Pump_It_Up

Shahab Kazemi

September 11, 2016

Predicting the operating status of water pumps in Tanzania

Background

We want to know how we can predict which water pumps are faulty? Using data from Taarifa and the Tanzanian Ministry of Water, we want to predict which pumps are functional, which need some repairs, or which ones do not work at all? This prediction of water pumps' operating status must be based on a number of variables about what kind of pump is operating, when it was installed, and how it is managed. A smart understanding of which water points are likely to fail can help the government improve inspection and maintenance operations and thereby ensure that clean, potable water is available to communities across Tanzania. Additional background information about the Water Point Mapping system in Tanzania is available on the Water Point Mapping Tanzania website.

Problem definition

The problem is to build a model that predicts, whether a water pump in Tanzania is likely to fail. The model should be optimized in such a way that it enables the Tanzanian government and its partners to priorities their water pump inspection and maintenance operations and thereby ensure that as many users as possible will have stable access to clean, potable water. The case is adapted from an ongoing data mining competition on DrivenData sponsored by Taarifa.

Exploring the data

We are going to explore the data by searching for unanticipated trends and anomalies in order to gain understanding and ideas. Also it can help us to refine better discovery processes.

The data

The data set (pumpitup.csv) contains data from a cross-section of 59400 water pumps in Tanzania. There are 37 raw input variables:

- **amount_tsh** - Total static head (amount water available to water point)
- **date_recorded** - The date the row was entered (the date of the last status)
- **funder** - Who funded the well (maybe nominal - funder of pump) Maybe because of funder, and the way or amount his or her funding, the quality of pumps has been influenced (Good for managers to notice)
- **gps_height** - Altitude of the well
- **installer** - Organisation that installed the well (maybe the high rate of fault is related to some specific installers. So we can plan for managing them better) (changing them, replacing them .) (this input variable is completely related to the date of installing).
- **longitude** - GPS coordinate (maybe if the exact point of geographical location of the pump is highly correlated

Loading [Contrib]/a11y/accessibility-menu.js

with the target variable, the managers can think about the other environmental factors that is related to that specific location, e.g. we see that faults are more in some areas due to their longitudes or latitudes).
- **latitude** - GPS coordinate
- **wpt_name** - Name of the water point, if there is one (It cannot be related)
- **basin** - Geographic water basin (maybe nominal)
- **subvillage** - Geographic location (maybe nominal)
- **region** - Geographic location (maybe nominal)
- **region_code** - Geographic location (coded) (maybe nominal)
- **district_code** - Geographic location (coded) (maybe nominal)
- **lga** - Geographic location (maybe nominal)
- **ward** - Geographic location (maybe nominal)
- **population** - Population around the well
- **recorded_by** - Group entering this row of data (maybe the group has recorded a little or wrong information about the well, so there is a strange behavior of being in good status for pumps) (maybe nominal)
- **scheme_management** - Who operates the water point (maybe nominal)
- **scheme_name** - Who operates the water point (maybe nominal)
- **permit** - If the water point is permitted (maybe binary)
- **construction_year** - Year the water point was constructed
- **extraction_type** - The kind of extraction the water point uses (maybe nominal)
- **extraction_type_group** - The kind of extraction the water point uses (maybe nominal)
- **extraction_type_class** - The kind of extraction the water point uses (maybe nominal)
- **management** - How the water point is managed (maybe nominal)
- **management_group** - How the water point is managed (maybe nominal)
- **payment** - How the users of the water point pay (maybe the way of payment delays in getting the money, so the managers cannot manage the pumps well) (if the payment way relates highly to the way that installers install and maintain the pumps, the managers can decide about it to improve it) (maybe nominal)
- **payment_type** - How the users of the water point pay
- **water_quality** - The quality of the water
- **quality_group** - The quality of the water
- **quantity** - The quantity of water
- **quantity_group** - The quantity of water
- **source** - The source of the water
- **source_type** - The source of the water
- **source_class** - The source of the water
- **waterpoint_type** - The kind of water point
- **waterpoint_type_group** - The kind of water point
- **status_group** - operating status of the water point

Let’s have a look at our data:

Loading the data

```
pumpitup <- read.csv("input/pumpitup.csv",
                     na.strings = c("NA","NaN", " ", "", "unknown"))
#pumpitup <- read.csv("c:/aarhus/third semester/data science - visualization/exam/input/pumpit
up.csv",
#                     na.strings = c("NA","NaN", " ", "", "unknown"))

# Copy the data set in another one to keep the original
pump <- pumpitup
```

We are going to decide about each variable one by one, so whenever we need, we will go in deep with the data of the field (variable).

Before getting any sense of attributes, we have to look at target variable and also other variables …

```
summary(pump$status_group)
```

```
##              functional functional needs repair          non functional
##                   32259                    4317                   22824
```

Therefore, we have also 59400 records (pumps).

Target Variable

Because our target variable (status) has three values, we have to decide about converting it to other types or not. In fact, we do not know how being in a "needs repair" status can affect the population who use the pump's water. Maybe being in a "non-functional" status, affects less people in some areas than being in a "needs repair" in some other areas.

In fact, because the goal is maximizing the number of users who access to clean, potable water; and we do not know how much a "needs repair" pump affects this goal, we have to include the "needs repair" status to the "non-functional". So the main hypothesis in assuming "status_group" variable as a binary target variable is even "needs repair" pumps affect the cleanness and potability of the water, and we do not know how much.

Also we assume that if the "recorded_by" group has mentioned that a pump needs repair, then it means that not repairing the pump can affect the accessibility to clean and potable water. And this means "needs repair" has passed a threshold. This threshold means we can assume the target variable as a binary one.

Input Variables

We have assumed that some variables cannot be related to effects that take a pump into a "non-functional" or "needs repair". Because having many input variables means complexity of the model and also raising the variance of the target variable finally. In fact, the relationships between the status of the pumps and all the input variables are complex and still not fully understood. When applying learning techniques, variable and model selection are critical issues. Variable selection is useful to discard irrelevant inputs, leading to simpler models that are easier to interpret and that usually give better performances. So we try to ignore some unrelated (in our understanding) input variables in our model.

Getting sense of data

Grouping can help us to see how many values are outliers, so after that we can decide better if we have to drop it or factorize it:

1. **Population**

First we check population in each region

```
library(dplyr)

pump %>% tbl_df %>%
  group_by(region_code, district_code) %>%
  summarise("Number of pumps" = n(), population = sum(population)) %>%
  arrange(region_code, district_code)
```

```
## Source: local data frame [130 x 4]
## Groups: region_code [27]
```

```
##
##    region_code district_code `Number of pumps` population
##          <int>         <int>              <int>      <int>
## 1            1             0                 23          0
## 2            1             1                888          0
## 3            1             3                361          0
## 4            1             4                347          0
## 5            1             5                358          0
## 6            1             6                224          0
## 7            2             1                189        189
## 8            2             2               1206     217803
## 9            2             3                109      90952
## 10           2             5                201      48052
## # ... with 120 more rows
```

It seems that for some records the number of pumps is equal to the number of people (seems strange - region code = 2, district code = 1). Or even there are many pumps which anyone use them! (Region code = 1) We calculate how many records we have that the number of people are less or equal the number of pumps (in comparison to whole number of pumps we have: 59400).

```
pump %>% tbl_df %>%
  group_by(region_code, district_code) %>%
  summarise("Number of pumps" = n(), population = sum(population)) %>%
  filter(population <= n()) %>%
  arrange(region_code, district_code)
```

```
## Source: local data frame [39 x 4]
## Groups: region_code [7]
##
##    region_code district_code `Number of pumps` population
##          <int>         <int>              <int>      <int>
## 1            1             0                 23          0
## 2            1             1                888          0
## 3            1             3                361          0
## 4            1             4                347          0
## 5            1             5                358          0
## 6            1             6                224          0
## 7           11             2                530          0
## 8           12             1                298          0
## 9           12             2                500          0
## 10          12             3                871          0
## # ... with 29 more rows
```

We have 39 records of 130 that their data is strange. This number is too high for considering the population variable as an affecting variable.

Also we have:

```
summary(pump$population)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0    25.0   179.9   215.0 30500.0
```

More than 1/4 of records do not have enough information for population, so we can drop it.

2. **latitude** and **longitude**

Another assumption we can make is it is possible that two neighbor regions (or districts) can use each other's pumps if they are in the same longitude and latitude.

First we test how many distinct longitude and latitude we have.

```
pump %>% tbl_df %>%
  distinct(latitude, longitude)
```

```
## # A tibble: 57,520 × 2
##      latitude  longitude
##         <dbl>      <dbl>
## 1  -5.118154  33.12583
## 2  -9.395641  34.77072
## 3  -6.279268  36.11506
## 4  -3.187555  37.14743
## 5  -6.099290  36.16489
## 6  -6.972403  39.28612
## 7  -3.852983  33.22988
## 8  -6.719257  36.31362
## 9  -6.014358  35.93944
## 10 -2.530703  31.69337
## # ... with 57,510 more rows
```

57520 distinct record. So approximately we have a one to one correspondence between each pump and its **lat** and **long**.

```
pump %>% tbl_df %>%
  group_by(latitude, longitude) %>%
  summarise(n = n()) %>%
  arrange(n) %>%
  head(3)
```

```
## Source: local data frame [3 x 3]
## Groups: latitude [3]
##
##    latitude  longitude     n
##       <dbl>      <dbl> <int>
## 1 -11.64944  37.35114     1
## 2 -11.64838  37.35187     1
## 3 -11.58630  37.08270     1
```

```
pump %>% tbl_df %>%
  group_by(latitude, longitude) %>%
  summarise(n = n()) %>%
  arrange(desc(n)) %>%
  head(3)
```

```
## Source: local data frame [3 x 3]
## Groups: latitude [3]
##
##      latitude longitude     n
##         <dbl>     <dbl> <int>
## 1 -0.00000002   0.00000  1812
## 2 -7.17720290  37.31425     2
## 3 -7.17715480  37.31891     2
```

We see 1812 records out of 59400. so the best we can do is to omit them. Because when the ‘longitude’ equals to zero, we have our very small values for ‘latitude’ then we just drop the records with zero value in ‘longitude’.

```
library(Hmisc)
pump[, 7][pump[, 7] == 0] <- NA
pump <- pump[-which(is.na(pump$longitude)), ]
```

And again test for correct values:

```
pump %>% tbl_df %>%
  distinct(latitude, longitude)
```

```
## # A tibble: 57,519 × 2
##     latitude longitude
##        <dbl>     <dbl>
## 1  -5.118154  33.12583
## 2  -9.395641  34.77072
## 3  -6.279268  36.11506
## 4  -3.187555  37.14743
## 5  -6.099290  36.16489
## 6  -6.972403  39.28612
## 7  -3.852983  33.22988
## 8  -6.719257  36.31362
## 9  -6.014358  35.93944
## 10 -2.530703  31.69337
## # ... with 57,509 more rows
```

```
pump %>% tbl_df %>%
  group_by(latitude, longitude) %>%
  summarise(n = n()) %>%
  arrange(desc(n)) %>%
  head(3)
```

```
## Source: local data frame [3 x 3]
## Groups: latitude [3]
##
##    latitude longitude     n
##       <dbl>     <dbl> <int>
## 1 -7.177203  37.31425     2
## 2 -7.177155  37.31891     2
## 3 -7.175174  37.32891     2
```

3. **basin**

```
str(pump$basin)
```

```
##  Factor w/ 9 levels "Internal","Lake Nyasa",..: 4 7 9 6 9 9 1 7 9 5 ...
```

```
pump %>% tbl_df %>%
  group_by(basin) %>%
  summarise("Number of pumps" = n())
```

```
## # A tibble: 9 × 2
##                    basin `Number of pumps`
##                   <fctr>             <int>
## 1               Internal              7785
## 2             Lake Nyasa              5085
## 3             Lake Rukwa              2454
## 4        Lake Tanganyika              6333
## 5          Lake Victoria              8535
## 6                Pangani              8940
## 7                 Rufiji              7976
## 8 Ruvuma / Southern Coast             4493
## 9             Wami / Ruvu             5987
```

4. **subvillage**

```
str(pump$subvillage)
```

```
##  Factor w/ 19287 levels "'A' Kati","##",..: 9075 8911 17875 18715 8624 13586 14807 13631
10150 10822 ...
```

19288 of 59400 record. Too many record for learning. We will drop it.

