total: 6.3s	remaining: 30.3s
172:	learn: 0.7854239	test: 0.7761563 best: 0.7764930 (170)	total: 6.35s	remaining: 30.3s
173:	learn: 0.7856164	test: 0.7759879 best: 0.7764930 (170)	total: 6.4s	remaining: 30.4s
174:	learn: 0.7857607	test: 0.7759879 best: 0.7764930 (170)	total: 6.44s	remaining: 30.4s
175:	learn: 0.7860253	test: 0.7759879 best: 0.7764930 (170)	total: 6.49s	remaining: 30.4s
176:	learn: 0.7860012	test: 0.7757634 best: 0.7764930 (170)	total: 6.54s	remaining: 30.4s
177:	learn: 0.7863620	test: 0.7757634 best: 0.7764930 (170)	total: 6.58s	remaining: 30.4s
```

```
178:	learn: 0.7865304	test: 0.7759317 best: 0.7764930 (170)	total: 6.61s	remaining: 30.3s
179:	learn: 0.7865063	test: 0.7763808 best: 0.7764930 (170)	total: 6.64s	remaining: 30.3s
180:	learn: 0.7866747	test: 0.7764369 best: 0.7764930 (170)	total: 6.68s	remaining: 30.2s
181:	learn: 0.7871798	test: 0.7766053 best: 0.7766053 (181)	total: 6.71s	remaining: 30.2s
182:	learn: 0.7870836	test: 0.7763247 best: 0.7766053 (181)	total: 6.74s	remaining: 30.1s
183:	learn: 0.7875406	test: 0.7768298 best: 0.7768298 (183)	total: 6.77s	remaining: 30s
184:	learn: 0.7874684	test: 0.7770543 best: 0.7770543 (184)	total: 6.8s	remaining: 30s
185:	learn: 0.7875406	test: 0.7771666 best: 0.7771666 (185)	total: 6.83s	remaining: 29.9s
186:	learn: 0.7877330	test: 0.7776156 best: 0.7776156 (186)	total: 6.87s	remaining: 29.8s
187:	learn: 0.7877330	test: 0.7772789 best: 0.7776156 (186)	total: 6.89s	remaining: 29.8s
188:	learn: 0.7878773	test: 0.7774472 best: 0.7776156 (186)	total: 6.92s	remaining: 29.7s
189:	learn: 0.7881900	test: 0.7772227 best: 0.7776156 (186)	total: 6.95s	remaining: 29.6s
190:	learn: 0.7879254	test: 0.7773911 best: 0.7776156 (186)	total: 6.99s	remaining: 29.6s
191:	learn: 0.7880698	test: 0.7775034 best: 0.7776156 (186)	total: 7.02s	remaining: 29.6s
192:	learn: 0.7883584	test: 0.7778401 best: 0.7778401 (192)	total: 7.08s	remaining: 29.6s
193:	learn: 0.7885268	test: 0.7779524 best: 0.7779524 (193)	total: 7.12s	remaining: 29.6s
194:	learn: 0.7886951	test: 0.7778963 best: 0.7779524 (193)	total: 7.16s	remaining: 29.5s
195:	learn: 0.7889597	test: 0.7781769 best: 0.7781769 (195)	total: 7.21s	remaining: 29.6s
196:	learn: 0.7889357	test: 0.7781769 best: 0.7781769 (195)	total: 7.25s	remaining: 29.5s
197:	learn: 0.7890319	test: 0.7782330 best: 0.7782330 (197)	total: 7.29s	remaining: 29.5s
198:	learn: 0.7885749	test: 0.7785698 best: 0.7785698 (198)	total: 7.34s	remaining: 29.5s
199:	learn: 0.7885749	test: 0.7789066 best: 0.7789066 (199)	total: 7.38s	remaining: 29.5s
200:	learn: 0.7886230	test: 0.7784576 best: 0.7789066 (199)	total: 7.42s	remaining: 29.5s
201:	learn: 0.7886230	test: 0.7784576 best: 0.7789066 (199)	total: 7.45s	remaining: 29.4s
202:	learn: 0.7885027	test: 0.7786821 best: 0.7789066 (199)	total: 7.47s	remaining: 29.3s
203:	learn: 0.7886470	test: 0.7788505 best: 0.7789066 (199)	total: 7.5s	remaining: 29.3s
204:	learn: 0.7888154	test: 0.7787943 best: 0.7789066 (199)	total: 7.53s	remaining: 29.2s
205:	learn: 0.7890559	test: 0.7782892 best: 0.7789066 (199)	total: 7.55s	remaining: 29.1s
206:	learn: 0.7892724	test: 0.7782330 best: 0.7789066 (199)	total: 7.58s	remaining: 29s
207:	learn: 0.7893927	test: 0.7778963 best: 0.7789066 (199)	total: 7.6s	remaining: 29s
208:	learn: 0.7895370	test: 0.7783453 best: 0.7789066 (199)	total: 7.63s	remaining: 28.9s
209:	learn: 0.7894889	test: 0.7790750 best: 0.7790750 (209)	total: 7.66s	remaining: 28.8s
210:	learn: 0.7894889	test: 0.7792434 best: 0.7792434 (210)	total: 7.68s	remaining: 28.7s
211:	learn: 0.7895129	test: 0.7793556 best: 0.7793556 (211)	total: 7.71s	remaining: 28.7s
212:	learn: 0.7896813	test: 0.7792434 best: 0.7793556 (211)	total: 7.74s	remaining: 28.6s
213:	learn: 0.7900902	test: 0.7798047 best: 0.7798047 (213)	total: 7.77s	remaining: 28.5s
214:	learn: 0.7901864	test: 0.7791311 best: 0.7798047 (213)	total: 7.81s	remaining: 28.5s
215:	learn: 0.7902105	test: 0.7794118 best: 0.7798047 (213)	total: 7.86s	remaining: 28.5s
216:	learn: 0.7904991	test: 0.7793556 best: 0.7798047 (213)	total: 7.9s	remaining: 28.5s
217:	learn: 0.7906194	test: 0.7795240 best: 0.7798047 (213)	total: 7.95s	remaining: 28.5s
218:	learn: 0.7907637	test: 0.7802537 best: 0.7802537 (218)	total: 7.99s	remaining: 28.5s
219:	learn: 0.7912207	test: 0.7800853 best: 0.7802537 (218)	total: 8.03s	remaining: 28.5s
220:	learn: 0.7913410	test: 0.7803098 best: 0.7803098 (220)	total: 8.08s	remaining: 28.5s
221:	learn: 0.7913410	test: 0.7803660 best: 0.7803660 (221)	total: 8.13s	remaining: 28.5s
222:	learn: 0.7911485	test: 0.7801976 best: 0.7803660 (221)	total: 8.18s	remaining: 28.5s
223:	learn: 0.7913650	test: 0.7808150 best: 0.7808150 (223)	total: 8.23s	remaining: 28.5s
224:	learn: 0.7916777	test: 0.7810395 best: 0.7810395 (224)	total: 8.27s	remaining: 28.5s
225:	learn: 0.7915574	test: 0.7807589 best: 0.7810395 (224)	total: 8.32s	remaining: 28.5s
226:	learn: 0.7919904	test: 0.7808150 best: 0.7810395 (224)	total: 8.37s	remaining: 28.5s
227:	learn: 0.7920625	test: 0.7809834 best: 0.7810395 (224)	total: 8.41s	remaining: 28.5s
228:	learn: 0.7921587	test: 0.7809834 best: 0.7810395 (224)	total: 8.46s	remaining: 28.5s
229:	learn: 0.7925195	test: 0.7812640 best: 0.7812640 (229)	total: 8.51s	remaining: 28.5s
230:	learn: 0.7925436	test: 0.7809834 best: 0.7812640 (229)	total: 8.56s	remaining: 28.5s
231:	learn: 0.7927601	test: 0.7813202 best: 0.7813202 (231)	total: 8.6s	remaining: 28.5s
232:	learn: 0.7926639	test: 0.7810956 best: 0.7813202 (231)	total: 8.64s	remaining: 28.4s
233:	learn: 0.7926158	test: 0.7810956 best: 0.7813202 (231)	total: 8.68s	remaining: 28.4s
234:	learn: 0.7926158	test: 0.7813202 best: 0.7813202 (231)	total: 8.72s	remaining: 28.4s
235:	learn: 0.7927601	test: 0.7811518 best: 0.7813202 (231)	total: 8.76s	remaining: 28.4s
236:	learn: 0.7928563	test: 0.7812079 best: 0.7813202 (231)	total: 8.79s	remaining: 28.3s
237:	learn: 0.7928322	test: 0.7808150 best: 0.7813202 (231)	total: 8.82s	remaining: 28.2s
238:	learn: 0.7929044	test: 0.7806466 best: 0.7813202 (231)	total: 8.85s	remaining: 28.2s
239:	learn: 0.7928563	test: 0.7812640 best: 0.7813202 (231)	total: 8.87s	remaining: 28.1s
240:	learn: 0.7926398	test: 0.7810395 best: 0.7813202 (231)	total: 8.9s	remaining: 28s
241:	learn: 0.7927360	test: 0.7811518 best: 0.7813202 (231)	total: 8.93s	remaining: 28s
242:	learn: 0.7931930	test: 0.7814885 best: 0.7814885 (242)	total: 8.96s	remaining: 27.9s
243:	learn: 0.7932652	test: 0.7816569 best: 0.7816569 (243)	total: 8.98s	remaining: 27.8s
244:	learn: 0.7933373	test: 0.7817692 best: 0.7817692 (244)	total: 9.01s	remaining: 27.8s
245:	learn: 0.7936019	test: 0.7819937 best: 0.7819937 (245)	total: 9.04s	remaining: 27.7s
246:	learn: 0.7939146	test: 0.7821060 best: 0.7821060 (246)	total: 9.07s	remaining: 27.7s