5. **region**, **region_code**

```
str(pump$region)
```

```
##  Factor w/ 21 levels "Arusha","Dar es Salaam",..: 20 4 3 7 3 15 18 3 3 5 ...
```

```
summary(pump$region)
```

```
##        Arusha Dar es Salaam        Dodoma        Iringa        Kagera
##          3350           805          2201          5294          3316
##        Kigoma   Kilimanjaro         Lindi       Manyara          Mara
##          2816          4379          1546          1583          1969
##         Mbeya      Morogoro        Mtwara        Mwanza         Pwani
##          4639          4006          1730          2295          2635
##         Rukwa        Ruvuma     Shinyanga       Singida        Tabora
##          1808          2640          3977          2093          1959
##         Tanga
```

```
##          2547
```

```
str(pump$region_code)
```

```
##  int [1:57588] 14 11 1 3 1 60 17 1 1 18 ...
```

```
pump$region_code <- as.factor(pump$region_code)
str(pump$region_code)
```

```
##  Factor w/ 27 levels "1","2","3","4",..: 14 11 1 3 1 24 17 1 1 18 ...
```

```
summary(pump$region_code)
```

```
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2201 3024 4379 2513 4040 1609  805  300  390 2640 5297 4639 2093 1979 1808
##   16   17   18   19   20   21   24   40   60   80   90   99
## 2816 3954 3324 2295 1969 1583  326    1 1025 1238  917  423
```

```
pump %>% tbl_df %>%
  group_by(region, region_code) %>%
  summarise("Number of pumps" = n())
```

```
## Source: local data frame [31 x 3]
## Groups: region [?]
##
##           region region_code `Number of pumps`
##           <fctr>      <fctr>             <int>
## 1         Arusha           2              3024
## 2         Arusha          24               326
## 3  Dar es Salaam           7               805
## 4         Dodoma           1              2201
## 5         Iringa          11              5294
## 6         Kagera          18              3316
## 7         Kigoma          16              2816
## 8    Kilimanjaro           3              4379
## 9          Lindi           8               300
## 10         Lindi          18                 8
## # ... with 21 more rows
```

Appearantly ‘region’ and ‘region_code’ are not connected to each other, so we are not going to drop none of them.

6. **district**

```
str(pump$district_code)
```

```
##  int [1:57588] 3 4 4 5 4 43 3 1 5 8 ...
```

```
pump$district_code <- as.factor(pump$district_code)
str(pump$district_code)
```

```
##  Factor w/ 20 levels "0","1","2","3",..: 4 5 5 6 5 14 4 2 6 9 ...
```

```
summary(pump$district_code)
```

```
##     0     1     2     3     4     5     6     7     8    13    23    30
##    23 11146 10909  9998  8996  4356  3586  3343  1043   391   293   995
##    33    43    53    60    62    63    67    80
##   874   505   745    63   109   195     6    12
```

```
pump %>% tbl_df %>%
  group_by(region, region_code, district_code) %>%
  summarise("Number of pumps" = n())
```

```
## Source: local data frame [134 x 4]
## Groups: region, region_code [?]
##
##           region region_code district_code `Number of pumps`
##           <fctr>      <fctr>        <fctr>             <int>
## 1         Arusha           2             1               189
## 2         Arusha           2             2              1206
## 3         Arusha           2             3               109
## 4         Arusha           2             5               201
## 5         Arusha           2             6               310
## 6         Arusha           2             7              1009
## 7         Arusha          24            30               326
## 8  Dar es Salaam           7             1                93
## 9  Dar es Salaam           7             2               497
## 10 Dar es Salaam           7             3               215
## # ... with 124 more rows
```

‘district’ is not connected to other two variables, so we are not going to drop none of them.

7. **wpt_name**

```
str(pump$wpt_name)
```

```
##  Factor w/ 37400 levels "24","A Kulwa",..: 29563 32437 1067 446 2598 20291 28857 4091 906
7 35289 ...
```

More than half of all arecords we have in wpt_name. So we will ignore it.

8. **scheme_name**

```
str(pump$scheme_name)
```

```
##  Factor w/ 2696 levels "14 Kambarage",..: NA NA 1460 1088 NA NA NA 1140 525 1325 ...
```

Too many different scheme_name, so we will drop it.

9. **scheme_management**

```
str(pump$scheme_management)
```

```
##  Factor w/ 12 levels "Company","None",..: 8 NA 8 10 8 5 8 8 8 8 ...
```

```
pump %>% tbl_df %>%
  group_by(scheme_management) %>%
  summarise("Number of pumps" = n())
```

```
## # A tibble: 13 × 2
##     scheme_management `Number of pumps`
##                <fctr>             <int>
## 1             Company              1061
## 2                None                 1
## 3               Other               765
## 4          Parastatal              1607
## 5    Private operator              1063
## 6                 SWC                97
## 7               Trust                72
## 8                 VWC             36143
## 9      Water authority             3151
## 10         Water Board             2747
## 11                WUA              2882
## 12                WUG              4249
## 13                 NA              3750
```

We have to check if the ‘NA’ name means the ‘Null Value’:

```
pump %>% tbl_df %>%
  group_by(scheme_management) %>%
  summarise("Number of pumps" = n()) %>%
  filter(is.na(scheme_management))
```

```
## # A tibble: 1 × 2
##   scheme_management `Number of pumps`
##              <fctr>             <int>
## 1                NA              3750
```

Null values are few, but the most records (pumps) belong to a specific scheme_management. However, we are not going to drop it.

We can drop the records with null values.

```
pump[, 19][pump[, 19] == "None"] <- NA
pump <- pump[-which(is.na(pump$scheme_management)), ]
```

We check again if we have dropped null values correctly.

```
pump %>% tbl_df %>%
  group_by(scheme_management) %>%
  summarise("Number of pumps" = n())
```

```
## # A tibble: 11 × 2
##    scheme_management `Number of pumps`
##               <fctr>             <int>
## 1            Company              1061
## 2              Other               765
## 3         Parastatal              1607
## 4   Private operator              1063
## 5                SWC                97
## 6              Trust                72
## 7                VWC             36143
## 8    Water authority              3151
## 9        Water Board              2747
## 10               WUA              2882
## 11               WUG              4249
```

10. **permit**

```
str(pump$permit)
```

```
##  Factor w/ 2 levels "False","True": 2 2 2 2 1 2 2 1 2 1 ...
```

```
pump %>% tbl_df %>%
  group_by(permit) %>%
  summarise("Number of pumps" = n())
```

```
## # A tibble: 3 × 2
##   permit `Number of pumps`
##   <fctr>             <int>
## 1  False             15971
## 2   True             34851
## 3     NA              3015
```

A few records with null values, so we can drop them.

```
pump <- pump[-which(is.na(pump$permit)), ]
pump %>% tbl_df %>%
  group_by(permit) %>%
  summarise("Number of pumps" = n())
```

```
## # A tibble: 2 × 2
##   permit `Number of pumps`
##   <fctr>             <int>
## 1  False             15971
## 2   True             34851
```

11. **extaction_type**, **group**, **class**

```
str(pump$extraction_type)
```

```
##  Factor w/ 18 levels "afridev","cemo",..: 1 8 4 9 15 10 10 8 8 15 ...
```

```
str(pump$extraction_type_group)
```

```
##  Factor w/ 13 levels "afridev","gravity",..: 1 5 2 6 11 7 7 5 5 11 ...
```

```
str(pump$extraction_type_class)
```

```
##  Factor w/ 7 levels "gravity","handpump",..: 2 3 1 2 6 4 4 3 3 6 ...
```

```
pump %>% tbl_df %>%
  group_by(ext = extraction_type, grp = extraction_type_group,
           cls = extraction_type_class) %>%
  summarise(n())
```

```
## Source: local data frame [18 x 4]
## Groups: ext, grp [?]
##
##                          ext             grp          cls `n()`
##                       <fctr>          <fctr>       <fctr> <int>
## 1                    afridev         afridev     handpump  1405
## 2                       cemo other motorpump    motorpump    89
## 3                     climax other motorpump    motorpump    32
## 4                    gravity         gravity      gravity 24693
## 5              india mark ii   india mark ii     handpump  2040
## 6             india mark iii  india mark iii     handpump    90
## 7                        ksb     submersible  submersible  1346
## 8                       mono            mono    motorpump  2519
## 9                nira/tanira     nira/tanira     handpump  6202
## 10                     other           other        other  4627
## 11 other - mkulima/shinyanga  other handpump     handpump     2
## 12         other - play pump  other handpump     handpump    76
## 13         other - rope pump       rope pump    rope pump   280
## 14            other - swn 81  other handpump     handpump   218
## 15               submersible     submersible  submersible  4215
## 16                    swn 80          swn 80     handpump  2863
## 17                    walimi  other handpump     handpump    20
## 18                  windmill    wind-powered wind-powered   105
```

12. **management** and **management_group**

```
str(pump$management)
```

```
##  Factor w/ 11 levels "company","other",..: 7 7 9 7 5 7 7 5 7 5 ...
```

```
str(pump$management_group)
```

```
##  Factor w/ 4 levels "commercial","other",..: 4 4 4 4 1 4 4 1 4 1 ...
```

```
pump %>% tbl_df %>%
  group_by(mng = management, grp = management_group) %>%
  summarise(n())
```

```
## Source: local data frame [12 x 3]
## Groups: mng [?]
##
##                   mng        grp `n()`
##                <fctr>     <fctr> <int>
## 1             company commercial   657
## 2               other      other   557
## 3    other - school      other    99
## 4          parastatal parastatal  1513
## 5  private operator commercial  1773
## 6               trust commercial    76
## 7                 vwc user-group 35198
## 8   water authority commercial   822
## 9        water board user-group  2829
## 10                wua user-group  2460
## 11                wug user-group  4750
## 12                 NA         NA    88
```

A few records with null values, so we can drop them.

```
pump <- pump[-which(is.na(pump$management)), ]
pump %>% tbl_df %>%
  group_by(mng = management, grp = management_group) %>%
  summarise(n())
```

```
## Source: local data frame [11 x 3]
## Groups: mng [?]
##
##                   mng        grp `n()`
##                <fctr>     <fctr> <int>
## 1             company commercial   657
## 2               other      other   557
## 3    other - school      other    99
## 4          parastatal parastatal  1513
## 5  private operator commercial  1773
## 6               trust commercial    76
## 7                 vwc user-group 35198
## 8   water authority commercial   822
## 9        water board user-group  2829
## 10                wua user-group  2460
## 11                wug user-group  4750
```

13. **payment** and **payment_type**

```
str(pump$payment)
```

```
##  Factor w/ 6 levels "never pay","other",..: NA 5 5 NA 5 1 1 5 1 5 ...
```

```
str(pump$payment_type)
```

```
##  Factor w/ 6 levels "annually","monthly",..: NA 6 6 NA 6 3 3 6 3 6 ...
```

```
pump %>% tbl_df %>%
  group_by(pay = payment, typ = payment_type) %>%
  summarise(n())
```

```
## Source: local data frame [7 x 3]
## Groups: pay [?]
##
##                     pay        typ `n()`
##                  <fctr>     <fctr> <int>
## 1             never pay  never pay 21152
## 2                 other      other   820
## 3          pay annually   annually  3525
## 4           pay monthly    monthly  7730
## 5        pay per bucket per bucket  8489
## 6 pay when scheme fails on failure  3593
## 7                    NA         NA  5425
```

We can drop ‘payment’ and also the records with null values.

```
pump <- pump[-which(is.na(pump$payment)), ]
pump %>% tbl_df %>%
  group_by(pay = payment, typ = payment_type) %>%
  summarise(n())
```

```
## Source: local data frame [6 x 3]
## Groups: pay [?]
##
##                     pay        typ `n()`
##                  <fctr>     <fctr> <int>
## 1             never pay  never pay 21152
## 2                 other      other   820
## 3          pay annually   annually  3525
## 4           pay monthly    monthly  7730
## 5        pay per bucket per bucket  8489
## 6 pay when scheme fails on failure  3593
```

14. **water_quality** and **quality_group**

```
str(pump$water_quality)
```

```
##  Factor w/ 7 levels "coloured","fluoride",..: 7 7 7 7 4 7 7 5 7 7 ...
```

```
str(pump$quality_group)
```

```
##  Factor w/ 5 levels "colored","fluoride",..: 3 3 3 3 4 3 3 5 3 3 ...
```

```
pump %>% tbl_df %>%
  group_by(quality = water_quality, grp = quality_group) %>%
  summarise(n())
```

```
## Source: local data frame [8 x 3]
## Groups: quality [?]
##
##               quality      grp `n()`
##                <fctr>   <fctr> <int>
## 1            coloured  colored   384
## 2            fluoride fluoride   111
## 3 fluoride abandoned fluoride    11
## 4               milky    milky   281
## 5               salty    salty  3712
## 6    salty abandoned    salty   201
## 7                soft     good 39947
## 8                  NA       NA   662
```

We can drop the records with null values.

```
pump <- pump[-which(is.na(pump$water_quality)), ]
pump %>% tbl_df %>%
  group_by(quality = water_quality, grp = quality_group) %>%
  summarise(n())
```

```
## Source: local data frame [7 x 3]
## Groups: quality [?]
##
##               quality      grp `n()`
##                <fctr>   <fctr> <int>
## 1            coloured  colored   384
## 2            fluoride fluoride   111
## 3 fluoride abandoned fluoride    11
## 4               milky    milky   281
## 5               salty    salty  3712
## 6    salty abandoned    salty   201
## 7                soft     good 39947
```

15. **quantity** and **quantity_group**

```
str(pump$quantity)
```

```
##  Factor w/ 4 levels "dry","enough",..: 3 2 2 4 3 1 3 3 1 2 ...
```

```
str(pump$quantity_group)
```

```
##  Factor w/ 4 levels "dry","enough",..: 3 2 2 4 3 1 3 3 1 2 ...
```

```
pump %>% tbl_df %>%
  group_by(quantity, grp = quantity_group) %>%
  summarise(n())
```

```
## Source: local data frame [5 x 3]
## Groups: quantity [?]
##
##       quantity          grp `n()`
##         <fctr>       <fctr> <int>
## 1          dry          dry  4080
## 2       enough       enough 27056
## 3 insufficient insufficient 10335
## 4     seasonal     seasonal  3032
## 5           NA           NA   144
```

We can drop quantity_group and also the null records.

```
pump <- pump[-which(is.na(pump$quantity)), ]
pump %>% tbl_df %>%
  group_by(quantity, grp = quantity_group) %>%
  summarise(n())
```

```
## Source: local data frame [4 x 3]
## Groups: quantity [?]
##
##       quantity          grp `n()`
##         <fctr>       <fctr> <int>
## 1          dry          dry  4080
## 2       enough       enough 27056
## 3 insufficient insufficient 10335
## 4     seasonal     seasonal  3032
```

16. **source**, **type**, **class**

```
str(pump$source)
```

```
##  Factor w/ 9 levels "dam","hand dtw",..: 4 9 4 8 9 4 4 4 9 9 ...
```

```
str(pump$source_type)
```

```
##  Factor w/ 7 levels "borehole","dam",..: 1 7 1 6 7 1 1 1 7 7 ...
```

```
str(pump$source_class)
```

```
##  Factor w/ 2 levels "groundwater",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
pump %>% tbl_df %>%
  group_by(source, type = source_type, cls = source_class) %>%
  summarise(n())
```

```
## Source: local data frame [10 x 4]
## Groups: source, type [?]
##
##                   source                 type         cls `n()`
##                   <fctr>               <fctr>      <fctr> <int>
## 1                    dam                  dam     surface   553
## 2               hand dtw             borehole groundwater   774
## 3                   lake           river/lake     surface   473
## 4            machine dbh             borehole groundwater  8491
## 5                  other                other          NA   160
## 6  rainwater harvesting rainwater harvesting     surface  1167
## 7                  river           river/lake     surface  8609
## 8           shallow well         shallow well groundwater 10587
## 9                 spring               spring groundwater 13664
## 10                    NA                other          NA    25
```

We can drop the null records.

```
pump <- pump[-which(is.na(pump$source_class)), ]
pump %>% tbl_df %>%
  group_by(source, type = source_type, cls = source_class) %>%
  summarise(n())
```

```
## Source: local data frame [8 x 4]
## Groups: source, type [?]
##
##                   source                 type         cls `n()`
##                   <fctr>               <fctr>      <fctr> <int>
## 1                    dam                  dam     surface   553
## 2               hand dtw             borehole groundwater   774
## 3                   lake           river/lake     surface   473
## 4            machine dbh             borehole groundwater  8491
## 5 rainwater harvesting rainwater harvesting     surface  1167
## 6                  river           river/lake     surface  8609
## 7           shallow well         shallow well groundwater 10587
## 8                 spring               spring groundwater 13664
```

17. **waterpoint_type** and **group**

```
str(pump$waterpoint_type)
```

```
##  Factor w/ 7 levels "cattle trough",..: 3 2 3 7 6 3 3 2 2 2 ...
```

```
str(pump$waterpoint_type_group)
```

```
##  Factor w/ 6 levels "cattle trough",..: 2 2 2 6 5 2 2 2 2 2 ...
```

```
pump %>% tbl_df %>%
  group_by(type = waterpoint_type, grp = waterpoint_type_group) %>%
  summarise(n())
```

```
## Source: local data frame [7 x 3]
## Groups: type [?]
##
##                          type                grp `n()`
##                        <fctr>             <fctr> <int>
## 1               cattle trough      cattle trough    72
## 2          communal standpipe communal standpipe 23047
## 3 communal standpipe multiple communal standpipe  4928
## 4                         dam                dam     6
## 5                   hand pump          hand pump 12010
## 6             improved spring    improved spring   483
## 7                       other              other  3772
```

18. **amount_tsh**

```
str(pump$amount_tsh)
```

```
##  num [1:44318] 0 10 50 0 0 0 0 30 0 0 ...
```

```
pump %>% tbl_df %>%
  group_by(amount_tsh) %>%
  summarise("Number of pumps" = n())
```

```
## # A tibble: 94 × 2
##    amount_tsh `Number of pumps`
##         <dbl>             <int>
## 1        0.00             27928
## 2        0.20                 2
## 3        0.25                 1
## 4        1.00                 3
## 5        2.00                13
## 6        5.00               373
## 7        6.00               190
## 8        7.00                69
## 9        9.00                 1
## 10      10.00               786
## # ... with 84 more rows
```

With more than 2/3 of all records with zero amount of water, we are going to drop amount_tsh.

19. **funder**

```
str(pump$funder)
```

```
##  Factor w/ 1897 levels "0","A/co Germany",..: 839 437 1271 457 1830 1830 280 761 1545 1866
...
```

Too many records for 'funder'. We are going to drop it.

20. **gps_height**

```
str(pump$gps_height)
```

```
##  int [1:44318] 0 1639 28 0 0 0 0 64 1014 1606 ...
```

```
pump %>% tbl_df %>%
  group_by(gps_height) %>%
  summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 2,425 × 2
##    gps_height     n
##         <int> <int>
## 1           0 14479
## 2         -15    54
## 3         -13    46
## 4         -18    43
## 5         -16    43
## 6         -14    42
## 7         -20    41
## 8         -23    40
## 9         -19    40
## 10        -11    38
## # ... with 2,415 more rows
```

because close to half of records have 'height' of zero, we cannot use it.

21. **installer**

```
str(pump$installer)
```

```
##  Factor w/ 2144 levels "-","0","A.D.B",..: 1000 229 1461 571 564 288 375 1034 434 434 ...
```

Too many records for 'installer'. We are going to drop it.

22. **lga and ward**

```
str(pump$lga)
```

```
##  Factor w/ 125 levels "Arusha Rural",..: 12 17 70 105 77 15 13 96 112 1 ...
```

```
str(pump$ward)
```

```
##  Factor w/ 2092 levels "Aghondi","Akheri",..: 1357 1091 2061 2027 2079 392 1711 1256 660 1
763 ...
```

Too many records for ‘lga’ and ‘ward’. We are going to drop them.

23. **construction_year**

```
str(pump$construction_year)
```

```
##  int [1:44318] 0 1999 0 0 0 0 0 2007 1984 2010 ...
```

```
pump %>% tbl_df %>%
  group_by(construction_year) %>%
  summarise(n = n()) %>%
  arrange(desc(n))
```

```
## # A tibble: 55 × 2
##    construction_year     n
##                <int> <int>
## 1                  0 14736
## 2               2008  2219
## 3               2009  2071
## 4               2010  2046
## 5               2000  1362
## 6               2007  1246
## 7               2006  1219
## 8               2003  1062
## 9               2011   925
## 10              1978   902
## # ... with 45 more rows
```

because more than 1/3 of records have ‘year’ of zero, we cannot use it.

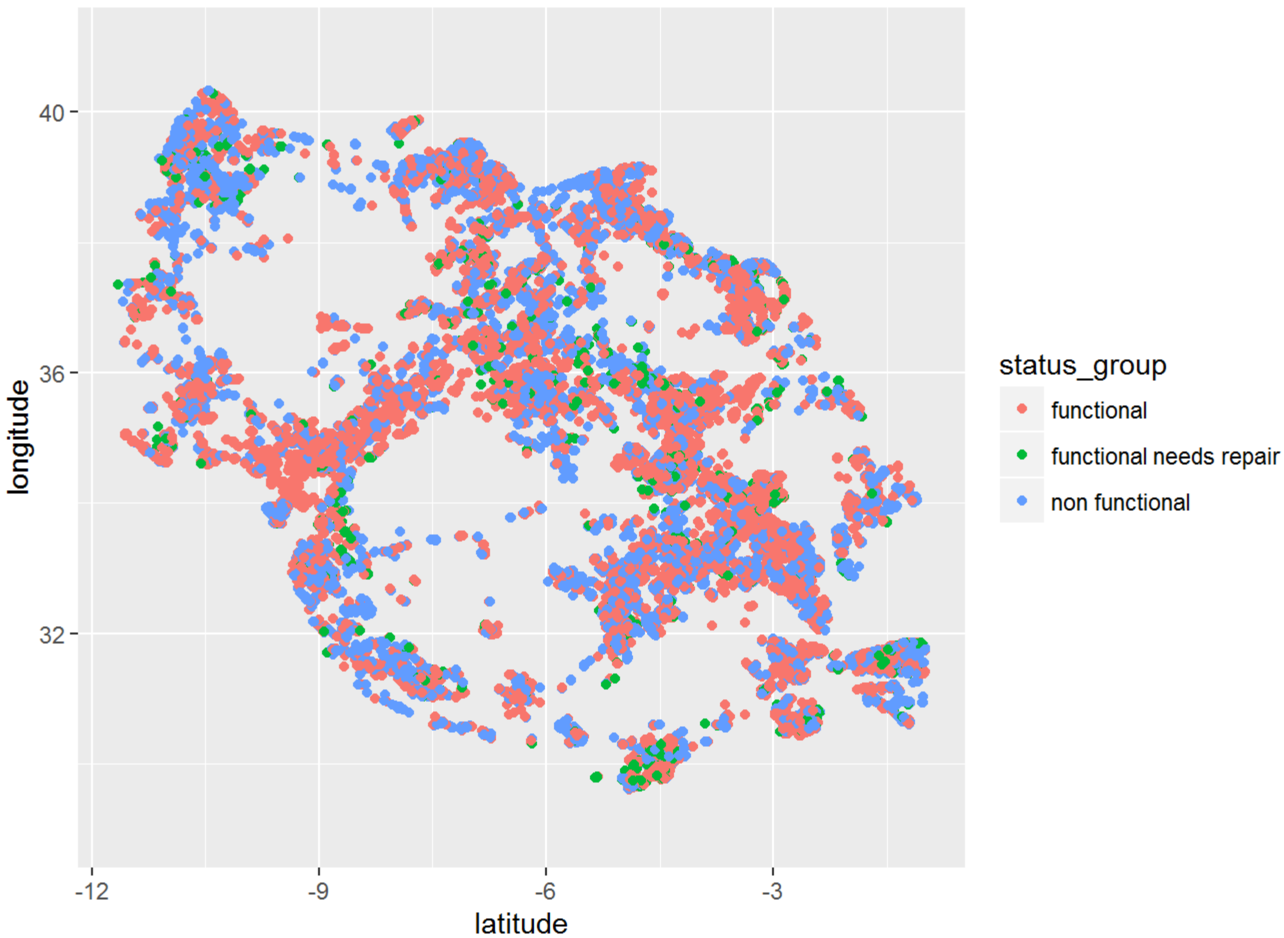
Dropping some variables

```
drop_pump <- subset(pump,
                    select = -c(id, date_recorded, wpt_name, subvillage, lga,
                                ward, recorded_by, scheme_name,
                                construction_year, payment, quantity_group,
                                amount_tsh, funder, gps_height, installer,
                                population))
```

Plotting

One of the best plots we can use is scatter plot. It shows how the continuous variables are related to the target variable. So we can check it how the ‘latitude’ and ‘longitude’ variables are connected to the ‘functionality’ of pumps.

```r
library(ggplot2)
drop_pump %>% ggplot(aes(x = latitude, y = longitude, color = status_group)) +
  geom_point() +
  scale_y_continuous(limits = c(29, 41))
```