247:	learn: 0.7938665	test: 0.7820498 best: 0.7821060 (246)	total: 9.11s	remaining: 27.6s
248:	learn: 0.7940349	test: 0.7817131 best: 0.7821060 (246)	total: 9.14s	remaining: 27.6s
249:	learn: 0.7942514	test: 0.7818253 best: 0.7821060 (246)	total: 9.17s	remaining: 27.5s
250:	learn: 0.7943235	test: 0.7821621 best: 0.7821621 (250)	total: 9.2s	remaining: 27.5s
251:	learn: 0.7943957	test: 0.7818253 best: 0.7821621 (250)	total: 9.23s	remaining: 27.4s
252:	learn: 0.7945881	test: 0.7819376 best: 0.7821621 (250)	total: 9.26s	remaining: 27.3s
253:	learn: 0.7946121	test: 0.7819376 best: 0.7821621 (250)	total: 9.29s	remaining: 27.3s
254:	learn: 0.7947324	test: 0.7818253 best: 0.7821621 (250)	total: 9.32s	remaining: 27.2s
255:	learn: 0.7945881	test: 0.7817131 best: 0.7821621 (250)	total: 9.35s	remaining: 27.2s
256:	learn: 0.7948046	test: 0.7819937 best: 0.7821621 (250)	total: 9.38s	remaining: 27.1s
257:	learn: 0.7945881	test: 0.7818815 best: 0.7821621 (250)	total: 9.41s	remaining: 27.1s
258:	learn: 0.7948286	test: 0.7819937 best: 0.7821621 (250)	total: 9.44s	remaining: 27s
259:	learn: 0.7950210	test: 0.7816569 best: 0.7821621 (250)	total: 9.47s	remaining: 27s
260:	learn: 0.7950210	test: 0.7818253 best: 0.7821621 (250)	total: 9.51s	remaining: 26.9s
261:	learn: 0.7949970	test: 0.7817131 best: 0.7821621 (250)	total: 9.55s	remaining: 26.9s
262:	learn: 0.7949008	test: 0.7820498 best: 0.7821621 (250)	total: 9.58s	remaining: 26.9s
263:	learn: 0.7948286	test: 0.7818253 best: 0.7821621 (250)	total: 9.61s	remaining: 26.8s
264:	learn: 0.7950932	test: 0.7817692 best: 0.7821621 (250)	total: 9.64s	remaining: 26.7s
265:	learn: 0.7952616	test: 0.7815447 best: 0.7821621 (250)	total: 9.68s	remaining: 26.7s
266:	learn: 0.7954299	test: 0.7813202 best: 0.7821621 (250)	total: 9.7s	remaining: 26.6s
267:	learn: 0.7955262	test: 0.7810956 best: 0.7821621 (250)	total: 9.73s	remaining: 26.6s
268:	learn: 0.7954059	test: 0.7810395 best: 0.7821621 (250)	total: 9.76s	remaining: 26.5s
269:	learn: 0.7957186	test: 0.7809273 best: 0.7821621 (250)	total: 9.79s	remaining: 26.5s
270:	learn: 0.7956705	test: 0.7809273 best: 0.7821621 (250)	total: 9.82s	remaining: 26.4s
271:	learn: 0.7957186	test: 0.7810956 best: 0.7821621 (250)	total: 9.86s	remaining: 26.4s
272:	learn: 0.7960313	test: 0.7811518 best: 0.7821621 (250)	total: 9.89s	remaining: 26.3s
273:	learn: 0.7960313	test: 0.7812079 best: 0.7821621 (250)	total: 9.93s	remaining: 26.3s
274:	learn: 0.7957667	test: 0.7812079 best: 0.7821621 (250)	total: 9.96s	remaining: 26.3s
275:	learn: 0.7959110	test: 0.7812640 best: 0.7821621 (250)	total: 9.99s	remaining: 26.2s
276:	learn: 0.7959351	test: 0.7815447 best: 0.7821621 (250)	total: 10s	remaining: 26.2s
277:	learn: 0.7961996	test: 0.7818815 best: 0.7821621 (250)	total: 10.1s	remaining: 26.1s
278:	learn: 0.7961996	test: 0.7815447 best: 0.7821621 (250)	total: 10.1s	remaining: 26.1s
279:	learn: 0.7964161	test: 0.7813202 best: 0.7821621 (250)	total: 10.1s	remaining: 26s
280:	learn: 0.7964883	test: 0.7808150 best: 0.7821621 (250)	total: 10.2s	remaining: 26s
281:	learn: 0.7964883	test: 0.7808711 best: 0.7821621 (250)	total: 10.2s	remaining: 25.9s
282:	learn: 0.7965604	test: 0.7810956 best: 0.7821621 (250)	total: 10.2s	remaining: 25.9s
283:	learn: 0.7969934	test: 0.7809273 best: 0.7821621 (250)	total: 10.2s	remaining: 25.8s
284:	learn: 0.7969212	test: 0.7809834 best: 0.7821621 (250)	total: 10.3s	remaining: 25.8s
285:	learn: 0.7969693	test: 0.7812640 best: 0.7821621 (250)	total: 10.3s	remaining: 25.7s
286:	learn: 0.7969693	test: 0.7814885 best: 0.7821621 (250)	total: 10.3s	remaining: 25.7s
287:	learn: 0.7970896	test: 0.7816008 best: 0.7821621 (250)	total: 10.4s	remaining: 25.6s
288:	learn: 0.7970655	test: 0.7816008 best: 0.7821621 (250)	total: 10.4s	remaining: 25.6s
289:	learn: 0.7971137	test: 0.7813202 best: 0.7821621 (250)	total: 10.5s	remaining: 25.6s
290:	learn: 0.7973061	test: 0.7814885 best: 0.7821621 (250)	total: 10.5s	remaining: 25.6s
291:	learn: 0.7975947	test: 0.7816569 best: 0.7821621 (250)	total: 10.5s	remaining: 25.6s
292:	learn: 0.7976188	test: 0.7819937 best: 0.7821621 (250)	total: 10.6s	remaining: 25.5s
293:	learn: 0.7975947	test: 0.7818815 best: 0.7821621 (250)	total: 10.6s	remaining: 25.5s
294:	learn: 0.7974985	test: 0.7819937 best: 0.7821621 (250)	total: 10.7s	remaining: 25.5s
295:	learn: 0.7977390	test: 0.7819376 best: 0.7821621 (250)	total: 10.7s	remaining: 25.4s
296:	learn: 0.7975947	test: 0.7818253 best: 0.7821621 (250)	total: 10.7s	remaining: 25.4s
297:	learn: 0.7976188	test: 0.7818815 best: 0.7821621 (250)	total: 10.8s	remaining: 25.4s
298:	learn: 0.7977150	test: 0.7817131 best: 0.7821621 (250)	total: 10.8s	remaining: 25.4s
299:	learn: 0.7980758	test: 0.7815447 best: 0.7821621 (250)	total: 10.9s	remaining: 25.3s
300:	learn: 0.7983163	test: 0.7818815 best: 0.7821621 (250)	total: 10.9s	remaining: 25.3s
301:	learn: 0.7982682	test: 0.7819376 best: 0.7821621 (250)	total: 10.9s	remaining: 25.3s
302:	learn: 0.7985087	test: 0.7821060 best: 0.7821621 (250)	total: 11s	remaining: 25.2s
303:	learn: 0.7986049	test: 0.7820498 best: 0.7821621 (250)	total: 11s	remaining: 25.2s
304:	learn: 0.7987492	test: 0.7818815 best: 0.7821621 (250)	total: 11s	remaining: 25.1s
305:	learn: 0.7985087	test: 0.7817692 best: 0.7821621 (250)	total: 11.1s	remaining: 25.1s
306:	learn: 0.7986771	test: 0.7819937 best: 0.7821621 (250)	total: 11.1s	remaining: 25s
307:	learn: 0.7987492	test: 0.7819937 best: 0.7821621 (250)	total: 11.1s	remaining: 25s
308:	learn: 0.7986290	test: 0.7821060 best: 0.7821621 (250)	total: 11.1s	remaining: 24.9s
309:	learn: 0.7987252	test: 0.7820498 best: 0.7821621 (250)	total: 11.2s	remaining: 24.9s
310:	learn: 0.7990138	test: 0.7817131 best: 0.7821621 (250)	total: 11.2s	remaining: 24.8s
311:	learn: 0.7993506	test: 0.7820498 best: 0.7821621 (250)	total: 11.2s	remaining: 24.8s
312:	learn: 0.7995430	test: 0.7827795 best: 0.7827795 (312)	total: 11.3s	remaining: 24.7s
313:	learn: 0.7996392	test: 0.7826673 best: 0.7827795 (312)	total: 11.3s	remaining: 24.7s
314:	learn: 0.7998316	test: 0.7829479 best: 0.7829479 (314)	total: 11.3s	remaining: 24.6s
315:	learn: 0.8000481	test: 0.7827795 best: 0.7829479 (314)	total: 11.3s	remaining: 24.6s
316:	learn: 0.7998076	test: 0.7832286 best: 0.7832286 (316)	total: 11.4s	remaining: 24.5s
317:	learn: 0.7998076	test: 0.7830040 best: 0.7832286 (316)	total: 11.4s	remaining: 24.5s
318:	learn: 0.8003608	test: 0.7830602 best: 0.7832286 (316)	total: 11.4s	remaining: 24.4s
319:	learn: 0.8003127	test: 0.7833969 best: 0.7833969 (319)	total: 11.5s	remaining: 24.4s
320:	learn: 0.8000722	test: 0.7834531 best: 0.7834531 (320)	total: 11.5s	remaining: 24.3s
321:	learn: 0.8002165	test: 0.7830040 best: 0.7834531 (320)	total: 11.5s	remaining: 24.3s
```