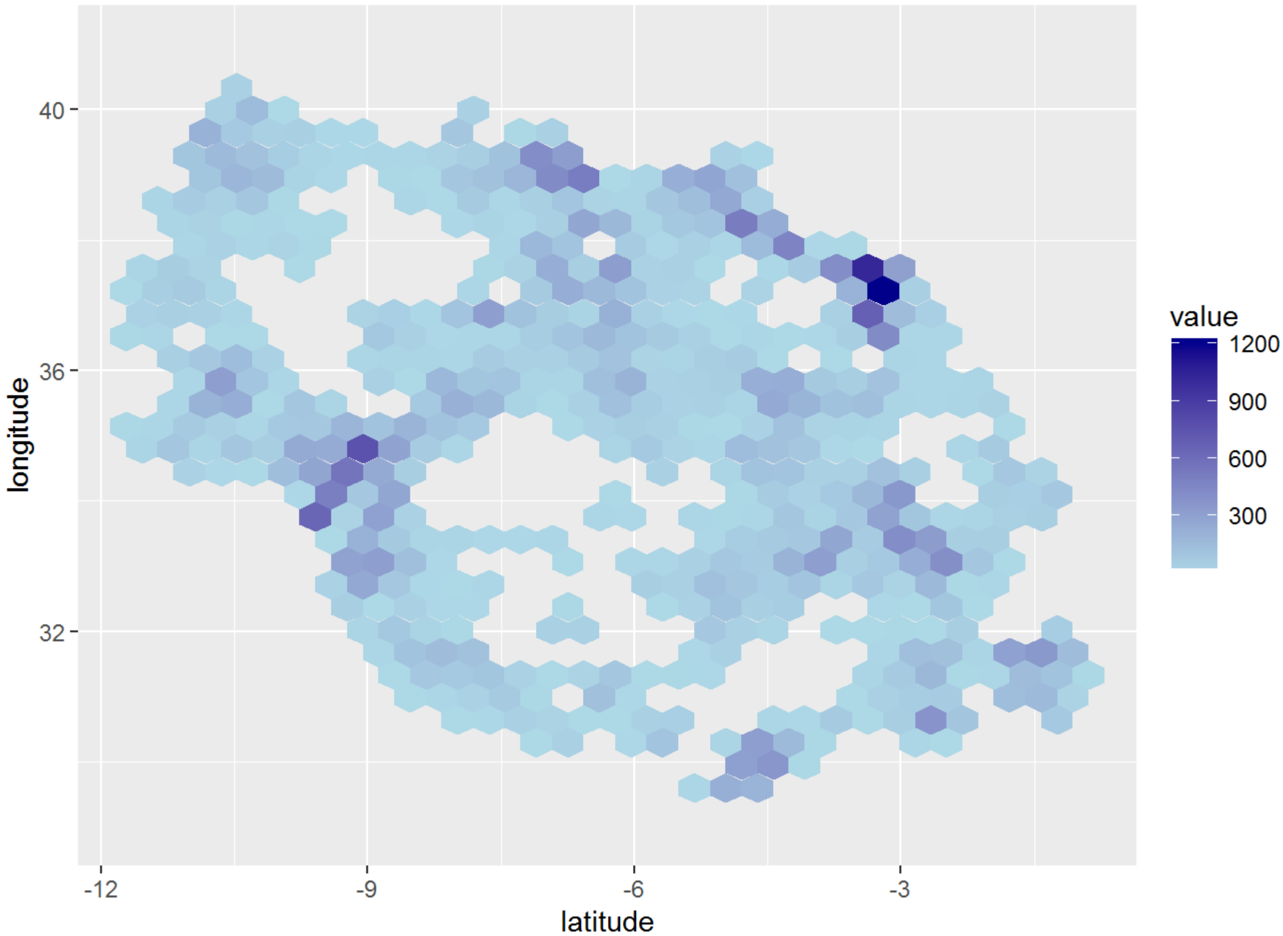


The plot shows in all areas, the functional pumps and non-functional ones are intermingled to each other, so we cannot decide certainly about how the government can manage the areas with non-functional pumps.

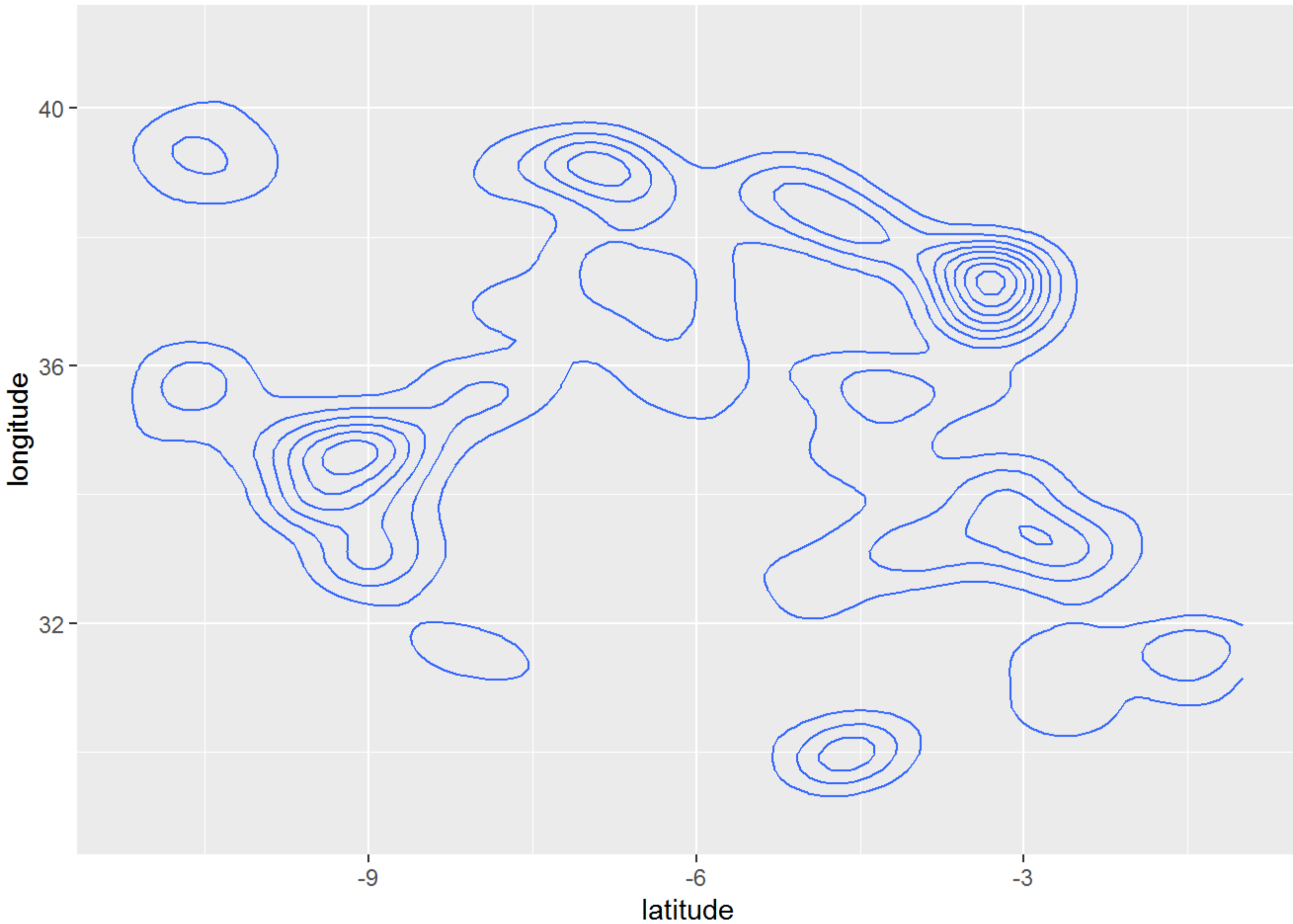
The only point we can get (from this plot) is if the density of pumps in an area is high, then the chance of having non-functional pumps is also high.

We can check the density (and then compare it to the functionality of pumps in first plot) with some other density plots:

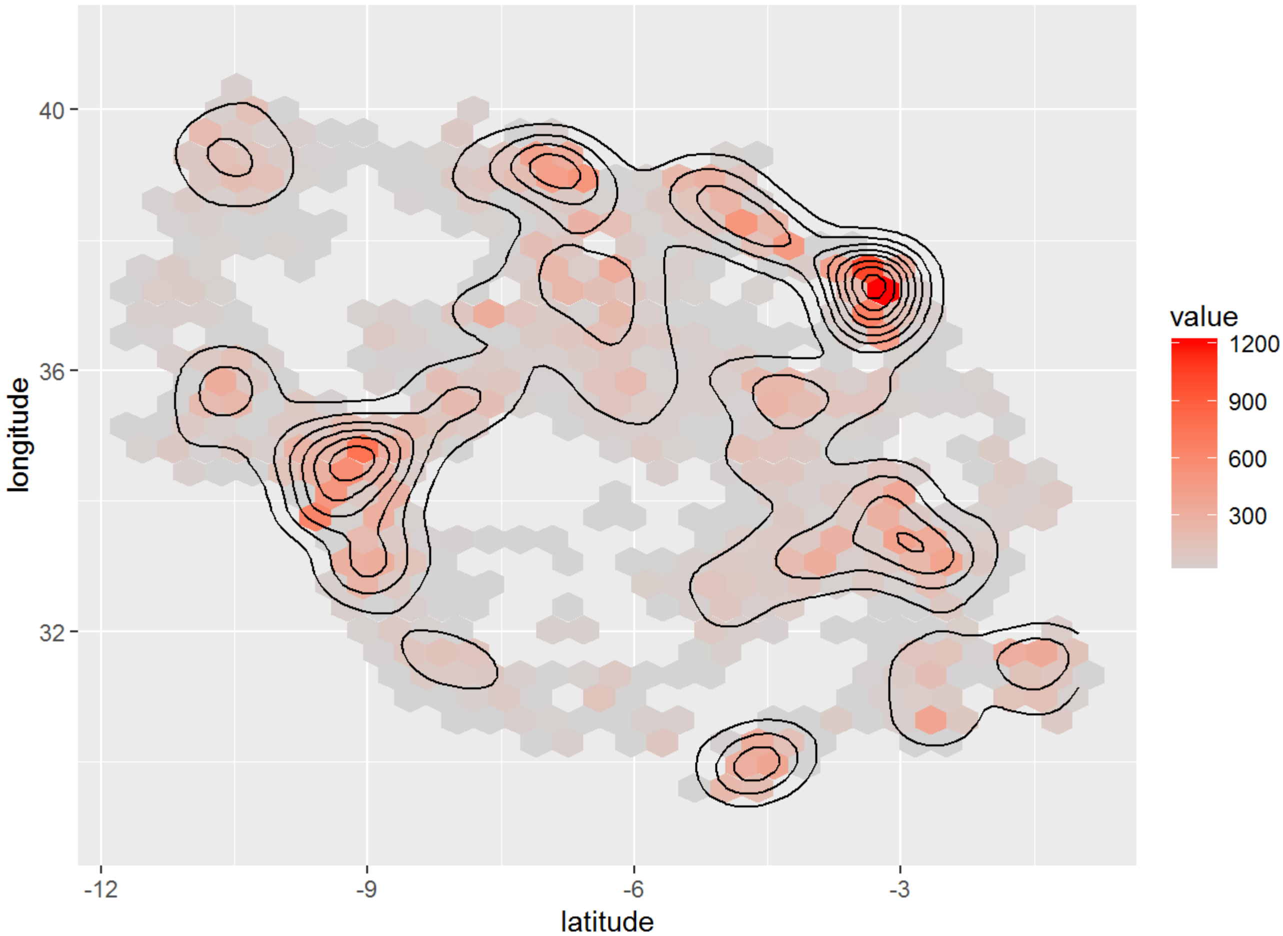
```r
library(hexbin)
drop_pump %>% ggplot(aes(x = latitude, y = longitude)) +
  geom_hex() +                        # another way for showing 2-dim density
  scale_fill_gradient(low = "lightblue", high = "darkblue") +
  scale_y_continuous(limits = c(29, 41))
```



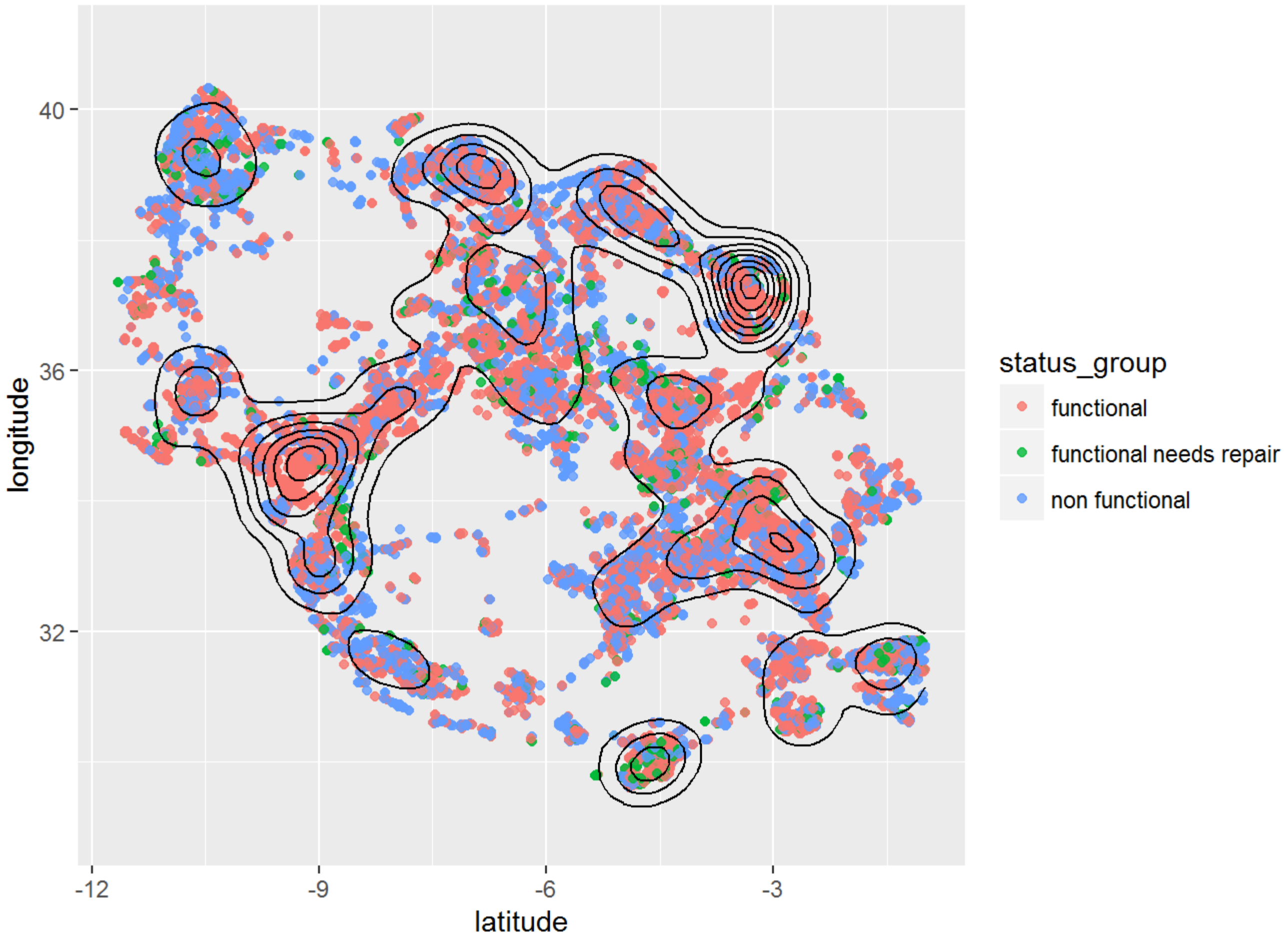

```
drop_pump %>% ggplot(aes(x = latitude, y = longitude)) +
  geom_density_2d() +              # showing a 2-dim density
  scale_y_continuous(limits = c(29, 41))
```




```
drop_pump %>% ggplot(aes(x = latitude, y = longitude)) +
  geom_hex() +
  scale_fill_gradient(low = "lightgray", high = "red") +
  geom_density2d(color = "black") +
  scale_y_continuous(limits = c(29, 41))
```




```
drop_pump %>% ggplot(aes(x = latitude, y = longitude, color = status_group)) +
  geom_point(alpha = 0.8) +
  #geom_hex() +
  scale_fill_gradient(low = "lightgray", high = "red") +
  geom_density2d(color = "black") +
  scale_y_continuous(limits = c(29, 41))
```



We can guess from the last plot, wherever density of pumps increases, the chance of being in a perfect form also increases. Maybe the reason is in areas with fewer number of pumps, using the limited number of pumps increases, and causes depreciation of those pumps.

Learning from data

Because we use linear models (regression) more on situations where we have a numeric target variable, it seems it is not suitable for our case.

But let's try it.