```
322:	learn: 0.7999759	test: 0.7831724 best: 0.7834531 (320)	total: 11.6s	remaining: 24.2s
323:	learn: 0.8001684	test: 0.7831724 best: 0.7834531 (320)	total: 11.6s	remaining: 24.2s
324:	learn: 0.8002646	test: 0.7834531 best: 0.7834531 (320)	total: 11.6s	remaining: 24.1s
325:	learn: 0.8003127	test: 0.7834531 best: 0.7834531 (320)	total: 11.7s	remaining: 24.1s
326:	learn: 0.8004811	test: 0.7831163 best: 0.7834531 (320)	total: 11.7s	remaining: 24s
327:	learn: 0.8003848	test: 0.7827795 best: 0.7834531 (320)	total: 11.7s	remaining: 24s
328:	learn: 0.8004811	test: 0.7826673 best: 0.7834531 (320)	total: 11.7s	remaining: 23.9s
329:	learn: 0.8012026	test: 0.7830602 best: 0.7834531 (320)	total: 11.8s	remaining: 23.9s
330:	learn: 0.8013470	test: 0.7830602 best: 0.7834531 (320)	total: 11.8s	remaining: 23.8s
331:	learn: 0.8011786	test: 0.7833408 best: 0.7834531 (320)	total: 11.8s	remaining: 23.8s
332:	learn: 0.8015153	test: 0.7832847 best: 0.7834531 (320)	total: 11.9s	remaining: 23.8s
333:	learn: 0.8015634	test: 0.7826673 best: 0.7834531 (320)	total: 11.9s	remaining: 23.7s
334:	learn: 0.8016356	test: 0.7826673 best: 0.7834531 (320)	total: 11.9s	remaining: 23.7s
335:	learn: 0.8016115	test: 0.7827795 best: 0.7834531 (320)	total: 12s	remaining: 23.6s
336:	learn: 0.8015875	test: 0.7828918 best: 0.7834531 (320)	total: 12s	remaining: 23.6s
337:	learn: 0.8018521	test: 0.7830040 best: 0.7834531 (320)	total: 12s	remaining: 23.5s
338:	learn: 0.8017318	test: 0.7828357 best: 0.7834531 (320)	total: 12s	remaining: 23.5s
339:	learn: 0.8020686	test: 0.7826673 best: 0.7834531 (320)	total: 12.1s	remaining: 23.4s
340:	learn: 0.8021167	test: 0.7830602 best: 0.7834531 (320)	total: 12.1s	remaining: 23.4s
341:	learn: 0.8021888	test: 0.7827234 best: 0.7834531 (320)	total: 12.1s	remaining: 23.3s
342:	learn: 0.8023572	test: 0.7826673 best: 0.7834531 (320)	total: 12.2s	remaining: 23.3s
343:	learn: 0.8023572	test: 0.7825550 best: 0.7834531 (320)	total: 12.2s	remaining: 23.2s
344:	learn: 0.8024534	test: 0.7826111 best: 0.7834531 (320)	total: 12.2s	remaining: 23.2s
345:	learn: 0.8025496	test: 0.7826111 best: 0.7834531 (320)	total: 12.2s	remaining: 23.1s
346:	learn: 0.8027661	test: 0.7832286 best: 0.7834531 (320)	total: 12.3s	remaining: 23.1s
347:	learn: 0.8026699	test: 0.7828918 best: 0.7834531 (320)	total: 12.3s	remaining: 23.1s
348:	learn: 0.8028382	test: 0.7833969 best: 0.7834531 (320)	total: 12.3s	remaining: 23s
349:	learn: 0.8028382	test: 0.7831163 best: 0.7834531 (320)	total: 12.4s	remaining: 23s
350:	learn: 0.8028623	test: 0.7831724 best: 0.7834531 (320)	total: 12.4s	remaining: 22.9s
351:	learn: 0.8030547	test: 0.7828918 best: 0.7834531 (320)	total: 12.4s	remaining: 22.9s
352:	learn: 0.8030547	test: 0.7828357 best: 0.7834531 (320)	total: 12.5s	remaining: 22.9s
353:	learn: 0.8029345	test: 0.7835092 best: 0.7835092 (353)	total: 12.5s	remaining: 22.8s
354:	learn: 0.8032231	test: 0.7834531 best: 0.7835092 (353)	total: 12.5s	remaining: 22.8s
355:	learn: 0.8035839	test: 0.7837337 best: 0.7837337 (355)	total: 12.6s	remaining: 22.7s
356:	learn: 0.8036320	test: 0.7839582 best: 0.7839582 (356)	total: 12.6s	remaining: 22.7s
357:	learn: 0.8039687	test: 0.7831163 best: 0.7839582 (356)	total: 12.6s	remaining: 22.7s
358:	learn: 0.8040649	test: 0.7832286 best: 0.7839582 (356)	total: 12.7s	remaining: 22.6s
359:	learn: 0.8041371	test: 0.7833969 best: 0.7839582 (356)	total: 12.7s	remaining: 22.6s
360:	learn: 0.8042574	test: 0.7835653 best: 0.7839582 (356)	total: 12.7s	remaining: 22.5s
361:	learn: 0.8041852	test: 0.7836215 best: 0.7839582 (356)	total: 12.8s	remaining: 22.5s
362:	learn: 0.8041130	test: 0.7839021 best: 0.7839582 (356)	total: 12.8s	remaining: 22.4s
363:	learn: 0.8041130	test: 0.7840144 best: 0.7840144 (363)	total: 12.8s	remaining: 22.4s
364:	learn: 0.8041612	test: 0.7840705 best: 0.7840705 (364)	total: 12.9s	remaining: 22.4s
365:	learn: 0.8042093	test: 0.7841266 best: 0.7841266 (365)	total: 12.9s	remaining: 22.3s
366:	learn: 0.8044257	test: 0.7843511 best: 0.7843511 (366)	total: 12.9s	remaining: 22.3s
367:	learn: 0.8044738	test: 0.7846879 best: 0.7846879 (367)	total: 12.9s	remaining: 22.2s
368:	learn: 0.8047144	test: 0.7844634 best: 0.7846879 (367)	total: 13s	remaining: 22.2s
369:	learn: 0.8047865	test: 0.7847441 best: 0.7847441 (369)	total: 13s	remaining: 22.2s
370:	learn: 0.8051233	test: 0.7849686 best: 0.7849686 (370)	total: 13.1s	remaining: 22.2s
371:	learn: 0.8051714	test: 0.7849686 best: 0.7849686 (370)	total: 13.1s	remaining: 22.2s
372:	learn: 0.8051233	test: 0.7849124 best: 0.7849686 (370)	total: 13.2s	remaining: 22.2s
373:	learn: 0.8052435	test: 0.7849124 best: 0.7849686 (370)	total: 13.2s	remaining: 22.1s
374:	learn: 0.8053638	test: 0.7845195 best: 0.7849686 (370)	total: 13.3s	remaining: 22.1s
375:	learn: 0.8053879	test: 0.7839582 best: 0.7849686 (370)	total: 13.3s	remaining: 22.1s
376:	learn: 0.8057005	test: 0.7844634 best: 0.7849686 (370)	total: 13.3s	remaining: 22.1s
377:	learn: 0.8058449	test: 0.7846879 best: 0.7849686 (370)	total: 13.4s	remaining: 22s
378:	learn: 0.8059651	test: 0.7842950 best: 0.7849686 (370)	total: 13.4s	remaining: 22s
379:	learn: 0.8061575	test: 0.7846318 best: 0.7849686 (370)	total: 13.5s	remaining: 22s
380:	learn: 0.8059892	test: 0.7849124 best: 0.7849686 (370)	total: 13.5s	remaining: 21.9s
381:	learn: 0.8062297	test: 0.7847441 best: 0.7849686 (370)	total: 13.5s	remaining: 21.9s
382:	learn: 0.8062538	test: 0.7846318 best: 0.7849686 (370)	total: 13.6s	remaining: 21.8s
383:	learn: 0.8061094	test: 0.7849124 best: 0.7849686 (370)	total: 13.6s	remaining: 21.8s
384:	learn: 0.8061575	test: 0.7847441 best: 0.7849686 (370)	total: 13.6s	remaining: 21.8s
385:	learn: 0.8059892	test: 0.7849686 best: 0.7849686 (370)	total: 13.6s	remaining: 21.7s
386:	learn: 0.8062538	test: 0.7846318 best: 0.7849686 (370)	total: 13.7s	remaining: 21.7s
387:	learn: 0.8065905	test: 0.7847441 best: 0.7849686 (370)	total: 13.7s	remaining: 21.6s
388:	learn: 0.8063500	test: 0.7848563 best: 0.7849686 (370)	total: 13.8s	remaining: 21.6s
389:	learn: 0.8063740	test: 0.7850808 best: 0.7850808 (389)	total: 13.8s	remaining: 21.6s
390:	learn: 0.8067108	test: 0.7849124 best: 0.7850808 (389)	total: 13.9s	remaining: 21.6s
391:	learn: 0.8065905	test: 0.7848002 best: 0.7850808 (389)	total: 13.9s	remaining: 21.6s
392:	learn: 0.8067589	test: 0.7846879 best: 0.7850808 (389)	total: 14s	remaining: 21.6s
393:	learn: 0.8069513	test: 0.7845757 best: 0.7850808 (389)	total: 14s	remaining: 21.6s
394:	learn: 0.8070956	test: 0.7845195 best: 0.7850808 (389)	total: 14.1s	remaining: 21.5s
395:	learn: 0.8076007	test: 0.7848002 best: 0.7850808 (389)	total: 14.1s	remaining: 21.5s
396:	learn: 0.8076007	test: 0.7846879 best: 0.7850808 (389)	total: 14.1s	remaining: 21.5s
397:	learn: 0.8075526	test: 0.7848002 best: 0.7850808 (389)	total: 14.2s	remaining: 21.4s
398:	learn: 0.8075767	test: 0.7846879 best: 0.7850808 (389)	total: 14.2s	remaining: 21.4s
399:	learn: 0.8073361	test: 0.7846879 best: 0.7850808 (389)	total: 14.3s	remaining: 21.4s
400:	learn: 0.8074083	test: 0.7845757 best: 0.7850808 (389)	total: 14.3s	remaining: 21.3s
401:	learn: 0.8075767	test: 0.7844634 best: 0.7850808 (389)	total: 14.3s	remaining: 21.3s
402:	learn: 0.8073121	test: 0.7842389 best: 0.7850808 (389)	total: 14.4s	remaining: 21.3s
403:	learn: 0.8075526	test: 0.7845195 best: 0.7850808 (389)	total: 14.4s	remaining: 21.2s
404:	learn: 0.8077691	test: 0.7845195 best: 0.7850808 (389)	total: 14.4s	remaining: 21.2s
405:	learn: 0.8079615	test: 0.7847441 best: 0.7850808 (389)	total: 14.5s	remaining: 21.2s
406:	learn: 0.8078653	test: 0.7845195 best: 0.7850808 (389)	total: 14.5s	remaining: 21.1s
407:	learn: 0.8079375	test: 0.7846318 best: 0.7850808 (389)	total: 14.5s	remaining: 21.1s
408:	learn: 0.8077691	test: 0.7848563 best: 0.7850808 (389)	total: 14.5s	remaining: 21s
409:	learn: 0.8078172	test: 0.7848002 best: 0.7850808 (389)	total: 14.6s	remaining: 21s
410:	learn: 0.8079134	test: 0.7849686 best: 0.7850808 (389)	total: 14.6s	remaining: 20.9s
411:	learn: 0.8082020	test: 0.7849686 best: 0.7850808 (389)	total: 14.6s	remaining: 20.9s
412:	learn: 0.8083464	test: 0.7850808 best: 0.7850808 (389)	total: 14.7s	remaining: 20.8s
413:	learn: 0.8084666	test: 0.7850247 best: 0.7850808 (389)	total: 14.7s	remaining: 20.8s
414:	learn: 0.8085147	test: 0.7849124 best: 0.7850808 (389)	total: 14.7s	remaining: 20.7s
415:	learn: 0.8086831	test: 0.7849686 best: 0.7850808 (389)	total: 14.8s	remaining: 20.7s
416:	learn: 0.8088034	test: 0.7853053 best: 0.7853053 (416)	total: 14.8s	remaining: 20.7s
417:	learn: 0.8089717	test: 0.7854737 best: 0.7854737 (417)	total: 14.9s	remaining: 20.7s
418:	learn: 0.8091161	test: 0.7855860 best: 0.7855860 (418)	total: 14.9s	remaining: 20.7s
419:	learn: 0.8090198	test: 0.7855299 best: 0.7855860 (418)	total: 15s	remaining: 20.7s
420:	learn: 0.8092123	test: 0.7859228 best: 0.7859228 (420)	total: 15s	remaining: 20.6s
421:	learn: 0.8093325	test: 0.7855299 best: 0.7859228 (420)	total: 15.1s	remaining: 20.6s
422:	learn: 0.8096693	test: 0.7855299 best: 0.7859228 (420)	total: 15.1s	remaining: 20.6s
423:	learn: 0.8096693	test: 0.7851931 best: 0.7859228 (420)	total: 15.1s	remaining: 20.5s
424:	learn: 0.8098857	test: 0.7851931 best: 0.7859228 (420)	total: 15.1s	remaining: 20.5s
425:	learn: 0.8100060	test: 0.7850247 best: 0.7859228 (420)	total: 15.2s	remaining: 20.4s
426:	learn: 0.8101984	test: 0.7850808 best: 0.7859228 (420)	total: 15.2s	remaining: 20.4s
427:	learn: 0.8103668	test: 0.7856421 best: 0.7859228 (420)	total: 15.2s	remaining: 20.4s
428:	learn: 0.8104149	test: 0.7854737 best: 0.7859228 (420)	total: 15.3s	remaining: 20.3s
429:	learn: 0.8106073	test: 0.7854737 best: 0.7859228 (420)	total: 15.3s	remaining: 20.3s
430:	learn: 0.8104871	test: 0.7855299 best: 0.7859228 (420)	total: 15.3s	remaining: 20.2s
431:	learn: 0.8106554	test: 0.7853615 best: 0.7859228 (420)	total: 15.4s	remaining: 20.2s
432:	learn: 0.8106314	test: 0.7858105 best: 0.7859228 (420)	total: 15.4s	remaining: 20.2s
433:	learn: 0.8107757	test: 0.7860350 best: 0.7860350 (433)	total: 15.4s	remaining: 20.1s
434:	learn: 0.8104149	test: 0.7860912 best: 0.7860912 (434)	total: 15.4s	remaining: 20.1s
435:	learn: 0.8104630	test: 0.7858666 best: 0.7860912 (434)	total: 15.5s	remaining: 20s
436:	learn: 0.8108960	test: 0.7859789 best: 0.7860912 (434)	total: 15.5s	remaining: 20s
437:	learn: 0.8107998	test: 0.7864279 best: 0.7864279 (437)	total: 15.6s	remaining: 20s
438:	learn: 0.8108479	test: 0.7865963 best: 0.7865963 (438)	total: 15.6s	remaining: 19.9s
439:	learn: 0.8107276	test: 0.7865963 best: 0.7865963 (438)	total: 15.6s	remaining: 19.9s
440:	learn: 0.8107998	test: 0.7866524 best: 0.7866524 (440)	total: 15.6s	remaining: 19.8s
441:	learn: 0.8108719	test: 0.7865402 best: 0.7866524 (440)	total: 15.7s	remaining: 19.8s
442:	learn: 0.8110162	test: 0.7864279 best: 0.7866524 (440)	total: 15.7s	remaining: 19.7s
443:	learn: 0.8110884	test: 0.7864841 best: 0.7866524 (440)	total: 15.7s	remaining: 19.7s
444:	learn: 0.8113049	test: 0.7863718 best: 0.7866524 (440)	total: 15.8s	remaining: 19.7s
445:	learn: 0.8111606	test: 0.7865402 best: 0.7866524 (440)	total: 15.8s	remaining: 19.6s
446:	learn: 0.8111846	test: 0.7867086 best: 0.7867086 (446)	total: 15.8s	remaining: 19.6s
447:	learn: 0.8112808	test: 0.7868770 best: 0.7868770 (447)	total: 15.9s	remaining: 19.5s
448:	learn: 0.8112808	test: 0.7868770 best: 0.7868770 (447)	total: 15.9s	remaining: 19.5s
449:	learn: 0.8112327	test: 0.7868208 best: 0.7868770 (447)	total: 15.9s	remaining: 19.4s
450:	learn: 0.8110884	test: 0.7867647 best: 0.7868770 (447)	total: 15.9s	remaining: 19.4s
451:	learn: 0.8110643	test: 0.7863718 best: 0.7868770 (447)	total: 16s	remaining: 19.4s
452:	learn: 0.8111606	test: 0.7858666 best: 0.7868770 (447)	total: 16s	remaining: 19.3s
453:	learn: 0.8114732	test: 0.7862595 best: 0.7868770 (447)	total: 16s	remaining: 19.3s
454:	learn: 0.8115935	test: 0.7860912 best: 0.7868770 (447)	total: 16.1s	remaining: 19.2s
455:	learn: 0.8115935	test: 0.7862034 best: 0.7868770 (447)	total: 16.1s	remaining: 19.2s
456:	learn: 0.8116416	test: 0.7862595 best: 0.7868770 (447)	total: 16.1s	remaining: 19.2s
457:	learn: 0.8117859	test: 0.7858666 best: 0.7868770 (447)	total: 16.2s	remaining: 19.1s
458:	learn: 0.8117138	test: 0.7859228 best: 0.7868770 (447)	total: 16.2s	remaining: 19.1s
459:	learn: 0.8117619	test: 0.7859228 best: 0.7868770 (447)	total: 16.2s	remaining: 19.1s
460:	learn: 0.8115695	test: 0.7859228 best: 0.7868770 (447)	total: 16.3s	remaining: 19s
461:	learn: 0.8116897	test: 0.7859789 best: 0.7868770 (447)	total: 16.3s	remaining: 19s
462:	learn: 0.8117859	test: 0.7859789 best: 0.7868770 (447)	total: 16.3s	remaining: 18.9s
463:	learn: 0.8117619	test: 0.7862595 best: 0.7868770 (447)	total: 16.4s	remaining: 18.9s
464:	learn: 0.8118581	test: 0.7861473 best: 0.7868770 (447)	total: 16.4s	remaining: 18.9s
465:	learn: 0.8118581	test: 0.7860350 best: 0.7868770 (447)	total: 16.4s	remaining: 18.8s
```