Linear Regression

```
drop_pump %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  lm(isFunctional ~ latitude + longitude + basin + region + region_code +
       district_code + scheme_management + permit + extraction_type +
       extraction_type_group + extraction_type_class + management +
       management_group + payment_type + water_quality + quality_group +
       quantity + source + source_type + source_class + waterpoint_type +
       waterpoint_type_group,
     data = .) %>%
  summary
```

```
##
```

```
## Call:
## lm(formula = isFunctional ~ latitude + longitude + basin + region +
##     region_code + district_code + scheme_management + permit +
##     extraction_type + extraction_type_group + extraction_type_class +
##     management + management_group + payment_type + water_quality +
##     quality_group + quantity + source + source_type + source_class +
##     waterpoint_type + waterpoint_type_group, data = .)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1202 -0.3564  0.1197  0.3186  1.4455
##
## Coefficients: (53 not defined because of singularities)
##                                             Estimate Std. Error t value
## (Intercept)                                 0.085321   0.239089   0.357
## latitude                                   -0.014484   0.005553  -2.609
## longitude                                  -0.006321   0.005588  -1.131
## basinLake Nyasa                             0.183244   0.021666   8.458
## basinLake Rukwa                            -0.042177   0.020032  -2.106
## basinLake Tanganyika                       -0.008919   0.016521  -0.540
## basinLake Victoria                         -0.004682   0.015251  -0.307
## basinPangani                               -0.029129   0.016789  -1.735
## basinRufiji                                 0.051895   0.018358   2.827
## basinRuvuma / Southern Coast                0.058575   0.028378   2.064
## basinWami / Ruvu                            0.032856   0.016730   1.964
## regionDar es Salaam                        -0.094258   0.047438  -1.987
## regionDodoma                               -0.041233   0.039350  -1.048
## regionIringa                                0.169559   0.242198   0.700
## regionKagera                                0.863387   0.269159   3.208
## regionKigoma                               -0.260388   0.051066  -5.099
## regionKilimanjaro                          -0.136927   0.037490  -3.652
## regionLindi                                 0.441275   0.197415   2.235
## regionManyara                              -0.020898   0.035161  -0.594
## regionMara                                 -0.189927   0.041982  -4.524
## regionMbeya                                -0.323551   0.048647  -6.651
## regionMorogoro                             -0.185352   0.042048  -4.408
## regionMtwara                               -0.046216   0.061429  -0.752
## regionMwanza                               -0.044260   0.040809  -1.085
## regionPwani                                 0.543850   0.193842   2.806
## regionRukwa                                -0.250611   0.049874  -5.025
## regionRuvuma                               -0.247687   0.051466  -4.813
## regionShinyanga                            -0.099982   0.038486  -2.598
## regionSingida                              -0.132859   0.038621  -3.440
## regionTabora                               -0.188935   0.102724  -1.839
## regionTanga                                 0.064085   0.081848   0.783
## region_code2                                0.022897   0.035743   0.641
## region_code3                                      NA         NA      NA
## region_code4                               -0.054817   0.075579  -0.725
## region_code5                                      NA         NA      NA
## region_code6                               -0.638071   0.190058  -3.357
## region_code7                                      NA         NA      NA
## region_code8                               -0.807744   0.193794  -4.168
## region_code9                               -0.163148   0.039954  -4.083
## region_code10                                     NA         NA      NA
```

```
## region_code11                              -0.289365   0.237934  -1.216
## region_code12                                     NA         NA       NA
## region_code13                                     NA         NA       NA
## region_code14                               0.061556   0.096430   0.638
## region_code15                                     NA         NA       NA
## region_code16                                     NA         NA       NA
## region_code17                                     NA         NA       NA
## region_code18                              -1.001940   0.265200  -3.778
## region_code19                                     NA         NA       NA
## region_code20                                     NA         NA       NA
## region_code21                                     NA         NA       NA
## region_code24                                     NA         NA       NA
## region_code60                                     NA         NA       NA
## region_code80                                     NA         NA       NA
## region_code90                               0.483550   0.195314   2.476
## region_code99                                     NA         NA       NA
## district_code1                              0.410121   0.086900   4.719
## district_code2                              0.488051   0.086944   5.613
## district_code3                              0.469865   0.086909   5.406
## district_code4                              0.489205   0.086916   5.628
## district_code5                              0.427199   0.087020   4.909
## district_code6                              0.447848   0.087167   5.138
## district_code7                              0.430640   0.087353   4.930
## district_code8                              0.373225   0.088504   4.217
## district_code13                            -0.374981   0.178321  -2.103
## district_code23                            -0.178793   0.172090  -1.039
## district_code30                             0.664646   0.089434   7.432
## district_code33                            -0.370187   0.171977  -2.153
## district_code43                            -0.103079   0.169409  -0.608
## district_code53                            -0.412263   0.168987  -2.440
## district_code60                             0.059105   0.175391   0.337
## district_code62                            -0.106977   0.218518  -0.490
## district_code63                            -0.199132   0.173622  -1.147
## district_code67                                   NA         NA       NA
## district_code80                             0.505087   0.254943   1.981
## scheme_managementOther                      0.085305   0.040309   2.116
## scheme_managementParastatal                -0.031558   0.035298  -0.894
## scheme_managementPrivate operator          -0.037581   0.028858  -1.302
## scheme_managementSWC                        0.074532   0.094513   0.789
## scheme_managementTrust                      0.218029   0.092073   2.368
## scheme_managementVWC                        0.022812   0.024923   0.915
## scheme_managementWater authority           -0.029579   0.027136  -1.090
## scheme_managementWater Board                0.118865   0.028968   4.103
## scheme_managementWUA                        0.064971   0.027633   2.351
## scheme_managementWUG                        0.036892   0.028733   1.284
## permitTrue                                  0.077116   0.005725  13.469
## extraction_typecemo                        -0.301310   0.050999  -5.908
## extraction_typeclimax                      -0.594450   0.089041  -6.676
## extraction_typegravity                     -0.212385   0.022864  -9.289
## extraction_typeindia mark ii               -0.038163   0.015975  -2.389
## extraction_typeindia mark iii              -0.185110   0.048555  -3.812
## extraction_typeksb                         -0.384579   0.025993 -14.795
## extraction_typemono                        -0.282032   0.023617 -11.942
## extraction_typenira/tanira                  0.038102   0.014666   2.598
```

```
## extraction_typeother                       -0.297055   0.020451 -14.525
## extraction_typeother - mkulima/shinyanga   -0.610296   0.290626  -2.100
## extraction_typeother - play pump           -0.376617   0.051247  -7.349
## extraction_typeother - rope pump           -0.069177   0.030723  -2.252
## extraction_typeother - swn 81              -0.116724   0.032684  -3.571
## extraction_typesubmersible                 -0.233546   0.023048 -10.133
## extraction_typeswn 80                      -0.072178   0.015419  -4.681
## extraction_typewalimi                      -0.173414   0.095472  -1.816
## extraction_typewindmill                    -0.324514   0.047996  -6.761
## extraction_type_groupgravity                      NA         NA      NA
## extraction_type_groupindia mark ii                NA         NA      NA
## extraction_type_groupindia mark iii               NA         NA      NA
## extraction_type_groupmono                         NA         NA      NA
## extraction_type_groupnira/tanira                  NA         NA      NA
## extraction_type_groupother                        NA         NA      NA
## extraction_type_groupother handpump               NA         NA      NA
## extraction_type_groupother motorpump              NA         NA      NA
## extraction_type_grouprope pump                    NA         NA      NA
## extraction_type_groupsubmersible                  NA         NA      NA
## extraction_type_groupswn 80                       NA         NA      NA
## extraction_type_groupwind-powered                 NA         NA      NA
## extraction_type_classhandpump                     NA         NA      NA
## extraction_type_classmotorpump                    NA         NA      NA
## extraction_type_classother                        NA         NA      NA
## extraction_type_classrope pump                    NA         NA      NA
## extraction_type_classsubmersible                  NA         NA      NA
## extraction_type_classwind-powered                 NA         NA      NA
## managementother                             0.258323   0.043172   5.984
## managementother - school                    0.137599   0.095446   1.442
## managementparastatal                        0.308494   0.039187   7.872
## managementprivate operator                  0.391014   0.031558  12.390
## managementtrust                             0.088617   0.090867   0.975
## managementvwc                               0.137729   0.030630   4.497
## managementwater authority                   0.214099   0.036121   5.927
## managementwater board                       0.237329   0.033241   7.140
## managementwua                               0.150994   0.034243   4.409
## managementwug                               0.220263   0.033290   6.616
## management_groupother                             NA         NA      NA
## management_groupparastatal                        NA         NA      NA
## management_groupuser-group                        NA         NA      NA
## payment_typemonthly                        -0.053553   0.009166  -5.842
## payment_typenever pay                      -0.189237   0.008670 -21.825
## payment_typeon failure                     -0.057875   0.010791  -5.363
## payment_typeother                          -0.107263   0.018020  -5.953
## payment_typeper bucket                      0.001300   0.010404   0.125
## water_qualityfluoride                       0.140307   0.045881   3.058
## water_qualityfluoride abandoned            -0.211135   0.125978  -1.676
## water_qualitymilky                          0.079225   0.033192   2.387
## water_qualitysalty                         -0.018066   0.023286  -0.776
## water_qualitysalty abandoned               -0.103130   0.037348  -2.761
## water_qualitysoft                          -0.014635   0.022143  -0.661
## quality_groupfluoride                             NA         NA      NA
## quality_groupgood                                 NA         NA      NA
## quality_groupmilky                                NA         NA      NA
```

```
## quality_groupsalty                                   NA         NA      NA
## quantityenough                                 0.573006   0.007495  76.448
## quantityinsufficient                           0.474971   0.008272  57.421
## quantityseasonal                               0.500104   0.010768  46.444
## sourcehand dtw                                -0.062776   0.026370  -2.381
## sourcelake                                    -0.113079   0.027863  -4.058
## sourcemachine dbh                              0.019805   0.020694   0.957
## sourcerainwater harvesting                     0.126886   0.023904   5.308
## sourceriver                                    0.009626   0.020255   0.475
## sourceshallow well                             0.004970   0.021486   0.231
## sourcespring                                   0.077985   0.020493   3.805
## source_typedam                                       NA         NA      NA
## source_typerainwater harvesting                      NA         NA      NA
## source_typeriver/lake                                NA         NA      NA
## source_typeshallow well                              NA         NA      NA
## source_typespring                                    NA         NA      NA
## source_classsurface                                  NA         NA      NA
## waterpoint_typecommunal standpipe             -0.132902   0.049641  -2.677
## waterpoint_typecommunal standpipe multiple -0.290065   0.049917  -5.811
## waterpoint_typedam                             0.427658   0.176332   2.425
## waterpoint_typehand pump                      -0.213969   0.052700  -4.060
## waterpoint_typeimproved spring                -0.010569   0.053737  -0.197
## waterpoint_typeother                          -0.404152   0.050578  -7.991
## waterpoint_type_groupcommunal standpipe              NA         NA      NA
## waterpoint_type_groupdam                             NA         NA      NA
## waterpoint_type_grouphand pump                       NA         NA      NA
## waterpoint_type_groupimproved spring                 NA         NA      NA
## waterpoint_type_groupother                           NA         NA      NA
##                                            Pr(>|t|)
## (Intercept)                                0.721201
## latitude                                   0.009095 **
## longitude                                  0.258014
## basinLake Nyasa                             < 2e-16 ***
## basinLake Rukwa                            0.035249 *
## basinLake Tanganyika                       0.589321
## basinLake Victoria                         0.758847
## basinPangani                               0.082743 .
## basinRufiji                                0.004703 **
## basinRuvuma / Southern Coast               0.039012 *
## basinWami / Ruvu                           0.049551 *
## regionDar es Salaam                        0.046933 *
## regionDodoma                               0.294707
## regionIringa                               0.483879
## regionKagera                               0.001339 **
## regionKigoma                               3.43e-07 ***
## regionKilimanjaro                          0.000260 ***
## regionLindi                                0.025404 *
## regionManyara                              0.552276
## regionMara                                 6.08e-06 ***
## regionMbeya                                2.95e-11 ***
## regionMorogoro                             1.05e-05 ***
## regionMtwara                               0.451847
## regionMwanza                               0.278117
## regionPwani                                0.005024 **
```

```
## regionRukwa                              5.06e-07 ***
## regionRuvuma                             1.49e-06 ***
## regionShinyanga                          0.009383 **
## regionSingida                            0.000582 ***
## regionTabora                             0.065884 .
## regionTanga                              0.433645
## region_code2                             0.521788
## region_code3                                   NA
## region_code4                             0.468276
## region_code5                                   NA
## region_code6                             0.000788 ***
## region_code7                                   NA
## region_code8                             3.08e-05 ***
## region_code9                             4.45e-05 ***
## region_code10                                  NA
## region_code11                            0.223932
## region_code12                                  NA
## region_code13                                  NA
## region_code14                            0.523252
## region_code15                                  NA
## region_code16                                  NA
## region_code17                                  NA
## region_code18                            0.000158 ***
## region_code19                                  NA
## region_code20                                  NA
## region_code21                                  NA
## region_code24                                  NA
## region_code60                                  NA
## region_code80                                  NA
## region_code90                            0.013299 *
## region_code99                                  NA
## district_code1                           2.37e-06 ***
## district_code2                           2.00e-08 ***
## district_code3                           6.46e-08 ***
## district_code4                           1.83e-08 ***
## district_code5                           9.18e-07 ***
## district_code6                           2.79e-07 ***
## district_code7                           8.26e-07 ***
## district_code8                           2.48e-05 ***
## district_code13                          0.035485 *
## district_code23                          0.298833
## district_code30                          1.09e-13 ***
## district_code33                          0.031360 *
## district_code43                          0.542883
## district_code53                          0.014707 *
## district_code60                          0.736124
## district_code62                          0.624450
## district_code63                          0.251418
## district_code67                                NA
## district_code80                          0.047578 *
## scheme_managementOther                   0.034330 *
## scheme_managementParastatal              0.371304
## scheme_managementPrivate operator        0.192833
## scheme_managementSWC                     0.430359
```

```
## scheme_managementTrust                           0.017889 *
## scheme_managementVWC                             0.360042
## scheme_managementWater authority                 0.275715
## scheme_managementWater Board                     4.08e-05 ***
## scheme_managementWUA                             0.018719 *
## scheme_managementWUG                             0.199165
## permitTrue                                        < 2e-16 ***
## extraction_typecemo                              3.49e-09 ***
## extraction_typeclimax                            2.48e-11 ***
## extraction_typegravity                            < 2e-16 ***
## extraction_typeindia mark ii                     0.016899 *
## extraction_typeindia mark iii                    0.000138 ***
## extraction_typeksb                                < 2e-16 ***
## extraction_typemono                               < 2e-16 ***
## extraction_typenira/tanira                       0.009382 **
## extraction_typeother                              < 2e-16 ***
## extraction_typeother - mkulima/shinyanga         0.035740 *
## extraction_typeother - play pump                 2.03e-13 ***
## extraction_typeother - rope pump                 0.024350 *
## extraction_typeother - swn 81                    0.000356 ***
## extraction_typesubmersible                        < 2e-16 ***
## extraction_typeswn 80                            2.86e-06 ***
## extraction_typewalimi                            0.069319 .
## extraction_typewindmill                          1.39e-11 ***
## extraction_type_groupgravity                           NA
## extraction_type_groupindia mark ii                     NA
## extraction_type_groupindia mark iii                    NA
## extraction_type_groupmono                              NA
## extraction_type_groupnira/tanira                       NA
## extraction_type_groupother                             NA
## extraction_type_groupother handpump                    NA
## extraction_type_groupother motorpump                   NA
## extraction_type_grouprope pump                         NA
## extraction_type_groupsubmersible                       NA
## extraction_type_groupswn 80                            NA
## extraction_type_groupwind-powered                      NA
## extraction_type_classhandpump                          NA
## extraction_type_classmotorpump                         NA
## extraction_type_classother                             NA
## extraction_type_classrope pump                         NA
## extraction_type_classsubmersible                       NA
## extraction_type_classwind-powered                      NA
## managementother                                  2.20e-09 ***
## managementother - school                         0.149409
## managementparastatal                             3.56e-15 ***
## managementprivate operator                        < 2e-16 ***
## managementtrust                                  0.329448
## managementvwc                                    6.93e-06 ***
## managementwater authority                        3.10e-09 ***
## managementwater board                            9.50e-13 ***
## managementwua                                    1.04e-05 ***
## managementwug                                    3.72e-11 ***
## management_groupother                                  NA
## management_groupparastatal                             NA
```

```
## management_groupuser-group                          NA
## payment_typemonthly                          5.18e-09 ***
## payment_typenever pay                         < 2e-16 ***
## payment_typeon failure                       8.20e-08 ***
## payment_typeother                            2.66e-09 ***
## payment_typeper bucket                       0.900542
## water_qualityfluoride                        0.002229 **
## water_qualityfluoride abandoned              0.093751 .
## water_qualitymilky                           0.016996 *
## water_qualitysalty                           0.437852
## water_qualitysalty abandoned                 0.005759 **
## water_qualitysoft                            0.508673
## quality_groupfluoride                              NA
## quality_groupgood                                  NA
## quality_groupmilky                                 NA
## quality_groupsalty                                 NA
## quantityenough                                < 2e-16 ***
## quantityinsufficient                          < 2e-16 ***
## quantityseasonal                              < 2e-16 ***
## sourcehand dtw                               0.017289 *
## sourcelake                                   4.95e-05 ***
## sourcemachine dbh                            0.338549
## sourcerainwater harvesting                   1.11e-07 ***
## sourceriver                                  0.634602
## sourceshallow well                           0.817072
## sourcespring                                 0.000142 ***
## source_typedam                                     NA
## source_typerainwater harvesting                    NA
## source_typeriver/lake                              NA
## source_typeshallow well                            NA
## source_typespring                                  NA
## source_classsurface                                NA
## waterpoint_typecommunal standpipe            0.007425 **
## waterpoint_typecommunal standpipe multiple 6.25e-09 ***
## waterpoint_typedam                           0.015300 *
## waterpoint_typehand pump                     4.91e-05 ***
## waterpoint_typeimproved spring               0.844082
## waterpoint_typeother                         1.37e-15 ***
## waterpoint_type_groupcommunal standpipe            NA
## waterpoint_type_groupdam                           NA
## waterpoint_type_grouphand pump                     NA
## waterpoint_type_groupimproved spring               NA
## waterpoint_type_groupother                         NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4103 on 44195 degrees of freedom
## Multiple R-squared:  0.3157, Adjusted R-squared:  0.3138
## F-statistic: 167.1 on 122 and 44195 DF,  p-value: < 2.2e-16
```

After running for the first time with linear model, we can see some factorized input variables have NA coefficients (because of **singularities**), so we can ignore them in our following models.

Singularities means two or more variables are perfectly collinear; so we can drop these features from our models.

Furthermore a small p-value indicates that it is unlikely we will observe a relationship between the predictor and response (status_group) variables due to chance. Typically, a p-value of 5% or less is a good cut-off point. But because here we have all p-values very less than 5%, so we can compare the features regarding their effects on the target variable using asterisks.

```
drop <- subset(drop_pump,
               select = -c(region_code, district_code, extraction_type_group,
                           extraction_type_class, management_group,
                           quality_group, source_type, source_class,
                           waterpoint_type_group))
str(drop)
```

```
## 'data.frame':    44318 obs. of  14 variables:
##  $ longitude        : num  36.1 37.1 39.3 33.2 36.3 ...
##  $ latitude         : num  -6.28 -3.19 -6.97 -3.85 -6.72 ...
##  $ basin            : Factor w/ 9 levels "Internal","Lake Nyasa",..: 9 6 9 1 7 9 5 6 2 6 .
..
##  $ region           : Factor w/ 21 levels "Arusha","Dar es Salaam",..: 3 7 15 18 3 3 5 21
17 1 ...
##  $ scheme_management: Factor w/ 12 levels "Company","None",..: 8 10 5 8 8 8 8 5 8 8 ...
##  $ permit           : Factor w/ 2 levels "False","True": 2 2 1 2 2 1 2 1 2 2 ...
##  $ extraction_type  : Factor w/ 18 levels "afridev","cemo",..: 8 4 15 10 10 8 8 15 4 4 ...
##  $ management       : Factor w/ 11 levels "company","other",..: 7 9 5 7 7 5 7 5 7 7 ...
##  $ payment_type     : Factor w/ 6 levels "annually","monthly",..: 6 6 6 3 3 6 3 6 4 3 ...
##  $ water_quality    : Factor w/ 7 levels "coloured","fluoride",..: 7 7 7 7 4 7 7 5 7 7 ...
##  $ quantity         : Factor w/ 4 levels "dry","enough",..: 3 2 2 4 3 1 3 3 1 2 ...
##  $ source           : Factor w/ 9 levels "dam","hand dtw",..: 4 9 4 8 9 4 4 4 9 9 ...
##  $ waterpoint_type  : Factor w/ 7 levels "cattle trough",..: 3 2 3 7 6 3 3 2 2 2 ...
##  $ status_group     : Factor w/ 3 levels "functional","functional needs repair",..: 1 1 1
3 1 3 3 3 3 1 ...
```

So we have just 13 variables: **latitude**, **longitude**, **basin**, **region**, **scheme_management**, **permit**, **extraction_type**, **management**, **payment_type**, **water_quality**, **quantity**, **source**, **waterpoint_type** to implement learning algorithms.