```
466:	learn: 0.8120265	test: 0.7861473 best: 0.7868770 (447)	total: 16.5s	remaining: 18.8s
467:	learn: 0.8122670	test: 0.7862034 best: 0.7868770 (447)	total: 16.5s	remaining: 18.8s
468:	learn: 0.8123873	test: 0.7860350 best: 0.7868770 (447)	total: 16.5s	remaining: 18.7s
469:	learn: 0.8123632	test: 0.7860350 best: 0.7868770 (447)	total: 16.6s	remaining: 18.7s
470:	learn: 0.8123391	test: 0.7861473 best: 0.7868770 (447)	total: 16.6s	remaining: 18.6s
471:	learn: 0.8123873	test: 0.7861473 best: 0.7868770 (447)	total: 16.6s	remaining: 18.6s
472:	learn: 0.8125797	test: 0.7863157 best: 0.7868770 (447)	total: 16.7s	remaining: 18.6s
473:	learn: 0.8126037	test: 0.7863157 best: 0.7868770 (447)	total: 16.7s	remaining: 18.5s
474:	learn: 0.8126278	test: 0.7863718 best: 0.7868770 (447)	total: 16.7s	remaining: 18.5s
475:	learn: 0.8127721	test: 0.7862034 best: 0.7868770 (447)	total: 16.7s	remaining: 18.4s
476:	learn: 0.8128443	test: 0.7867647 best: 0.7868770 (447)	total: 16.8s	remaining: 18.4s
477:	learn: 0.8127721	test: 0.7865963 best: 0.7868770 (447)	total: 16.8s	remaining: 18.4s
478:	learn: 0.8129164	test: 0.7871015 best: 0.7871015 (478)	total: 16.9s	remaining: 18.3s
479:	learn: 0.8130126	test: 0.7865963 best: 0.7871015 (478)	total: 16.9s	remaining: 18.3s
480:	learn: 0.8131329	test: 0.7867086 best: 0.7871015 (478)	total: 16.9s	remaining: 18.3s
481:	learn: 0.8132051	test: 0.7867086 best: 0.7871015 (478)	total: 17s	remaining: 18.3s
482:	learn: 0.8130848	test: 0.7865402 best: 0.7871015 (478)	total: 17s	remaining: 18.2s
483:	learn: 0.8132051	test: 0.7868208 best: 0.7871015 (478)	total: 17.1s	remaining: 18.2s
484:	learn: 0.8131088	test: 0.7868208 best: 0.7871015 (478)	total: 17.1s	remaining: 18.2s
485:	learn: 0.8130848	test: 0.7871576 best: 0.7871576 (485)	total: 17.1s	remaining: 18.1s
486:	learn: 0.8131810	test: 0.7876628 best: 0.7876628 (486)	total: 17.2s	remaining: 18.1s
487:	learn: 0.8132772	test: 0.7874383 best: 0.7876628 (486)	total: 17.2s	remaining: 18.1s
488:	learn: 0.8133734	test: 0.7872699 best: 0.7876628 (486)	total: 17.3s	remaining: 18.1s
489:	learn: 0.8132291	test: 0.7873821 best: 0.7876628 (486)	total: 17.3s	remaining: 18s
490:	learn: 0.8132291	test: 0.7873821 best: 0.7876628 (486)	total: 17.4s	remaining: 18s
491:	learn: 0.8131329	test: 0.7876628 best: 0.7876628 (486)	total: 17.4s	remaining: 18s
492:	learn: 0.8134215	test: 0.7873821 best: 0.7876628 (486)	total: 17.4s	remaining: 17.9s
493:	learn: 0.8132532	test: 0.7873821 best: 0.7876628 (486)	total: 17.4s	remaining: 17.9s
494:	learn: 0.8131329	test: 0.7874944 best: 0.7876628 (486)	total: 17.5s	remaining: 17.8s
495:	learn: 0.8133734	test: 0.7872137 best: 0.7876628 (486)	total: 17.5s	remaining: 17.8s
496:	learn: 0.8133975	test: 0.7872699 best: 0.7876628 (486)	total: 17.6s	remaining: 17.8s
497:	learn: 0.8134696	test: 0.7876628 best: 0.7876628 (486)	total: 17.6s	remaining: 17.8s
498:	learn: 0.8135177	test: 0.7876066 best: 0.7876628 (486)	total: 17.7s	remaining: 17.7s
499:	learn: 0.8135418	test: 0.7874944 best: 0.7876628 (486)	total: 17.7s	remaining: 17.7s
500:	learn: 0.8137823	test: 0.7876066 best: 0.7876628 (486)	total: 17.8s	remaining: 17.7s
501:	learn: 0.8137102	test: 0.7876628 best: 0.7876628 (486)	total: 17.8s	remaining: 17.7s
502:	learn: 0.8138064	test: 0.7875505 best: 0.7876628 (486)	total: 17.8s	remaining: 17.6s
503:	learn: 0.8138064	test: 0.7875505 best: 0.7876628 (486)	total: 17.9s	remaining: 17.6s
504:	learn: 0.8139026	test: 0.7875505 best: 0.7876628 (486)	total: 17.9s	remaining: 17.6s
505:	learn: 0.8137823	test: 0.7874944 best: 0.7876628 (486)	total: 18s	remaining: 17.5s
506:	learn: 0.8135899	test: 0.7876628 best: 0.7876628 (486)	total: 18s	remaining: 17.5s
507:	learn: 0.8136621	test: 0.7876066 best: 0.7876628 (486)	total: 18s	remaining: 17.5s
508:	learn: 0.8136621	test: 0.7874944 best: 0.7876628 (486)	total: 18s	remaining: 17.4s
509:	learn: 0.8139026	test: 0.7879434 best: 0.7879434 (509)	total: 18.1s	remaining: 17.4s
510:	learn: 0.8139747	test: 0.7878873 best: 0.7879434 (509)	total: 18.1s	remaining: 17.3s
511:	learn: 0.8140950	test: 0.7877750 best: 0.7879434 (509)	total: 18.1s	remaining: 17.3s
512:	learn: 0.8141672	test: 0.7876628 best: 0.7879434 (509)	total: 18.2s	remaining: 17.2s
513:	learn: 0.8144317	test: 0.7876066 best: 0.7879434 (509)	total: 18.2s	remaining: 17.2s
514:	learn: 0.8145039	test: 0.7874944 best: 0.7879434 (509)	total: 18.2s	remaining: 17.2s
515:	learn: 0.8144799	test: 0.7874383 best: 0.7879434 (509)	total: 18.3s	remaining: 17.1s
516:	learn: 0.8146723	test: 0.7874383 best: 0.7879434 (509)	total: 18.3s	remaining: 17.1s
517:	learn: 0.8146482	test: 0.7876628 best: 0.7879434 (509)	total: 18.4s	remaining: 17.1s
518:	learn: 0.8146723	test: 0.7877189 best: 0.7879434 (509)	total: 18.4s	remaining: 17.1s
519:	learn: 0.8148406	test: 0.7878312 best: 0.7879434 (509)	total: 18.5s	remaining: 17.1s
520:	learn: 0.8148888	test: 0.7878873 best: 0.7879434 (509)	total: 18.5s	remaining: 17s
521:	learn: 0.8152736	test: 0.7882802 best: 0.7882802 (521)	total: 18.6s	remaining: 17s
522:	learn: 0.8152014	test: 0.7877189 best: 0.7882802 (521)	total: 18.6s	remaining: 17s
523:	learn: 0.8152977	test: 0.7876628 best: 0.7882802 (521)	total: 18.7s	remaining: 16.9s
524:	learn: 0.8153217	test: 0.7877189 best: 0.7882802 (521)	total: 18.7s	remaining: 16.9s
525:	learn: 0.8155141	test: 0.7879996 best: 0.7882802 (521)	total: 18.7s	remaining: 16.9s
526:	learn: 0.8157306	test: 0.7879996 best: 0.7882802 (521)	total: 18.8s	remaining: 16.8s
527:	learn: 0.8157547	test: 0.7881118 best: 0.7882802 (521)	total: 18.8s	remaining: 16.8s
528:	learn: 0.8157306	test: 0.7879996 best: 0.7882802 (521)	total: 18.9s	remaining: 16.8s
529:	learn: 0.8157306	test: 0.7879996 best: 0.7882802 (521)	total: 18.9s	remaining: 16.8s
530:	learn: 0.8159471	test: 0.7882802 best: 0.7882802 (521)	total: 18.9s	remaining: 16.7s
531:	learn: 0.8159952	test: 0.7886731 best: 0.7886731 (531)	total: 19s	remaining: 16.7s
532:	learn: 0.8158749	test: 0.7883925 best: 0.7886731 (531)	total: 19s	remaining: 16.7s
533:	learn: 0.8158990	test: 0.7885047 best: 0.7886731 (531)	total: 19.1s	remaining: 16.6s
534:	learn: 0.8160433	test: 0.7885047 best: 0.7886731 (531)	total: 19.1s	remaining: 16.6s
535:	learn: 0.8160192	test: 0.7882241 best: 0.7886731 (531)	total: 19.1s	remaining: 16.6s
536:	learn: 0.8158268	test: 0.7882241 best: 0.7886731 (531)	total: 19.2s	remaining: 16.5s
537:	learn: 0.8160673	test: 0.7881679 best: 0.7886731 (531)	total: 19.2s	remaining: 16.5s
538:	learn: 0.8160673	test: 0.7884486 best: 0.7886731 (531)	total: 19.2s	remaining: 16.4s
539:	learn: 0.8161155	test: 0.7884486 best: 0.7886731 (531)	total: 19.3s	remaining: 16.4s
540:	learn: 0.8161395	test: 0.7882802 best: 0.7886731 (531)	total: 19.3s	remaining: 16.4s
541:	learn: 0.8166687	test: 0.7883363 best: 0.7886731 (531)	total: 19.3s	remaining: 16.3s
542:	learn: 0.8166446	test: 0.7880557 best: 0.7886731 (531)	total: 19.3s	remaining: 16.3s
543:	learn: 0.8167408	test: 0.7880557 best: 0.7886731 (531)	total: 19.4s	remaining: 16.2s
544:	learn: 0.8167649	test: 0.7884486 best: 0.7886731 (531)	total: 19.4s	remaining: 16.2s
545:	learn: 0.8166927	test: 0.7884486 best: 0.7886731 (531)	total: 19.4s	remaining: 16.2s
546:	learn: 0.8170054	test: 0.7885047 best: 0.7886731 (531)	total: 19.5s	remaining: 16.1s
547:	learn: 0.8169814	test: 0.7882802 best: 0.7886731 (531)	total: 19.5s	remaining: 16.1s
548:	learn: 0.8171738	test: 0.7886731 best: 0.7886731 (531)	total: 19.5s	remaining: 16s
549:	learn: 0.8172459	test: 0.7885608 best: 0.7886731 (531)	total: 19.6s	remaining: 16s
550:	learn: 0.8172219	test: 0.7885608 best: 0.7886731 (531)	total: 19.6s	remaining: 16s
551:	learn: 0.8170535	test: 0.7883363 best: 0.7886731 (531)	total: 19.6s	remaining: 15.9s
552:	learn: 0.8171016	test: 0.7882802 best: 0.7886731 (531)	total: 19.7s	remaining: 15.9s
553:	learn: 0.8171738	test: 0.7882241 best: 0.7886731 (531)	total: 19.7s	remaining: 15.9s
554:	learn: 0.8170295	test: 0.7883363 best: 0.7886731 (531)	total: 19.7s	remaining: 15.8s
555:	learn: 0.8170776	test: 0.7881118 best: 0.7886731 (531)	total: 19.8s	remaining: 15.8s
556:	learn: 0.8171257	test: 0.7881679 best: 0.7886731 (531)	total: 19.8s	remaining: 15.7s
557:	learn: 0.8172459	test: 0.7883363 best: 0.7886731 (531)	total: 19.8s	remaining: 15.7s
558:	learn: 0.8172700	test: 0.7885608 best: 0.7886731 (531)	total: 19.9s	remaining: 15.7s
559:	learn: 0.8173422	test: 0.7887292 best: 0.7887292 (559)	total: 19.9s	remaining: 15.6s
560:	learn: 0.8172219	test: 0.7886170 best: 0.7887292 (559)	total: 19.9s	remaining: 15.6s
561:	learn: 0.8175827	test: 0.7884486 best: 0.7887292 (559)	total: 19.9s	remaining: 15.5s
562:	learn: 0.8177751	test: 0.7883925 best: 0.7887292 (559)	total: 20s	remaining: 15.5s
563:	learn: 0.8177029	test: 0.7886170 best: 0.7887292 (559)	total: 20s	remaining: 15.5s
564:	learn: 0.8176548	test: 0.7886731 best: 0.7887292 (559)	total: 20s	remaining: 15.4s
565:	learn: 0.8176308	test: 0.7884486 best: 0.7887292 (559)	total: 20.1s	remaining: 15.4s
566:	learn: 0.8176548	test: 0.7885608 best: 0.7887292 (559)	total: 20.1s	remaining: 15.3s
567:	learn: 0.8177511	test: 0.7883925 best: 0.7887292 (559)	total: 20.1s	remaining: 15.3s
568:	learn: 0.8178473	test: 0.7887292 best: 0.7887292 (559)	total: 20.1s	remaining: 15.3s
569:	learn: 0.8177029	test: 0.7886731 best: 0.7887292 (559)	total: 20.2s	remaining: 15.2s
570:	learn: 0.8177751	test: 0.7885047 best: 0.7887292 (559)	total: 20.2s	remaining: 15.2s
571:	learn: 0.8177992	test: 0.7886170 best: 0.7887292 (559)	total: 20.2s	remaining: 15.1s
572:	learn: 0.8178232	test: 0.7886731 best: 0.7887292 (559)	total: 20.3s	remaining: 15.1s
573:	learn: 0.8180637	test: 0.7883925 best: 0.7887292 (559)	total: 20.3s	remaining: 15.1s
574:	learn: 0.8181118	test: 0.7883363 best: 0.7887292 (559)	total: 20.3s	remaining: 15s
575:	learn: 0.8181118	test: 0.7885608 best: 0.7887292 (559)	total: 20.4s	remaining: 15s
576:	learn: 0.8181840	test: 0.7881679 best: 0.7887292 (559)	total: 20.4s	remaining: 14.9s
577:	learn: 0.8182321	test: 0.7884486 best: 0.7887292 (559)	total: 20.4s	remaining: 14.9s
578:	learn: 0.8182562	test: 0.7882802 best: 0.7887292 (559)	total: 20.5s	remaining: 14.9s
579:	learn: 0.8185207	test: 0.7883925 best: 0.7887292 (559)	total: 20.5s	remaining: 14.8s
580:	learn: 0.8186651	test: 0.7882802 best: 0.7887292 (559)	total: 20.6s	remaining: 14.8s
581:	learn: 0.8186170	test: 0.7884486 best: 0.7887292 (559)	total: 20.6s	remaining: 14.8s
582:	learn: 0.8186170	test: 0.7884486 best: 0.7887292 (559)	total: 20.7s	remaining: 14.8s
583:	learn: 0.8188815	test: 0.7881679 best: 0.7887292 (559)	total: 20.7s	remaining: 14.7s
584:	learn: 0.8187853	test: 0.7881679 best: 0.7887292 (559)	total: 20.8s	remaining: 14.7s
585:	learn: 0.8190018	test: 0.7880557 best: 0.7887292 (559)	total: 20.8s	remaining: 14.7s
586:	learn: 0.8189537	test: 0.7883363 best: 0.7887292 (559)	total: 20.9s	remaining: 14.7s
587:	learn: 0.8190018	test: 0.7885047 best: 0.7887292 (559)	total: 20.9s	remaining: 14.6s
588:	learn: 0.8190018	test: 0.7885047 best: 0.7887292 (559)	total: 20.9s	remaining: 14.6s
589:	learn: 0.8190018	test: 0.7886170 best: 0.7887292 (559)	total: 21s	remaining: 14.6s
590:	learn: 0.8190018	test: 0.7886170 best: 0.7887292 (559)	total: 21s	remaining: 14.5s
591:	learn: 0.8191221	test: 0.7886170 best: 0.7887292 (559)	total: 21.1s	remaining: 14.5s
592:	learn: 0.8191942	test: 0.7886731 best: 0.7887292 (559)	total: 21.1s	remaining: 14.5s
593:	learn: 0.8191461	test: 0.7888976 best: 0.7888976 (593)	total: 21.1s	remaining: 14.4s
594:	learn: 0.8191942	test: 0.7887854 best: 0.7888976 (593)	total: 21.2s	remaining: 14.4s
595:	learn: 0.8193145	test: 0.7887292 best: 0.7888976 (593)	total: 21.2s	remaining: 14.4s
596:	learn: 0.8192904	test: 0.7883925 best: 0.7888976 (593)	total: 21.2s	remaining: 14.3s
597:	learn: 0.8191942	test: 0.7885608 best: 0.7888976 (593)	total: 21.3s	remaining: 14.3s
598:	learn: 0.8194348	test: 0.7885047 best: 0.7888976 (593)	total: 21.3s	remaining: 14.3s
599:	learn: 0.8193145	test: 0.7886731 best: 0.7888976 (593)	total: 21.3s	remaining: 14.2s
600:	learn: 0.8192664	test: 0.7886731 best: 0.7888976 (593)	total: 21.4s	remaining: 14.2s
601:	learn: 0.8195550	test: 0.7887854 best: 0.7888976 (593)	total: 21.4s	remaining: 14.2s
602:	learn: 0.8196753	test: 0.7885608 best: 0.7888976 (593)	total: 21.4s	remaining: 14.1s
603:	learn: 0.8196753	test: 0.7885047 best: 0.7888976 (593)	total: 21.5s	remaining: 14.1s
604:	learn: 0.8197234	test: 0.7886731 best: 0.7888976 (593)	total: 21.5s	remaining: 14s
605:	learn: 0.8197715	test: 0.7887292 best: 0.7888976 (593)	total: 21.5s	remaining: 14s
606:	learn: 0.8198918	test: 0.7886170 best: 0.7888976 (593)	total: 21.6s	remaining: 14s
607:	learn: 0.8199158	test: 0.7885608 best: 0.7888976 (593)	total: 21.6s	remaining: 13.9s
608:	learn: 0.8202044	test: 0.7887854 best: 0.7888976 (593)	total: 21.6s	remaining: 13.9s
609:	learn: 0.8201323	test: 0.7886170 best: 0.7888976 (593)	total: 21.6s	remaining: 13.8s
```

```
610:	learn: 0.8201323	test: 0.7887854	best: 0.7888976 (593)	total: 21.7s	remaining: 13.8s
611:	learn: 0.8202285	test: 0.7888415	best: 0.7888976 (593)	total: 21.7s	remaining: 13.8s
612:	learn: 0.8199158	test: 0.7888415	best: 0.7888976 (593)	total: 21.7s	remaining: 13.7s
613:	learn: 0.8201082	test: 0.7889537	best: 0.7889537 (613)	total: 21.8s	remaining: 13.7s
614:	learn: 0.8201323	test: 0.7891221	best: 0.7891221 (614)	total: 21.8s	remaining: 13.6s
615:	learn: 0.8203007	test: 0.7889537	best: 0.7891221 (614)	total: 21.8s	remaining: 13.6s
616:	learn: 0.8202285	test: 0.7889537	best: 0.7891221 (614)	total: 21.8s	remaining: 13.6s
617:	learn: 0.8203247	test: 0.7888976	best: 0.7891221 (614)	total: 21.9s	remaining: 13.5s
618:	learn: 0.8204931	test: 0.7892905	best: 0.7892905 (618)	total: 21.9s	remaining: 13.5s
619:	learn: 0.8202285	test: 0.7892905	best: 0.7892905 (618)	total: 21.9s	remaining: 13.4s
620:	learn: 0.8202285	test: 0.7893467	best: 0.7893467 (620)	total: 22s	remaining: 13.4s
621:	learn: 0.8203728	test: 0.7894028	best: 0.7894028 (621)	total: 22s	remaining: 13.4s
622:	learn: 0.8205412	test: 0.7892905	best: 0.7894028 (621)	total: 22s	remaining: 13.3s
623:	learn: 0.8207817	test: 0.7892905	best: 0.7894028 (621)	total: 22s	remaining: 13.3s
624:	learn: 0.8208539	test: 0.7891783	best: 0.7894028 (621)	total: 22.1s	remaining: 13.2s
625:	learn: 0.8209020	test: 0.7892905	best: 0.7894028 (621)	total: 22.1s	remaining: 13.2s
626:	learn: 0.8210463	test: 0.7891221	best: 0.7894028 (621)	total: 22.1s	remaining: 13.2s
627:	learn: 0.8210944	test: 0.7891783	best: 0.7894028 (621)	total: 22.2s	remaining: 13.1s
628:	learn: 0.8213590	test: 0.7890660	best: 0.7894028 (621)	total: 22.2s	remaining: 13.1s
629:	learn: 0.8215514	test: 0.7891221	best: 0.7894028 (621)	total: 22.2s	remaining: 13.1s
630:	learn: 0.8213830	test: 0.7895712	best: 0.7895712 (630)	total: 22.3s	remaining: 13s
631:	learn: 0.8215514	test: 0.7895150	best: 0.7895712 (630)	total: 22.3s	remaining: 13s
632:	learn: 0.8214793	test: 0.7895712	best: 0.7895712 (630)	total: 22.3s	remaining: 12.9s
633:	learn: 0.8215514	test: 0.7894028	best: 0.7895712 (630)	total: 22.4s	remaining: 12.9s
634:	learn: 0.8215274	test: 0.7894589	best: 0.7895712 (630)	total: 22.4s	remaining: 12.9s
635:	learn: 0.8217438	test: 0.7897957	best: 0.7897957 (635)	total: 22.4s	remaining: 12.8s
636:	learn: 0.8217438	test: 0.7896834	best: 0.7897957 (635)	total: 22.4s	remaining: 12.8s
637:	learn: 0.8218641	test: 0.7900202	best: 0.7900202 (637)	total: 22.5s	remaining: 12.8s
638:	learn: 0.8214552	test: 0.7900202	best: 0.7900202 (637)	total: 22.5s	remaining: 12.7s
639:	learn: 0.8215995	test: 0.7898518	best: 0.7900202 (637)	total: 22.6s	remaining: 12.7s
640:	learn: 0.8218641	test: 0.7903009	best: 0.7903009 (640)	total: 22.6s	remaining: 12.7s
641:	learn: 0.8217198	test: 0.7903570	best: 0.7903570 (641)	total: 22.6s	remaining: 12.6s
642:	learn: 0.8217438	test: 0.7901886	best: 0.7903570 (641)	total: 22.7s	remaining: 12.6s
643:	learn: 0.8217438	test: 0.7900202	best: 0.7903570 (641)	total: 22.7s	remaining: 12.6s
644:	learn: 0.8218160	test: 0.7903009	best: 0.7903570 (641)	total: 22.8s	remaining: 12.5s
645:	learn: 0.8220806	test: 0.7899641	best: 0.7903570 (641)	total: 22.8s	remaining: 12.5s
646:	learn: 0.8221287	test: 0.7897957	best: 0.7903570 (641)	total: 22.9s	remaining: 12.5s
647:	learn: 0.8221527	test: 0.7899079	best: 0.7903570 (641)	total: 22.9s	remaining: 12.4s
648:	learn: 0.8221046	test: 0.7900202	best: 0.7903570 (641)	total: 22.9s	remaining: 12.4s
649:	learn: 0.8220325	test: 0.7898518	best: 0.7903570 (641)	total: 23s	remaining: 12.4s
650:	learn: 0.8223452	test: 0.7900763	best: 0.7903570 (641)	total: 23s	remaining: 12.3s
651:	learn: 0.8222730	test: 0.7901886	best: 0.7903570 (641)	total: 23.1s	remaining: 12.3s
652:	learn: 0.8222971	test: 0.7902447	best: 0.7903570 (641)	total: 23.1s	remaining: 12.3s
653:	learn: 0.8222730	test: 0.7900202	best: 0.7903570 (641)	total: 23.2s	remaining: 12.3s
654:	learn: 0.8224895	test: 0.7900202	best: 0.7903570 (641)	total: 23.2s	remaining: 12.2s
655:	learn: 0.8224654	test: 0.7900763	best: 0.7903570 (641)	total: 23.3s	remaining: 12.2s
656:	learn: 0.8224895	test: 0.7901886	best: 0.7903570 (641)	total: 23.3s	remaining: 12.2s
657:	learn: 0.8225857	test: 0.7901325	best: 0.7903570 (641)	total: 23.4s	remaining: 12.1s
658:	learn: 0.8226338	test: 0.7899079	best: 0.7903570 (641)	total: 23.4s	remaining: 12.1s
659:	learn: 0.8228743	test: 0.7897957	best: 0.7903570 (641)	total: 23.4s	remaining: 12.1s
660:	learn: 0.8229224	test: 0.7896273	best: 0.7903570 (641)	total: 23.5s	remaining: 12s
661:	learn: 0.8231149	test: 0.7896834	best: 0.7903570 (641)	total: 23.5s	remaining: 12s
662:	learn: 0.8232351	test: 0.7899641	best: 0.7903570 (641)	total: 23.6s	remaining: 12s
663:	learn: 0.8232592	test: 0.7899641	best: 0.7903570 (641)	total: 23.6s	remaining: 12s
664:	learn: 0.8232832	test: 0.7902447	best: 0.7903570 (641)	total: 23.7s	remaining: 11.9s
665:	learn: 0.8233313	test: 0.7903570	best: 0.7903570 (641)	total: 23.7s	remaining: 11.9s
666:	learn: 0.8233313	test: 0.7901886	best: 0.7903570 (641)	total: 23.8s	remaining: 11.9s
667:	learn: 0.8232351	test: 0.7901886	best: 0.7903570 (641)	total: 23.8s	remaining: 11.8s
668:	learn: 0.8231630	test: 0.7901325	best: 0.7903570 (641)	total: 23.9s	remaining: 11.8s
669:	learn: 0.8232832	test: 0.7902447	best: 0.7903570 (641)	total: 23.9s	remaining: 11.8s
670:	learn: 0.8232832	test: 0.7904692	best: 0.7904692 (670)	total: 23.9s	remaining: 11.7s
671:	learn: 0.8233794	test: 0.7905254	best: 0.7905254 (671)	total: 24s	remaining: 11.7s
672:	learn: 0.8234997	test: 0.7905254	best: 0.7905254 (671)	total: 24s	remaining: 11.7s
673:	learn: 0.8235238	test: 0.7905254	best: 0.7905254 (671)	total: 24.1s	remaining: 11.7s
674:	learn: 0.8234756	test: 0.7904692	best: 0.7905254 (671)	total: 24.1s	remaining: 11.6s
675:	learn: 0.8235719	test: 0.7905254	best: 0.7905254 (671)	total: 24.2s	remaining: 11.6s
676:	learn: 0.8236200	test: 0.7898518	best: 0.7905254 (671)	total: 24.2s	remaining: 11.6s
677:	learn: 0.8237643	test: 0.7900763	best: 0.7905254 (671)	total: 24.3s	remaining: 11.5s
678:	learn: 0.8235959	test: 0.7901325	best: 0.7905254 (671)	total: 24.3s	remaining: 11.5s
679:	learn: 0.8241251	test: 0.7905254	best: 0.7905254 (671)	total: 24.4s	remaining: 11.5s
680:	learn: 0.8241732	test: 0.7906376	best: 0.7906376 (680)	total: 24.4s	remaining: 11.4s
681:	learn: 0.8241491	test: 0.7905815	best: 0.7906376 (680)	total: 24.4s	remaining: 11.4s
682:	learn: 0.8241251	test: 0.7910867	best: 0.7910867 (682)	total: 24.5s	remaining: 11.4s
683:	learn: 0.8241251	test: 0.7908060	best: 0.7910867 (682)	total: 24.5s	remaining: 11.3s
684:	learn: 0.8242934	test: 0.7905815	best: 0.7910867 (682)	total: 24.6s	remaining: 11.3s
685:	learn: 0.8242934	test: 0.7907499	best: 0.7910867 (682)	total: 24.6s	remaining: 11.3s
686:	learn: 0.8243175	test: 0.7906938	best: 0.7910867 (682)	total: 24.6s	remaining: 11.2s
687:	learn: 0.8247023	test: 0.7904131	best: 0.7910867 (682)	total: 24.7s	remaining: 11.2s
688:	learn: 0.8246783	test: 0.7901325	best: 0.7910867 (682)	total: 24.7s	remaining: 11.2s
689:	learn: 0.8245821	test: 0.7901886	best: 0.7910867 (682)	total: 24.8s	remaining: 11.1s
690:	learn: 0.8245340	test: 0.7901325	best: 0.7910867 (682)	total: 24.8s	remaining: 11.1s
691:	learn: 0.8247505	test: 0.7904692	best: 0.7910867 (682)	total: 24.9s	remaining: 11.1s
692:	learn: 0.8247745	test: 0.7905254	best: 0.7910867 (682)	total: 24.9s	remaining: 11s
693:	learn: 0.8247986	test: 0.7905254	best: 0.7910867 (682)	total: 24.9s	remaining: 11s
694:	learn: 0.8247264	test: 0.7904692	best: 0.7910867 (682)	total: 25s	remaining: 10.9s
695:	learn: 0.8247505	test: 0.7905254	best: 0.7910867 (682)	total: 25s	remaining: 10.9s
696:	learn: 0.8247264	test: 0.7906376	best: 0.7910867 (682)	total: 25s	remaining: 10.9s
697:	learn: 0.8247745	test: 0.7906376	best: 0.7910867 (682)	total: 25.1s	remaining: 10.8s
698:	learn: 0.8246783	test: 0.7904692	best: 0.7910867 (682)	total: 25.1s	remaining: 10.8s
699:	learn: 0.8247023	test: 0.7906376	best: 0.7910867 (682)	total: 25.1s	remaining: 10.8s
700:	learn: 0.8244618	test: 0.7903570	best: 0.7910867 (682)	total: 25.2s	remaining: 10.7s
701:	learn: 0.8244137	test: 0.7908060	best: 0.7910867 (682)	total: 25.2s	remaining: 10.7s
702:	learn: 0.8246061	test: 0.7904131	best: 0.7910867 (682)	total: 25.2s	remaining: 10.7s
703:	learn: 0.8247023	test: 0.7904692	best: 0.7910867 (682)	total: 25.3s	remaining: 10.6s
704:	learn: 0.8246783	test: 0.7903570	best: 0.7910867 (682)	total: 25.3s	remaining: 10.6s
705:	learn: 0.8251834	test: 0.7907499	best: 0.7910867 (682)	total: 25.3s	remaining: 10.5s
706:	learn: 0.8250150	test: 0.7906938	best: 0.7910867 (682)	total: 25.3s	remaining: 10.5s
707:	learn: 0.8248707	test: 0.7904692	best: 0.7910867 (682)	total: 25.4s	remaining: 10.5s
708:	learn: 0.8249910	test: 0.7904692	best: 0.7910867 (682)	total: 25.4s	remaining: 10.4s
709:	learn: 0.8248707	test: 0.7906938	best: 0.7910867 (682)	total: 25.4s	remaining: 10.4s
710:	learn: 0.8251112	test: 0.7901886	best: 0.7910867 (682)	total: 25.5s	remaining: 10.4s
711:	learn: 0.8249910	test: 0.7901886	best: 0.7910867 (682)	total: 25.5s	remaining: 10.3s
712:	learn: 0.8251112	test: 0.7903009	best: 0.7910867 (682)	total: 25.5s	remaining: 10.3s
713:	learn: 0.8252556	test: 0.7909744	best: 0.7910867 (682)	total: 25.6s	remaining: 10.2s
714:	learn: 0.8253277	test: 0.7911428	best: 0.7911428 (714)	total: 25.6s	remaining: 10.2s
715:	learn: 0.8255923	test: 0.7911428	best: 0.7911428 (714)	total: 25.6s	remaining: 10.2s
716:	learn: 0.8258088	test: 0.7908060	best: 0.7911428 (714)	total: 25.7s	remaining: 10.1s
717:	learn: 0.8258569	test: 0.7909183	best: 0.7911428 (714)	total: 25.7s	remaining: 10.1s
718:	learn: 0.8257607	test: 0.7906938	best: 0.7911428 (714)	total: 25.7s	remaining: 10.1s
719:	learn: 0.8259050	test: 0.7903570	best: 0.7911428 (714)	total: 25.8s	remaining: 10s
720:	learn: 0.8259771	test: 0.7906376	best: 0.7911428 (714)	total: 25.8s	remaining: 9.98s
721:	learn: 0.8257847	test: 0.7908621	best: 0.7911428 (714)	total: 25.8s	remaining: 9.94s
722:	learn: 0.8258328	test: 0.7906938	best: 0.7911428 (714)	total: 25.9s	remaining: 9.9s
723:	learn: 0.8260012	test: 0.7907499	best: 0.7911428 (714)	total: 25.9s	remaining: 9.87s
724:	learn: 0.8261936	test: 0.7908621	best: 0.7911428 (714)	total: 25.9s	remaining: 9.83s
725:	learn: 0.8261455	test: 0.7906376	best: 0.7911428 (714)	total: 25.9s	remaining: 9.79s
726:	learn: 0.8260974	test: 0.7904131	best: 0.7911428 (714)	total: 26s	remaining: 9.76s
727:	learn: 0.8261455	test: 0.7904692	best: 0.7911428 (714)	total: 26s	remaining: 9.72s
728:	learn: 0.8260734	test: 0.7905815	best: 0.7911428 (714)	total: 26s	remaining: 9.68s
729:	learn: 0.8263379	test: 0.7909744	best: 0.7911428 (714)	total: 26.1s	remaining: 9.64s
730:	learn: 0.8262898	test: 0.7908060	best: 0.7911428 (714)	total: 26.1s	remaining: 9.6s
731:	learn: 0.8263379	test: 0.7908621	best: 0.7911428 (714)	total: 26.1s	remaining: 9.56s
732:	learn: 0.8265785	test: 0.7909183	best: 0.7911428 (714)	total: 26.2s	remaining: 9.53s
733:	learn: 0.8266025	test: 0.7910305	best: 0.7911428 (714)	total: 26.2s	remaining: 9.49s
734:	learn: 0.8266987	test: 0.7909744	best: 0.7911428 (714)	total: 26.2s	remaining: 9.45s
735:	learn: 0.8266266	test: 0.7909183	best: 0.7911428 (714)	total: 26.2s	remaining: 9.41s
736:	learn: 0.8266025	test: 0.7909183	best: 0.7911428 (714)	total: 26.3s	remaining: 9.37s
737:	learn: 0.8266266	test: 0.7911428	best: 0.7911428 (714)	total: 26.3s	remaining: 9.34s
738:	learn: 0.8265785	test: 0.7913112	best: 0.7913112 (738)	total: 26.3s	remaining: 9.3s
739:	learn: 0.8266747	test: 0.7911428	best: 0.7913112 (738)	total: 26.4s	remaining: 9.27s
740:	learn: 0.8268671	test: 0.7913112	best: 0.7913112 (738)	total: 26.4s	remaining: 9.23s
741:	learn: 0.8268190	test: 0.7913112	best: 0.7913112 (738)	total: 26.5s	remaining: 9.2s
742:	learn: 0.8271557	test: 0.7916480	best: 0.7916480 (742)	total: 26.5s	remaining: 9.17s
743:	learn: 0.8273001	test: 0.7917041	best: 0.7917041 (743)	total: 26.6s	remaining: 9.14s
744:	learn: 0.8273482	test: 0.7915918	best: 0.7917041 (743)	total: 26.6s	remaining: 9.1s
745:	learn: 0.8274925	test: 0.7915357	best: 0.7917041 (743)	total: 26.6s	remaining: 9.07s
746:	learn: 0.8276127	test: 0.7917041	best: 0.7917041 (743)	total: 26.7s	remaining: 9.03s
747:	learn: 0.8273963	test: 0.7915357	best: 0.7917041 (743)	total: 26.7s	remaining: 9s
748:	learn: 0.8275165	test: 0.7914796	best: 0.7917041 (743)	total: 26.7s	remaining: 8.96s
749:	learn: 0.8274444	test: 0.7913673	best: 0.7917041 (743)	total: 26.8s	remaining: 8.93s
750:	learn: 0.8275406	test: 0.7912551	best: 0.7917041 (743)	total: 26.8s	remaining: 8.89s
751:	learn: 0.8276609	test: 0.7911428	best: 0.7917041 (743)	total: 26.9s	remaining: 8.86s
752:	learn: 0.8274684	test: 0.7915918	best: 0.7917041 (743)	total: 26.9s	remaining: 8.82s
753:	learn: 0.8273963	test: 0.7915918	best: 0.7917041 (743)	total: 26.9s	remaining: 8.79s
```