Evaluating regression models

Before anything for comparing the linear regression models, we have to have two separate samples: 'training' for learning from it, and 'test' to check how degree we can generalize the rules we have learned for prediction new records.

So let's divide up our records to these two sample (70% training and 30% test):

```
smp_size <- floor(0.7 * nrow(drop))

set.seed(123456)
train_ind <- sample(seq_len(nrow(drop)), size = smp_size)

train <- drop[train_ind, ]
test <- drop[-train_ind, ]
```

We have some variables with very low level of chance to drop them (p-value). We think they are good candidates for increasing the degree of them to test if we can find a better (still linear) model or not.

Because having more degrees in factor variables, does not make sense, we can play only with the degrees of two numeric variables we have ('latitude' and 'longitude'):

```
line <-
  train %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  lm(isFunctional ~ latitude + longitude + basin + region + scheme_management +
       permit + extraction_type + management + payment_type + water_quality +
       quantity + source + waterpoint_type,
     data = .)

poly <-
  train %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  lm(isFunctional ~ latitude + I(latitude^2) + longitude + I(longitude^2) +
       basin + region + scheme_management + permit + extraction_type +
       management + payment_type + water_quality + quantity + source +
       waterpoint_type,
     data = .)
```

Variable selection using automatic methods

Yet, we have thrown out some variables based on their p-values. However, we can check if we can throw out more variables or not. We have to make a balance between the number of variables in our model (complexity - increasing variance - overfitting) and the power of prediction of the model (increasing bias - underfitting). So using the 'leaps' package we want to check which variables have more chance to be effective on the power of prediction.

We are using some methods for selecting more effective variables: 1. Best subset of a particular size: At a very first step, we assume that we want to drop about half of our thirteen variables, So we want to consider just six or seven variables. Considering a suitable size of variables make it easier to interpret the effect of our variables on the model.

```
library("leaps")
reg_ex <- regsubsets(ifelse(status_group == "functional", 1, 0) ~ latitude +
                       longitude + basin + region + scheme_management + permit +
                       extraction_type,
                     data = train, nvmax = 7, really.big = TRUE)
```

```
## Reordering variables and trying again:
```

```
summary(reg_ex)
```

```
## Subset selection object
## Call: regsubsets.formula(ifelse(status_group == "functional", 1, 0) ~
##     latitude + longitude + basin + region + scheme_management +
##         permit + extraction_type, data = train, nvmax = 7, really.big = TRUE)
## 59 Variables  (and intercept)
##                                           Forced in Forced out
## latitude                                      FALSE      FALSE
## longitude                                     FALSE      FALSE
## basinLake Nyasa                               FALSE      FALSE
## basinLake Rukwa                               FALSE      FALSE
## basinLake Tanganyika                          FALSE      FALSE
```

```
## basinLake Victoria                              FALSE      FALSE
## basinPangani                                    FALSE      FALSE
## basinRufiji                                     FALSE      FALSE
## basinRuvuma / Southern Coast                    FALSE      FALSE
## basinWami / Ruvu                                FALSE      FALSE
## regionDar es Salaam                             FALSE      FALSE
## regionDodoma                                    FALSE      FALSE
## regionIringa                                    FALSE      FALSE
## regionKagera                                    FALSE      FALSE
## regionKigoma                                    FALSE      FALSE
## regionKilimanjaro                               FALSE      FALSE
## regionLindi                                     FALSE      FALSE
## regionManyara                                   FALSE      FALSE
## regionMara                                      FALSE      FALSE
## regionMbeya                                     FALSE      FALSE
## regionMorogoro                                  FALSE      FALSE
## regionMtwara                                    FALSE      FALSE
## regionMwanza                                    FALSE      FALSE
## regionPwani                                     FALSE      FALSE
## regionRukwa                                     FALSE      FALSE
## regionRuvuma                                    FALSE      FALSE
## regionShinyanga                                 FALSE      FALSE
## regionSingida                                   FALSE      FALSE
## regionTabora                                    FALSE      FALSE
## regionTanga                                     FALSE      FALSE
## scheme_managementOther                          FALSE      FALSE
## scheme_managementParastatal                     FALSE      FALSE
## scheme_managementPrivate operator               FALSE      FALSE
## scheme_managementSWC                            FALSE      FALSE
## scheme_managementTrust                          FALSE      FALSE
## scheme_managementVWC                            FALSE      FALSE
## scheme_managementWater authority                FALSE      FALSE
## scheme_managementWater Board                    FALSE      FALSE
## scheme_managementWUA                            FALSE      FALSE
## scheme_managementWUG                            FALSE      FALSE
## permitTrue                                      FALSE      FALSE
## extraction_typecemo                             FALSE      FALSE
## extraction_typeclimax                           FALSE      FALSE
## extraction_typegravity                          FALSE      FALSE
## extraction_typeindia mark ii                    FALSE      FALSE
## extraction_typeindia mark iii                   FALSE      FALSE
## extraction_typeksb                              FALSE      FALSE
## extraction_typemono                             FALSE      FALSE
## extraction_typenira/tanira                      FALSE      FALSE
## extraction_typeother                            FALSE      FALSE
## extraction_typeother - mkulima/shinyanga        FALSE      FALSE
## extraction_typeother - play pump                FALSE      FALSE
## extraction_typeother - rope pump                FALSE      FALSE
## extraction_typeother - swn 81                   FALSE      FALSE
## extraction_typesubmersible                      FALSE      FALSE
## extraction_typeswn 80                           FALSE      FALSE
## extraction_typewalimi                           FALSE      FALSE
## extraction_typewindmill                         FALSE      FALSE
## scheme_managementNone                           FALSE      FALSE
```

```
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          latitude longitude basinLake Nyasa basinLake Rukwa
## 1  ( 1 ) " "      " "       " "             " "
## 2  ( 1 ) " "      " "       " "             " "
## 3  ( 1 ) " "      " "       " "             " "
## 4  ( 1 ) " "      " "       " "             " "
## 5  ( 1 ) " "      " "       " "             " "
## 6  ( 1 ) " "      " "       " "             " "
## 7  ( 1 ) " "      " "       " "             " "
## 8  ( 1 ) " "      " "       " "             " "
##          basinLake Tanganyika basinLake Victoria basinPangani basinRufiji
## 1  ( 1 ) " "                  " "                " "          " "
## 2  ( 1 ) " "                  " "                " "          " "
## 3  ( 1 ) " "                  " "                " "          " "
## 4  ( 1 ) " "                  " "                " "          " "
## 5  ( 1 ) " "                  " "                " "          " "
## 6  ( 1 ) " "                  " "                " "          " "
## 7  ( 1 ) " "                  " "                " "          " "
## 8  ( 1 ) " "                  " "                " "          " "
##          basinRuvuma / Southern Coast basinWami / Ruvu regionDar es Salaam
## 1  ( 1 ) " "                          " "              " "
## 2  ( 1 ) " "                          " "              " "
## 3  ( 1 ) " "                          " "              " "
## 4  ( 1 ) " "                          " "              " "
## 5  ( 1 ) " "                          " "              " "
## 6  ( 1 ) " "                          " "              " "
## 7  ( 1 ) " "                          " "              " "
## 8  ( 1 ) " "                          " "              " "
##          regionDodoma regionIringa regionKagera regionKigoma
## 1  ( 1 ) " "          " "          " "          " "
## 2  ( 1 ) " "          "*"          " "          " "
## 3  ( 1 ) " "          "*"          " "          " "
## 4  ( 1 ) " "          "*"          " "          " "
## 5  ( 1 ) " "          "*"          " "          " "
## 6  ( 1 ) " "          "*"          " "          " "
## 7  ( 1 ) " "          "*"          " "          " "
## 8  ( 1 ) " "          "*"          " "          " "
##          regionKilimanjaro regionLindi regionManyara regionMara
## 1  ( 1 ) " "               " "         " "           " "
## 2  ( 1 ) " "               " "         " "           " "
## 3  ( 1 ) " "               " "         " "           " "
## 4  ( 1 ) " "               " "         " "           " "
## 5  ( 1 ) " "               " "         " "           " "
## 6  ( 1 ) " "               " "         " "           " "
## 7  ( 1 ) " "               " "         " "           " "
## 8  ( 1 ) " "               " "         " "           " "
##          regionMbeya regionMorogoro regionMtwara regionMwanza regionPwani
## 1  ( 1 ) " "         " "            " "          " "          " "
## 2  ( 1 ) " "         " "            " "          " "          " "
## 3  ( 1 ) " "         " "            " "          " "          " "
## 4  ( 1 ) " "         " "            " "          " "          " "
## 5  ( 1 ) "*"         " "            " "          " "          " "
## 6  ( 1 ) "*"         " "            "*"          " "          " "
```

```
## 7  ( 1 ) "*"         " "           "*"          " "          " "
## 8  ( 1 ) "*"         " "           "*"          " "          " "
##          regionRukwa regionRuvuma regionShinyanga regionSingida
## 1  ( 1 ) " "         " "          " "             " "
## 2  ( 1 ) " "         " "          " "             " "
## 3  ( 1 ) " "         " "          " "             " "
## 4  ( 1 ) " "         " "          " "             " "
## 5  ( 1 ) " "         " "          " "             " "
## 6  ( 1 ) " "         " "          " "             " "
## 7  ( 1 ) " "         " "          " "             " "
## 8  ( 1 ) " "         " "          " "             " "
##          regionTabora regionTanga scheme_managementNone
## 1  ( 1 ) " "          " "         " "
## 2  ( 1 ) " "          " "         " "
## 3  ( 1 ) " "          " "         " "
## 4  ( 1 ) " "          " "         " "
## 5  ( 1 ) " "          " "         " "
## 6  ( 1 ) " "          " "         " "
## 7  ( 1 ) " "          " "         " "
## 8  ( 1 ) " "          " "         " "
##          scheme_managementOther scheme_managementParastatal
## 1  ( 1 ) " "                    " "
## 2  ( 1 ) " "                    " "
## 3  ( 1 ) " "                    " "
## 4  ( 1 ) " "                    " "
## 5  ( 1 ) " "                    " "
## 6  ( 1 ) " "                    " "
## 7  ( 1 ) " "                    " "
## 8  ( 1 ) " "                    " "
##          scheme_managementPrivate operator scheme_managementSWC
## 1  ( 1 ) " "                                " "
## 2  ( 1 ) " "                                " "
## 3  ( 1 ) " "                                " "
## 4  ( 1 ) " "                                " "
## 5  ( 1 ) " "                                " "
## 6  ( 1 ) " "                                " "
## 7  ( 1 ) " "                                " "
## 8  ( 1 ) "*"                                " "
##          scheme_managementTrust scheme_managementVWC
## 1  ( 1 ) " "                    " "
## 2  ( 1 ) " "                    " "
## 3  ( 1 ) " "                    " "
## 4  ( 1 ) " "                    " "
## 5  ( 1 ) " "                    " "
## 6  ( 1 ) " "                    " "
## 7  ( 1 ) " "                    " "
## 8  ( 1 ) " "                    " "
##          scheme_managementWater authority scheme_managementWater Board
## 1  ( 1 ) " "                              " "
## 2  ( 1 ) " "                              " "
## 3  ( 1 ) " "                              "*"
## 4  ( 1 ) " "                              "*"
## 5  ( 1 ) " "                              "*"
## 6  ( 1 ) " "                              "*"
```

```
## 7  ( 1 ) " "                            "*"
## 8  ( 1 ) " "                            "*"
##          scheme_managementWUA scheme_managementWUG permitTrue
## 1  ( 1 ) " "                  " "                  " "
## 2  ( 1 ) " "                  " "                  " "
## 3  ( 1 ) " "                  " "                  " "
## 4  ( 1 ) " "                  " "                  " "
## 5  ( 1 ) " "                  " "                  " "
## 6  ( 1 ) " "                  " "                  " "
## 7  ( 1 ) " "                  " "                  " "
## 8  ( 1 ) " "                  " "                  " "
##          extraction_typecemo extraction_typeclimax extraction_typegravity
## 1  ( 1 ) " "                 " "                   " "
## 2  ( 1 ) " "                 " "                   " "
## 3  ( 1 ) " "                 " "                   " "
## 4  ( 1 ) " "                 " "                   " "
## 5  ( 1 ) " "                 " "                   " "
## 6  ( 1 ) " "                 " "                   " "
## 7  ( 1 ) " "                 " "                   " "
## 8  ( 1 ) " "                 " "                   " "
##          extraction_typeindia mark ii extraction_typeindia mark iii
## 1  ( 1 ) " "                          " "
## 2  ( 1 ) " "                          " "
## 3  ( 1 ) " "                          " "
## 4  ( 1 ) " "                          " "
## 5  ( 1 ) " "                          " "
## 6  ( 1 ) " "                          " "
## 7  ( 1 ) " "                          " "
## 8  ( 1 ) " "                          " "
##          extraction_typeksb extraction_typemono extraction_typenira/tanira
## 1  ( 1 ) " "                " "                 " "
## 2  ( 1 ) " "                " "                 " "
## 3  ( 1 ) " "                " "                 " "
## 4  ( 1 ) " "                " "                 "*"
## 5  ( 1 ) " "                " "                 "*"
## 6  ( 1 ) " "                " "                 "*"
## 7  ( 1 ) " "                "*"                 "*"
## 8  ( 1 ) " "                "*"                 "*"
##          extraction_typeother extraction_typeother - mkulima/shinyanga
## 1  ( 1 ) "*"                  " "
## 2  ( 1 ) "*"                  " "
## 3  ( 1 ) "*"                  " "
## 4  ( 1 ) "*"                  " "
## 5  ( 1 ) "*"                  " "
## 6  ( 1 ) "*"                  " "
## 7  ( 1 ) "*"                  " "
## 8  ( 1 ) "*"                  " "
##          extraction_typeother - play pump extraction_typeother - rope pump
## 1  ( 1 ) " "                              " "
## 2  ( 1 ) " "                              " "
## 3  ( 1 ) " "                              " "
## 4  ( 1 ) " "                              " "
## 5  ( 1 ) " "                              " "
## 6  ( 1 ) " "                              " "
```

```
## 7  ( 1 ) " "                              " "
## 8  ( 1 ) " "                              " "
##          extraction_typeother - swn 81 extraction_typesubmersible
## 1  ( 1 ) " "                           " "
## 2  ( 1 ) " "                           " "
## 3  ( 1 ) " "                           " "
## 4  ( 1 ) " "                           " "
## 5  ( 1 ) " "                           " "
## 6  ( 1 ) " "                           " "
## 7  ( 1 ) " "                           " "
## 8  ( 1 ) " "                           " "
##          extraction_typeswn 80 extraction_typewalimi
## 1  ( 1 ) " "                   " "
## 2  ( 1 ) " "                   " "
## 3  ( 1 ) " "                   " "
## 4  ( 1 ) " "                   " "
## 5  ( 1 ) " "                   " "
## 6  ( 1 ) " "                   " "
## 7  ( 1 ) " "                   " "
## 8  ( 1 ) " "                   " "
##          extraction_typewindmill
## 1  ( 1 ) " "
## 2  ( 1 ) " "
## 3  ( 1 ) " "
## 4  ( 1 ) " "
## 5  ( 1 ) " "
## 6  ( 1 ) " "
## 7  ( 1 ) " "
## 8  ( 1 ) " "
```