```
754:	learn: 0.8275887	test: 0.7916480 best: 0.7917041 (743)	total: 27s	remaining: 8.75s
755:	learn: 0.8276368	test: 0.7915357 best: 0.7917041 (743)	total: 27s	remaining: 8.72s
756:	learn: 0.8276849	test: 0.7914796 best: 0.7917041 (743)	total: 27s	remaining: 8.68s
757:	learn: 0.8276849	test: 0.7918163 best: 0.7918163 (757)	total: 27.1s	remaining: 8.64s
758:	learn: 0.8276609	test: 0.7919847 best: 0.7919847 (758)	total: 27.1s	remaining: 8.61s
759:	learn: 0.8277571	test: 0.7917041 best: 0.7919847 (758)	total: 27.1s	remaining: 8.57s
760:	learn: 0.8277811	test: 0.7920409 best: 0.7920409 (760)	total: 27.2s	remaining: 8.53s
761:	learn: 0.8276368	test: 0.7922654 best: 0.7922654 (761)	total: 27.2s	remaining: 8.49s
762:	learn: 0.8277090	test: 0.7923776 best: 0.7923776 (762)	total: 27.2s	remaining: 8.46s
763:	learn: 0.8277811	test: 0.7923776 best: 0.7923776 (762)	total: 27.3s	remaining: 8.42s
764:	learn: 0.8279254	test: 0.7925460 best: 0.7925460 (764)	total: 27.3s	remaining: 8.38s
765:	learn: 0.8279254	test: 0.7924899 best: 0.7925460 (764)	total: 27.3s	remaining: 8.34s
766:	learn: 0.8279735	test: 0.7925460 best: 0.7925460 (764)	total: 27.3s	remaining: 8.3s
767:	learn: 0.8278533	test: 0.7927144 best: 0.7927144 (767)	total: 27.4s	remaining: 8.27s
768:	learn: 0.8279976	test: 0.7925460 best: 0.7927144 (767)	total: 27.4s	remaining: 8.23s
769:	learn: 0.8279014	test: 0.7926022 best: 0.7927144 (767)	total: 27.4s	remaining: 8.19s
770:	learn: 0.8280216	test: 0.7924899 best: 0.7927144 (767)	total: 27.5s	remaining: 8.16s
771:	learn: 0.8281900	test: 0.7924899 best: 0.7927144 (767)	total: 27.5s	remaining: 8.12s
772:	learn: 0.8280938	test: 0.7923776 best: 0.7927144 (767)	total: 27.5s	remaining: 8.09s
773:	learn: 0.8280457	test: 0.7924338 best: 0.7927144 (767)	total: 27.6s	remaining: 8.05s
774:	learn: 0.8283343	test: 0.7928267 best: 0.7928267 (774)	total: 27.6s	remaining: 8.01s
775:	learn: 0.8282381	test: 0.7926583 best: 0.7928267 (774)	total: 27.6s	remaining: 7.97s
776:	learn: 0.8281660	test: 0.7927705 best: 0.7928267 (774)	total: 27.7s	remaining: 7.94s
777:	learn: 0.8282622	test: 0.7927705 best: 0.7928267 (774)	total: 27.7s	remaining: 7.9s
778:	learn: 0.8280938	test: 0.7927705 best: 0.7928267 (774)	total: 27.7s	remaining: 7.86s
779:	learn: 0.8281179	test: 0.7928267 best: 0.7928267 (774)	total: 27.8s	remaining: 7.83s
780:	learn: 0.8282622	test: 0.7925460 best: 0.7928267 (774)	total: 27.8s	remaining: 7.79s
781:	learn: 0.8284787	test: 0.7924338 best: 0.7928267 (774)	total: 27.8s	remaining: 7.75s
782:	learn: 0.8289597	test: 0.7923215 best: 0.7928267 (774)	total: 27.8s	remaining: 7.71s
783:	learn: 0.8290319	test: 0.7923215 best: 0.7928267 (774)	total: 27.9s	remaining: 7.68s
784:	learn: 0.8288876	test: 0.7925460 best: 0.7928267 (774)	total: 27.9s	remaining: 7.65s
785:	learn: 0.8289597	test: 0.7926022 best: 0.7928267 (774)	total: 28s	remaining: 7.62s
786:	learn: 0.8290319	test: 0.7926583 best: 0.7928267 (774)	total: 28s	remaining: 7.58s
787:	learn: 0.8289597	test: 0.7924338 best: 0.7928267 (774)	total: 28s	remaining: 7.54s
788:	learn: 0.8288394	test: 0.7927144 best: 0.7928267 (774)	total: 28.1s	remaining: 7.51s
789:	learn: 0.8287673	test: 0.7928828 best: 0.7928828 (789)	total: 28.1s	remaining: 7.47s
790:	learn: 0.8290078	test: 0.7928267 best: 0.7928828 (789)	total: 28.1s	remaining: 7.43s
791:	learn: 0.8290078	test: 0.7928267 best: 0.7928828 (789)	total: 28.2s	remaining: 7.39s
792:	learn: 0.8289357	test: 0.7925460 best: 0.7928828 (789)	total: 28.2s	remaining: 7.36s
793:	learn: 0.8290559	test: 0.7926022 best: 0.7928828 (789)	total: 28.2s	remaining: 7.33s
794:	learn: 0.8290078	test: 0.7925460 best: 0.7928828 (789)	total: 28.3s	remaining: 7.3s
795:	learn: 0.8288876	test: 0.7926583 best: 0.7928828 (789)	total: 28.4s	remaining: 7.26s
796:	learn: 0.8288635	test: 0.7924899 best: 0.7928828 (789)	total: 28.4s	remaining: 7.23s
797:	learn: 0.8288635	test: 0.7926022 best: 0.7928828 (789)	total: 28.4s	remaining: 7.2s
798:	learn: 0.8290319	test: 0.7926583 best: 0.7928828 (789)	total: 28.5s	remaining: 7.16s
799:	learn: 0.8290559	test: 0.7924899 best: 0.7928828 (789)	total: 28.5s	remaining: 7.13s
800:	learn: 0.8290319	test: 0.7929389 best: 0.7929389 (800)	total: 28.5s	remaining: 7.09s
801:	learn: 0.8292002	test: 0.7927144 best: 0.7929389 (800)	total: 28.6s	remaining: 7.05s
802:	learn: 0.8291040	test: 0.7926022 best: 0.7929389 (800)	total: 28.6s	remaining: 7.01s
803:	learn: 0.8292724	test: 0.7926583 best: 0.7929389 (800)	total: 28.6s	remaining: 6.98s
804:	learn: 0.8293927	test: 0.7927144 best: 0.7929389 (800)	total: 28.7s	remaining: 6.94s
805:	learn: 0.8292965	test: 0.7927144 best: 0.7929389 (800)	total: 28.7s	remaining: 6.9s
806:	learn: 0.8293205	test: 0.7926022 best: 0.7929389 (800)	total: 28.7s	remaining: 6.87s
807:	learn: 0.8294889	test: 0.7926583 best: 0.7929389 (800)	total: 28.7s	remaining: 6.83s
808:	learn: 0.8294167	test: 0.7922654 best: 0.7929389 (800)	total: 28.8s	remaining: 6.79s
809:	learn: 0.8297294	test: 0.7928267 best: 0.7929389 (800)	total: 28.8s	remaining: 6.75s
810:	learn: 0.8298497	test: 0.7929951 best: 0.7929951 (810)	total: 28.8s	remaining: 6.72s
811:	learn: 0.8298497	test: 0.7930512 best: 0.7930512 (811)	total: 28.9s	remaining: 6.68s
812:	learn: 0.8299459	test: 0.7932196 best: 0.7932196 (812)	total: 28.9s	remaining: 6.64s
813:	learn: 0.8297535	test: 0.7931073 best: 0.7932196 (812)	total: 28.9s	remaining: 6.61s
814:	learn: 0.8298737	test: 0.7930512 best: 0.7932196 (812)	total: 28.9s	remaining: 6.57s
815:	learn: 0.8298256	test: 0.7929389 best: 0.7932196 (812)	total: 29s	remaining: 6.53s
816:	learn: 0.8300180	test: 0.7930512 best: 0.7932196 (812)	total: 29s	remaining: 6.5s
817:	learn: 0.8300661	test: 0.7929951 best: 0.7932196 (812)	total: 29s	remaining: 6.46s
818:	learn: 0.8298978	test: 0.7928828 best: 0.7932196 (812)	total: 29.1s	remaining: 6.42s
819:	learn: 0.8299459	test: 0.7929389 best: 0.7932196 (812)	total: 29.1s	remaining: 6.38s
820:	learn: 0.8299699	test: 0.7931073 best: 0.7932196 (812)	total: 29.1s	remaining: 6.35s
821:	learn: 0.8301864	test: 0.7931073 best: 0.7932196 (812)	total: 29.2s	remaining: 6.31s
822:	learn: 0.8300902	test: 0.7928828 best: 0.7932196 (812)	total: 29.2s	remaining: 6.28s
823:	learn: 0.8300421	test: 0.7928828 best: 0.7932196 (812)	total: 29.2s	remaining: 6.24s
824:	learn: 0.8300421	test: 0.7929389 best: 0.7932196 (812)	total: 29.2s	remaining: 6.2s
825:	learn: 0.8301864	test: 0.7931634 best: 0.7932196 (812)	total: 29.3s	remaining: 6.17s
826:	learn: 0.8302105	test: 0.7931634 best: 0.7932196 (812)	total: 29.3s	remaining: 6.13s
827:	learn: 0.8303548	test: 0.7929951 best: 0.7932196 (812)	total: 29.3s	remaining: 6.09s
828:	learn: 0.8301864	test: 0.7927144 best: 0.7932196 (812)	total: 29.4s	remaining: 6.05s
829:	learn: 0.8301383	test: 0.7928267 best: 0.7932196 (812)	total: 29.4s	remaining: 6.02s
830:	learn: 0.8300421	test: 0.7927144 best: 0.7932196 (812)	total: 29.4s	remaining: 5.98s
831:	learn: 0.8301383	test: 0.7925460 best: 0.7932196 (812)	total: 29.5s	remaining: 5.95s
832:	learn: 0.8301383	test: 0.7926022 best: 0.7932196 (812)	total: 29.5s	remaining: 5.91s
833:	learn: 0.8301624	test: 0.7924899 best: 0.7932196 (812)	total: 29.6s	remaining: 5.88s
834:	learn: 0.8302105	test: 0.7924899 best: 0.7932196 (812)	total: 29.6s	remaining: 5.85s
835:	learn: 0.8303067	test: 0.7926583 best: 0.7932196 (812)	total: 29.6s	remaining: 5.82s
836:	learn: 0.8302826	test: 0.7926583 best: 0.7932196 (812)	total: 29.7s	remaining: 5.78s
837:	learn: 0.8302586	test: 0.7926583 best: 0.7932196 (812)	total: 29.7s	remaining: 5.75s
838:	learn: 0.8302826	test: 0.7927144 best: 0.7932196 (812)	total: 29.8s	remaining: 5.71s
839:	learn: 0.8303307	test: 0.7929389 best: 0.7932196 (812)	total: 29.8s	remaining: 5.68s
840:	learn: 0.8304269	test: 0.7927144 best: 0.7932196 (812)	total: 29.8s	remaining: 5.64s
841:	learn: 0.8305232	test: 0.7923776 best: 0.7932196 (812)	total: 29.9s	remaining: 5.61s
842:	learn: 0.8304029	test: 0.7923215 best: 0.7932196 (812)	total: 29.9s	remaining: 5.57s
843:	learn: 0.8303788	test: 0.7924899 best: 0.7932196 (812)	total: 29.9s	remaining: 5.54s
844:	learn: 0.8303788	test: 0.7924899 best: 0.7932196 (812)	total: 30s	remaining: 5.5s
845:	learn: 0.8305953	test: 0.7925460 best: 0.7932196 (812)	total: 30s	remaining: 5.46s
846:	learn: 0.8308118	test: 0.7926022 best: 0.7932196 (812)	total: 30s	remaining: 5.42s
847:	learn: 0.8308118	test: 0.7923215 best: 0.7932196 (812)	total: 30.1s	remaining: 5.39s
848:	learn: 0.8309321	test: 0.7923776 best: 0.7932196 (812)	total: 30.1s	remaining: 5.35s
849:	learn: 0.8306915	test: 0.7924338 best: 0.7932196 (812)	total: 30.1s	remaining: 5.32s
850:	learn: 0.8308599	test: 0.7925460 best: 0.7932196 (812)	total: 30.1s	remaining: 5.28s
851:	learn: 0.8309321	test: 0.7926022 best: 0.7932196 (812)	total: 30.2s	remaining: 5.24s
852:	learn: 0.8308599	test: 0.7923776 best: 0.7932196 (812)	total: 30.2s	remaining: 5.21s
853:	learn: 0.8309802	test: 0.7922654 best: 0.7932196 (812)	total: 30.2s	remaining: 5.17s
854:	learn: 0.8311726	test: 0.7923215 best: 0.7932196 (812)	total: 30.3s	remaining: 5.13s
855:	learn: 0.8312688	test: 0.7923215 best: 0.7932196 (812)	total: 30.3s	remaining: 5.09s
856:	learn: 0.8313891	test: 0.7921531 best: 0.7932196 (812)	total: 30.3s	remaining: 5.06s
857:	learn: 0.8312928	test: 0.7923215 best: 0.7932196 (812)	total: 30.4s	remaining: 5.03s
858:	learn: 0.8313650	test: 0.7921531 best: 0.7932196 (812)	total: 30.4s	remaining: 4.99s
859:	learn: 0.8315093	test: 0.7922654 best: 0.7932196 (812)	total: 30.5s	remaining: 4.96s
860:	learn: 0.8315093	test: 0.7922093 best: 0.7932196 (812)	total: 30.5s	remaining: 4.93s
861:	learn: 0.8315815	test: 0.7922093 best: 0.7932196 (812)	total: 30.6s	remaining: 4.9s
862:	learn: 0.8315815	test: 0.7923215 best: 0.7932196 (812)	total: 30.6s	remaining: 4.86s
863:	learn: 0.8315093	test: 0.7923215 best: 0.7932196 (812)	total: 30.7s	remaining: 4.83s
864:	learn: 0.8316055	test: 0.7923776 best: 0.7932196 (812)	total: 30.7s	remaining: 4.8s
865:	learn: 0.8316777	test: 0.7923776 best: 0.7932196 (812)	total: 30.8s	remaining: 4.76s
866:	learn: 0.8318220	test: 0.7929389 best: 0.7932196 (812)	total: 30.8s	remaining: 4.73s
867:	learn: 0.8318701	test: 0.7928828 best: 0.7932196 (812)	total: 30.9s	remaining: 4.7s
868:	learn: 0.8319663	test: 0.7930512 best: 0.7932196 (812)	total: 30.9s	remaining: 4.66s
869:	learn: 0.8319663	test: 0.7930512 best: 0.7932196 (812)	total: 31s	remaining: 4.63s
870:	learn: 0.8320625	test: 0.7930512 best: 0.7932196 (812)	total: 31s	remaining: 4.59s
871:	learn: 0.8320144	test: 0.7930512 best: 0.7932196 (812)	total: 31s	remaining: 4.56s
872:	learn: 0.8320625	test: 0.7931634 best: 0.7932196 (812)	total: 31.1s	remaining: 4.52s
873:	learn: 0.8320385	test: 0.7931634 best: 0.7932196 (812)	total: 31.1s	remaining: 4.49s
874:	learn: 0.8321106	test: 0.7927705 best: 0.7932196 (812)	total: 31.1s	remaining: 4.45s
875:	learn: 0.8321828	test: 0.7925460 best: 0.7932196 (812)	total: 31.2s	remaining: 4.41s
876:	learn: 0.8321828	test: 0.7925460 best: 0.7932196 (812)	total: 31.2s	remaining: 4.38s
877:	learn: 0.8322790	test: 0.7926583 best: 0.7932196 (812)	total: 31.2s	remaining: 4.34s
878:	learn: 0.8321347	test: 0.7930512 best: 0.7932196 (812)	total: 31.3s	remaining: 4.3s
879:	learn: 0.8324233	test: 0.7933318 best: 0.7933318 (879)	total: 31.3s	remaining: 4.27s
880:	learn: 0.8324474	test: 0.7933880 best: 0.7933880 (880)	total: 31.3s	remaining: 4.23s
881:	learn: 0.8323512	test: 0.7933880 best: 0.7933880 (880)	total: 31.4s	remaining: 4.2s
882:	learn: 0.8325436	test: 0.7934441 best: 0.7934441 (882)	total: 31.4s	remaining: 4.16s
883:	learn: 0.8327120	test: 0.7933318 best: 0.7934441 (882)	total: 31.4s	remaining: 4.12s
884:	learn: 0.8326879	test: 0.7933880 best: 0.7934441 (882)	total: 31.4s	remaining: 4.09s
885:	learn: 0.8326879	test: 0.7938370 best: 0.7938370 (885)	total: 31.5s	remaining: 4.05s
886:	learn: 0.8326398	test: 0.7935002 best: 0.7938370 (885)	total: 31.5s	remaining: 4.01s
887:	learn: 0.8327601	test: 0.7935002 best: 0.7938370 (885)	total: 31.5s	remaining: 3.98s
888:	learn: 0.8328803	test: 0.7935564 best: 0.7938370 (885)	total: 31.6s	remaining: 3.94s
889:	learn: 0.8328803	test: 0.7935002 best: 0.7938370 (885)	total: 31.6s	remaining: 3.9s
890:	learn: 0.8329765	test: 0.7935002 best: 0.7938370 (885)	total: 31.6s	remaining: 3.87s
891:	learn: 0.8330487	test: 0.7933318 best: 0.7938370 (885)	total: 31.7s	remaining: 3.83s
892:	learn: 0.8330006	test: 0.7931634 best: 0.7938370 (885)	total: 31.7s	remaining: 3.8s
893:	learn: 0.8331209	test: 0.7934441 best: 0.7938370 (885)	total: 31.7s	remaining: 3.76s
894:	learn: 0.8331690	test: 0.7941176 best: 0.7941176 (894)	total: 31.7s	remaining: 3.72s
895:	learn: 0.8332892	test: 0.7936686 best: 0.7941176 (894)	total: 31.8s	remaining: 3.69s
896:	learn: 0.8335779	test: 0.7935002 best: 0.7941176 (894)	total: 31.8s	remaining: 3.65s
897:	learn: 0.8335298	test: 0.7934441 best: 0.7941176 (894)	total: 31.8s	remaining: 3.62s
```

```
898:	learn: 0.8336981	test: 0.7935564 best: 0.7941176 (894)	total: 31.9s	remaining: 3.58s
899:	learn: 0.8339146	test: 0.7939493 best: 0.7941176 (894)	total: 31.9s	remaining: 3.54s
900:	learn: 0.8338184	test: 0.7935002 best: 0.7941176 (894)	total: 31.9s	remaining: 3.51s
901:	learn: 0.8337222	test: 0.7933318 best: 0.7941176 (894)	total: 32s	remaining: 3.47s
902:	learn: 0.8339387	test: 0.7930512 best: 0.7941176 (894)	total: 32s	remaining: 3.44s
903:	learn: 0.8337703	test: 0.7932757 best: 0.7941176 (894)	total: 32s	remaining: 3.4s
904:	learn: 0.8340830	test: 0.7929389 best: 0.7941176 (894)	total: 32s	remaining: 3.36s
905:	learn: 0.8340108	test: 0.7929951 best: 0.7941176 (894)	total: 32.1s	remaining: 3.33s
906:	learn: 0.8341070	test: 0.7929951 best: 0.7941176 (894)	total: 32.1s	remaining: 3.29s
907:	learn: 0.8341311	test: 0.7931073 best: 0.7941176 (894)	total: 32.2s	remaining: 3.26s
908:	learn: 0.8341792	test: 0.7933318 best: 0.7941176 (894)	total: 32.2s	remaining: 3.23s
909:	learn: 0.8342032	test: 0.7933880 best: 0.7941176 (894)	total: 32.3s	remaining: 3.19s
910:	learn: 0.8338906	test: 0.7936125 best: 0.7941176 (894)	total: 32.3s	remaining: 3.15s
911:	learn: 0.8339627	test: 0.7933880 best: 0.7941176 (894)	total: 32.3s	remaining: 3.12s
912:	learn: 0.8339627	test: 0.7933318 best: 0.7941176 (894)	total: 32.4s	remaining: 3.08s
913:	learn: 0.8340349	test: 0.7933318 best: 0.7941176 (894)	total: 32.4s	remaining: 3.05s
914:	learn: 0.8341551	test: 0.7935002 best: 0.7941176 (894)	total: 32.5s	remaining: 3.01s
915:	learn: 0.8343235	test: 0.7935564 best: 0.7941176 (894)	total: 32.5s	remaining: 2.98s
916:	learn: 0.8344919	test: 0.7932757 best: 0.7941176 (894)	total: 32.5s	remaining: 2.94s
917:	learn: 0.8345400	test: 0.7934441 best: 0.7941176 (894)	total: 32.6s	remaining: 2.91s
918:	learn: 0.8346362	test: 0.7935564 best: 0.7941176 (894)	total: 32.6s	remaining: 2.87s
919:	learn: 0.8347805	test: 0.7937809 best: 0.7941176 (894)	total: 32.6s	remaining: 2.84s
920:	learn: 0.8347805	test: 0.7935564 best: 0.7941176 (894)	total: 32.6s	remaining: 2.8s
921:	learn: 0.8349008	test: 0.7935002 best: 0.7941176 (894)	total: 32.7s	remaining: 2.76s
922:	learn: 0.8348767	test: 0.7936125 best: 0.7941176 (894)	total: 32.7s	remaining: 2.73s
923:	learn: 0.8349729	test: 0.7938370 best: 0.7941176 (894)	total: 32.7s	remaining: 2.69s
924:	learn: 0.8348767	test: 0.7936125 best: 0.7941176 (894)	total: 32.8s	remaining: 2.66s
925:	learn: 0.8349489	test: 0.7933318 best: 0.7941176 (894)	total: 32.8s	remaining: 2.62s
926:	learn: 0.8349008	test: 0.7932196 best: 0.7941176 (894)	total: 32.8s	remaining: 2.58s
927:	learn: 0.8348527	test: 0.7933318 best: 0.7941176 (894)	total: 32.9s	remaining: 2.55s
928:	learn: 0.8348286	test: 0.7933318 best: 0.7941176 (894)	total: 32.9s	remaining: 2.51s
929:	learn: 0.8349248	test: 0.7930512 best: 0.7941176 (894)	total: 32.9s	remaining: 2.48s
930:	learn: 0.8349008	test: 0.7932196 best: 0.7941176 (894)	total: 33s	remaining: 2.44s
931:	learn: 0.8349729	test: 0.7933318 best: 0.7941176 (894)	total: 33s	remaining: 2.41s
932:	learn: 0.8347805	test: 0.7932757 best: 0.7941176 (894)	total: 33.1s	remaining: 2.38s
933:	learn: 0.8349008	test: 0.7934441 best: 0.7941176 (894)	total: 33.2s	remaining: 2.34s
934:	learn: 0.8349729	test: 0.7933880 best: 0.7941176 (894)	total: 33.2s	remaining: 2.31s
935:	learn: 0.8350932	test: 0.7935002 best: 0.7941176 (894)	total: 33.2s	remaining: 2.27s
936:	learn: 0.8351654	test: 0.7935002 best: 0.7941176 (894)	total: 33.3s	remaining: 2.24s
937:	learn: 0.8353337	test: 0.7936125 best: 0.7941176 (894)	total: 33.3s	remaining: 2.2s
938:	learn: 0.8352135	test: 0.7936125 best: 0.7941176 (894)	total: 33.3s	remaining: 2.17s
939:	learn: 0.8352856	test: 0.7935564 best: 0.7941176 (894)	total: 33.4s	remaining: 2.13s
940:	learn: 0.8350692	test: 0.7935002 best: 0.7941176 (894)	total: 33.4s	remaining: 2.09s
941:	learn: 0.8350451	test: 0.7935002 best: 0.7941176 (894)	total: 33.4s	remaining: 2.06s
942:	learn: 0.8352135	test: 0.7931073 best: 0.7941176 (894)	total: 33.5s	remaining: 2.02s
943:	learn: 0.8350932	test: 0.7931634 best: 0.7941176 (894)	total: 33.5s	remaining: 1.99s
944:	learn: 0.8351413	test: 0.7931634 best: 0.7941176 (894)	total: 33.5s	remaining: 1.95s
945:	learn: 0.8352856	test: 0.7932196 best: 0.7941176 (894)	total: 33.6s	remaining: 1.92s
946:	learn: 0.8354299	test: 0.7932196 best: 0.7941176 (894)	total: 33.6s	remaining: 1.88s
947:	learn: 0.8351894	test: 0.7932757 best: 0.7941176 (894)	total: 33.7s	remaining: 1.85s
948:	learn: 0.8354781	test: 0.7933318 best: 0.7941176 (894)	total: 33.7s	remaining: 1.81s
949:	learn: 0.8353818	test: 0.7935002 best: 0.7941176 (894)	total: 33.7s	remaining: 1.78s
950:	learn: 0.8355262	test: 0.7936125 best: 0.7941176 (894)	total: 33.8s	remaining: 1.74s
951:	learn: 0.8356705	test: 0.7935564 best: 0.7941176 (894)	total: 33.8s	remaining: 1.7s
952:	learn: 0.8355502	test: 0.7935002 best: 0.7941176 (894)	total: 33.8s	remaining: 1.67s
953:	learn: 0.8355983	test: 0.7936125 best: 0.7941176 (894)	total: 33.9s	remaining: 1.63s
954:	learn: 0.8355743	test: 0.7936125 best: 0.7941176 (894)	total: 33.9s	remaining: 1.6s
955:	learn: 0.8355743	test: 0.7936125 best: 0.7941176 (894)	total: 33.9s	remaining: 1.56s
956:	learn: 0.8356945	test: 0.7934441 best: 0.7941176 (894)	total: 34s	remaining: 1.52s
957:	learn: 0.8357667	test: 0.7937247 best: 0.7941176 (894)	total: 34s	remaining: 1.49s
958:	learn: 0.8359351	test: 0.7935564 best: 0.7941176 (894)	total: 34s	remaining: 1.45s
959:	learn: 0.8358870	test: 0.7935564 best: 0.7941176 (894)	total: 34.1s	remaining: 1.42s
960:	learn: 0.8360313	test: 0.7934441 best: 0.7941176 (894)	total: 34.1s	remaining: 1.38s
961:	learn: 0.8360553	test: 0.7934441 best: 0.7941176 (894)	total: 34.1s	remaining: 1.35s
962:	learn: 0.8360313	test: 0.7936125 best: 0.7941176 (894)	total: 34.1s	remaining: 1.31s
963:	learn: 0.8361515	test: 0.7938931 best: 0.7941176 (894)	total: 34.2s	remaining: 1.28s
964:	learn: 0.8361756	test: 0.7940054 best: 0.7941176 (894)	total: 34.2s	remaining: 1.24s
965:	learn: 0.8361756	test: 0.7939493 best: 0.7941176 (894)	total: 34.3s	remaining: 1.21s
966:	learn: 0.8363199	test: 0.7937809 best: 0.7941176 (894)	total: 34.3s	remaining: 1.17s
967:	learn: 0.8364642	test: 0.7938370 best: 0.7941176 (894)	total: 34.4s	remaining: 1.14s
968:	learn: 0.8366085	test: 0.7937247 best: 0.7941176 (894)	total: 34.4s	remaining: 1.1s
969:	learn: 0.8365604	test: 0.7939493 best: 0.7941176 (894)	total: 34.5s	remaining: 1.07s
970:	learn: 0.8365845	test: 0.7940615 best: 0.7941176 (894)	total: 34.5s	remaining: 1.03s
971:	learn: 0.8366326	test: 0.7938370 best: 0.7941176 (894)	total: 34.6s	remaining: 996ms
972:	learn: 0.8368010	test: 0.7939493 best: 0.7941176 (894)	total: 34.6s	remaining: 960ms
973:	learn: 0.8368010	test: 0.7940615 best: 0.7941176 (894)	total: 34.6s	remaining: 924ms
974:	learn: 0.8367288	test: 0.7940054 best: 0.7941176 (894)	total: 34.6s	remaining: 888ms
975:	learn: 0.8368731	test: 0.7938931 best: 0.7941176 (894)	total: 34.7s	remaining: 853ms
976:	learn: 0.8369212	test: 0.7939493 best: 0.7941176 (894)	total: 34.7s	remaining: 817ms
977:	learn: 0.8367769	test: 0.7937809 best: 0.7941176 (894)	total: 34.7s	remaining: 782ms
978:	learn: 0.8368731	test: 0.7940615 best: 0.7941176 (894)	total: 34.8s	remaining: 746ms
979:	learn: 0.8368731	test: 0.7940615 best: 0.7941176 (894)	total: 34.8s	remaining: 710ms
980:	learn: 0.8369453	test: 0.7940054 best: 0.7941176 (894)	total: 34.8s	remaining: 675ms
981:	learn: 0.8370415	test: 0.7938370 best: 0.7941176 (894)	total: 34.9s	remaining: 639ms
982:	learn: 0.8371137	test: 0.7937247 best: 0.7941176 (894)	total: 34.9s	remaining: 604ms
983:	learn: 0.8370174	test: 0.7938370 best: 0.7941176 (894)	total: 35s	remaining: 568ms
984:	learn: 0.8371618	test: 0.7941176 best: 0.7941176 (894)	total: 35s	remaining: 533ms
985:	learn: 0.8370896	test: 0.7941738 best: 0.7941738 (985)	total: 35s	remaining: 497ms
986:	learn: 0.8371618	test: 0.7942860 best: 0.7942860 (986)	total: 35.1s	remaining: 462ms
987:	learn: 0.8372339	test: 0.7941738 best: 0.7942860 (986)	total: 35.1s	remaining: 426ms
988:	learn: 0.8371137	test: 0.7942860 best: 0.7942860 (986)	total: 35.1s	remaining: 391ms
989:	learn: 0.8369453	test: 0.7941738 best: 0.7942860 (986)	total: 35.2s	remaining: 355ms
990:	learn: 0.8369934	test: 0.7939493 best: 0.7942860 (986)	total: 35.2s	remaining: 320ms
991:	learn: 0.8369934	test: 0.7938931 best: 0.7942860 (986)	total: 35.3s	remaining: 284ms
992:	learn: 0.8370174	test: 0.7942299 best: 0.7942860 (986)	total: 35.3s	remaining: 249ms
993:	learn: 0.8370655	test: 0.7945106 best: 0.7945106 (993)	total: 35.3s	remaining: 213ms
994:	learn: 0.8371858	test: 0.7943983 best: 0.7945106 (993)	total: 35.3s	remaining: 178ms
995:	learn: 0.8372580	test: 0.7942860 best: 0.7945106 (993)	total: 35.4s	remaining: 142ms
996:	learn: 0.8374985	test: 0.7942860 best: 0.7945106 (993)	total: 35.4s	remaining: 107ms
997:	learn: 0.8375225	test: 0.7941738 best: 0.7945106 (993)	total: 35.4s	remaining: 71ms
998:	learn: 0.8376909	test: 0.7942860 best: 0.7945106 (993)	total: 35.5s	remaining: 35.5ms
999:	learn: 0.8376188	test: 0.7941738 best: 0.7945106 (993)	total: 35.5s	remaining: 0us
bestTest = 0.7945105523
bestIteration = 993
Shrink model to first 994 iterations.
[1] "2021-05-19 00:12:39 IST"
```

In [100…

```
print(model)
```

```
CatBoost model (994 trees)
Loss function: MultiClass
Fit to 16 features
```

In [101…

```
print(catboost.get_feature_importance(model,pool = train_pool))
```

```
                          [,1]
amount_tsh            2.142860
funder               10.917798
gps_height            5.341195
longitude             9.588181
latitude              9.827019
basin                 4.138758
population            5.003162
scheme_management     2.855447
extraction_type_class 7.481851
management            2.753674
payment               6.359920
quality_group         1.111670
quantity_group       15.423340
source                6.855155
waterpoint_type_group 4.058728
age                   6.141242
```

In [102…

```
catboost_pred <- catboost.predict(model,test_pool,prediction_type="Class")
```

In [103…

```
print("Accuracy")
print(sum(ifelse(catboost_pred==test_data_label,1,0))/nrow(test_data_catboost)*100)
```

```
[1] "Accuracy"
[1] 79.45106
```

In [104…

```
confusion_matrix_catboost <-table(test_data_label,catboost_pred)
```

In [105…
```
print("Recall Per Class - CatBoost")
recall_catboost <- calc_recall(confusion_matrix_catboost)
print(recall_catboost)
```

```
[1] "Recall Per Class - CatBoost"
[1] 90.49194 25.94595 73.96640
```

In [106…
```
print("Precision Per Class - CatBoost")
precision_catboost <- calc_precision(confusion_matrix_catboost)
print(precision_catboost)
```

```
[1] "Precision Per Class - CatBoost"
[1] 77.94890 60.75949 83.96352
```

In [107…
```
print("F1 - SCore - CatBoost")
f1_catboost <- calc_f1(precision_catboost,recall_catboost)
print(f1_catboost)
```

```
[1] "F1 - SCore - CatBoost"
[1] 83.75341 36.36364 78.64854
```

Task 4 - Predicting the outputs for the test file

Preprocessing the test data set as the same as the training dataset above

In [108…
```
testing_data_complete <- data.frame(testing_data_x)
testing_data_x <- subset(testing_data_x,select=-num_private)
testing_data_x <- subset(testing_data_x,select=-scheme_name)
testing_data_x$scheme_management[testing_data_x$scheme_management==NA]<-'Other'
testing_data_x$funder[testing_data_x$funder==NA]<-'Others'
testing_data_x <- subset(testing_data_x,select = -c(water_quality,payment_type,waterpoint_type,source_type,extraction_type,installer,quantity))
date_recorded_in_years <- as.numeric(format(as.Date(testing_data_x$"date_recorded"),"%Y"))
testing_data_x["age"] <-   date_recorded_in_years - testing_data_x["construction_year"]
testing_data_x$age[testing_data_x["age"]==as.numeric(format(as.Date(testing_data_x$"date_recorded"),"%Y"))] <- median(testing_data_x$age)
testing_data_x <- subset(testing_data_x,select=-c(date_recorded,construction_year))
testing_data_x["scheme_management"][is.na(testing_data_x["scheme_management"])]<- "Other"
testing_data_x <- subset(testing_data_x,select=-c(permit,public_meeting))
testing_data_x <- subset(testing_data_x, select = -c(wpt_name,subvillage,region_code,district_code,lga,ward,region))
testing_data_x["amount_tsh"] <- lapply((testing_data_x["amount_tsh"]),as.numeric)
testing_data_x["amount_tsh"][is.na(testing_data_x["amount_tsh"])]<- mean((testing_data_x$amount_tsh))
testing_data_x["gps_height"][is.na(testing_data_x["gps_height"])]<- mean(as.numeric(testing_data_x$gps_height))
testing_data_x["latitude"][is.na(testing_data_x["latitude"])]<- mean(as.numeric(testing_data_x$latitude))
testing_data_x["longitude"][is.na(testing_data_x["longitude"])]<- mean(as.numeric(testing_data_x$longitude))
testing_data_x$longitude[testing_data_x$longitude==0]<-median(testing_data_x$longitude)
testing_data_x["funder"][is.na(testing_data_x["funder"])] <- "Other"
testing_data_x <- subset(testing_data_x,select=-c(id))
```

Predicting the functionality of water pump using the Random Forest model. We used Random Forest for our final selection as it gave the best accuracy.

In [109…
```
predict_test <- predict(model_final_rf,testing_data_x)
predict_final_dataframe <- data.frame("id"=testing_data_complete$id,"status_group"=predict_test)
```

In [110…
```
write.csv(predict_final_dataframe,"Submission.csv")
```

Conclusions

After running all the models, a summarised view of their accuracies is described below.

Model	Accuracy
Random Forest	81.23%
KNN	77.03%
Decision Tree	77.07%
CatBoost	79.63%

Class-wise Precision Recall and F1 at the time of code execution

Model	Class	Precision	Recall	F1-Score
Random Forest	Functional	80.88	89.68	85.06
Random Forest	Functional Needs Repair	56.76	33.70	42.29
Random Forest	Non Functional	84.81	78.29	81.42

Model	Class	Precision	Recall	F1-Score
KNN	Functional	78.42	85.49	85.06
KNN	Functional Needs Repair	63.52	40.89	49.76
KNN	Non Functional	76.98	78.39	74.61

Model	Class	Precision	Recall	F1-Score
Decision Tree	Functional	66.72	93.46	77.86
Decision Tree	Functional Needs Repair	NaN	0	NaN
Decision Tree	Non Functional	81.98	51.96	63.60

Model	Class	Precision	Recall	F1-Score
CatBoost	Functional	77.85	90.62	83.75
CatBoost	Functional Needs Repair	57.95	25.32	35.24
CatBoost	Non Functional	84.31	73.74	78.67

F1 Scores take into account both precision and recall. Random Forest and CatBoost are proving to be good for Functional and Non Functional. They produce a low F1 Score for Functional Needs Repair. Therefore we can say that they are a bit biased. Same for KNN. But KNN gives a better F1 Score for Functional Needs Repair. Decision Trees are unable to predict any Functional Needs Repair

Notes -

1. After the recording of the presentation we realised that the Lasso model gave us an accuracy of 68% and not 54% as mentioned in the video.
2. The PDF report has been generated in A2 layout mode to prevent trimming of the code and comments leading to small number of pages.