So the summary shows we have to keep: ‘**region**’, ‘**scheme_management**’, and ‘**extraction_type**’ from the first seven variables.

```
reg_ex2 <- regsubsets(ifelse(status_group == "functional", 1, 0) ~ management +
                        payment_type + water_quality + quantity + source +
                        waterpoint_type,
                     data = train, nvmax = 6)
```

```
## Reordering variables and trying again:
```

The same process for the second half shows we have to keep: ‘**payment_type**’, ‘**quantity**’, and ‘**waterpoint_type**’ from the second six variables.

Therefore, we have selected ‘**region**’, ‘**scheme_management**’, ‘**extraction_type**’, ‘**payment_type**’, ‘**quantity**’, and ‘**waterpoint_type**’ for ‘best subset of a particular size’ forward.

```
reg_ex_back <- regsubsets(ifelse(status_group == "functional", 1, 0) ~
                            latitude + longitude + basin + region +
                            scheme_management + permit + extraction_type +
                            management + payment_type + water_quality +
                       quantity + source + waterpoint_type,
                     data = train, method = "backward")
```

```
## Reordering variables and trying again:
```

Also the same process (this time with ‘backward’ method enforces us to select ‘**payment_type**’, ‘**quantity**’, ‘**source**’ and ‘**waterpoint_type**’ for ‘best subset of a particular size’ backward.

Measure selection

We usually use root of mean squared error for a measure of how well the models are fitting, but because more variables we have here are categorical features, it does not mean too much here: (and this has to be changed to a prediction next)

```
rmse <- function(x,t) sqrt(mean(sum((t - x)^2)))
rmse(predict(line, test),
     test %>% mutate(isFunctional = ifelse(status_group == "functional", 1, 0)))
```

```
## [1] NA
```

```
rmse(predict(poly, test),
     test %>% mutate(isFunctional = ifelse(status_group == "functional", 1, 0)))
```

```
## [1] NA
```

Here we have two null values. In addition to the above mentioned reason (categorical features), because our problem is a kind of a classification problem, we think ‘rmse’ cannot be a good measure for assessing the power of prediction in applying linear regression for our problem.

We have obtained our coefficients using the line that makes the minimum distance between real target variables and predicted variables in each (data) point. But now for every new data point, we have to identify if the point belongs to ‘functional’ pumps or ‘non-functional’ pumps.

Here we have a classification problem, so we change the target variable in order to have a (-1, +1) variable, and after that we want to focus on the sign of our target variable e.g. (-1) for non-functional, and (+1) for functional pumps.

```
line <-
  train %>%
  mutate(pump_sign = ifelse(status_group == "functional", 1, -1)) %>%
  lm(pump_sign ~ latitude + longitude + basin + region + scheme_management +
       permit + extraction_type + management + payment_type + water_quality +
       quantity + source + waterpoint_type, data = .)
poly <-
  train %>%
  mutate(pump_sign = ifelse(status_group == "functional", 1, -1)) %>%
  lm(pump_sign ~ latitude + I(latitude^2) + longitude + I(longitude^2) +
       basin + region + scheme_management + permit + extraction_type +
       management + payment_type + water_quality + quantity + source +
       waterpoint_type, data = .)
ex <-
  train %>%
  mutate(pump_sign = ifelse(status_group == "functional", 1, -1)) %>%
  lm(pump_sign ~ region + scheme_management + extraction_type + payment_type +
       quantity + waterpoint_type, data = .)
ex_back <-
```

```
  train %>%
  mutate(pump_sign = ifelse(status_group == "functional", 1, -1)) %>%
  lm(pump_sign ~ payment_type + quantity + source + waterpoint_type, data = .)

reg_classify <- function(target_var) ifelse(target_var > 0, 1, -1)
classified_line <- reg_classify(predict(line, test))
classified_poly <- reg_classify(predict(poly, test))
classified_ex <- reg_classify(predict(ex, test))
classified_ex_back <- reg_classify(predict(ex_back, test))
```

Confusion Matrix

We use 'confusion matrix' as a very initiative tool to measuring power of prediction of our linear models:

```
formatted_data <- test %>%
  mutate(pump_sign = ifelse(status_group == "functional", 1, -1))
```

- **linear model**

```
(confusion_matrix <- table(formatted_data$pump_sign, classified_line,
                           dnn=c("Data", "Predictions")))
```

```
##     Predictions
## Data   -1    1
##   -1 3417 2341
##   1   956 6582
```

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7520307
```

- **Polynomial model**

```
(confusion_matrix <- table(formatted_data$pump_sign, classified_poly,
                           dnn=c("Data", "Predictions")))
```

```
##     Predictions
## Data   -1    1
##   -1 3449 2309
##   1   951 6587
```

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7548135
```

The 'accuracy rate' has not been changed very much in comparison with 'linear model'.

- **'best subset of a particular size' forward model**

```r
(confusion_matrix <- table(formatted_data$pump_sign, classified_ex,
                           dnn=c("Data", "Predictions")))
```

```
##     Predictions
## Data   -1    1
##   -1 3304 2454
##   1   944 6594
```

```r
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7444344
```

- **‘best subset of a particular size’ backward model**

```r
(confusion_matrix <- table(formatted_data$pump_sign, classified_ex_back,
                           dnn=c("Data", "Predictions")))
```

```
##     Predictions
## Data   -1    1
##   -1 2672 3086
##   1   568 6970
```

```r
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7251805
```

Actually we can see we have done great with our ‘variable selection methods’ especially with ‘forward’ one. We have just 6 features in ‘forward’ model, but our ‘accuracy rate’ is approximately the same with our linear model with 13 features.

Because we have a target variable that we can classify it into two main groups (functional, non-functional), so the logistic regression model can be a suitable model for us.

Logistic Regression

```r
learned <-
  train %>%
    mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
    glm(isFunctional ~ latitude + longitude + basin + region +
          scheme_management + permit + extraction_type + management +
          payment_type + water_quality + quantity + source + waterpoint_type,
     data = .,
     family = "binomial")
learned_forward <-
  train %>%
    mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
    glm(isFunctional ~ region + scheme_management + extraction_type +
          payment_type + quantity + waterpoint_type,
     data = .,
     family = "binomial")
```

Evaluating classification models

We use a function to produce 1 when the probability of working well is more than 50 percent. This function would be a measure for evaluating how well the logistic models can predict the functionality of the pumps.

```
classify <- function(probability) ifelse(probability < 0.5, 0, 1)
classified_pump <- classify(predict(learned, test))
classified_pump_forward <- classify(predict(learned_forward, test))
```

Confusion Matrix

```
formatted_data <- test %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0))

(confusion_matrix <- table(formatted_data$isFunctional, classified_pump,
                           dnn=c("Data", "Predictions")))
```

```
##      Predictions
## Data    0    1
##    0 4372 1386
##    1 2169 5369
```

```
(confusion_matrix_forward <- table(formatted_data$isFunctional,
                                   classified_pump_forward,
                                   dnn=c("Data", "Predictions")))
```

```
##      Predictions
## Data    0    1
##    0 4175 1583
##    1 2184 5354
```

Accuracy

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7326264
```

```
(accuracy <- sum(diag(confusion_matrix_forward)) / sum(confusion_matrix_forward))
```

```
## [1] 0.7166817
```

Here also we can see, the ‘forward’ model performs very well, even though our logistic models cannot perform as well as linear models regarding the prediction power.

We want to check if our measures are performing well at all or not. So we have to know about the proportion of ‘functional’ pumps (in whole of data).

```
tbl <- table(formatted_data$isFunctional)
```

```
tbl[1] / sum(tbl)
```

```
##         0
## 0.4330626
```

Because just 45% of all records are ‘functional’ pumps, so if we guess approximately in a random manner, then we cannot reach the 72% of guessing in our model.

Let’s try a completely random sample from our data (using sample() function):

```
accuracy <- function(confusion_matrix)
  sum(diag(confusion_matrix))/sum(confusion_matrix)
```

We are going to select our sample accuracy 8 times (for instance):

```
replicate(8, accuracy(table(formatted_data$isFunctional,
                            sample(formatted_data$isFunctional))))
```

```
## [1] 0.5024067 0.5051143 0.5111312 0.5108303 0.5169976 0.5004513 0.5097774
## [8] 0.5118833
```

With the above percents (around 50%), we see our model performs much better.

Sensitivity and Specificity

- ***Specificity***: how often the model predicts a negative case correctly.

```
(specificity <-
  confusion_matrix[1,1] / (confusion_matrix[1,1] + confusion_matrix[1,2]))
```

```
## [1] 0.7592914
```

- ***sensitivity***: how often the model predicts a positive case correctly.

```
(sensitivity <-
   confusion_matrix[2,2] / (confusion_matrix[2,1] + confusion_matrix[2,2]))
```

```
## [1] 0.7122579
```

Again we are going to check how well would be our guess (of sensitivity and specificity) in comparison with random permutation.

First we define two function for ‘sensitivity’ and ‘specificity’ (We have defined ‘accuracy’ function already):

```
specificity <- function(confusion_matrix)
  confusion_matrix[1,1]/(confusion_matrix[1,1]+confusion_matrix[1,2])
sensitivity <- function(confusion_matrix)
  confusion_matrix[2,2]/(confusion_matrix[2,1]+confusion_matrix[2,2])
```

Then, we define a function to calculate all three measures:

```
prediction_summary <- function(confusion_matrix)
  c("accuracy"    = accuracy(confusion_matrix),
    "specificity" = specificity(confusion_matrix),
    "sensitivity" = sensitivity(confusion_matrix))
```

After that, we use ‘sample’ function to generate random samples of data points:

```
random_prediction_summary <- function()
  prediction_summary(table(formatted_data$isFunctional,
                           sample(formatted_data$isFunctional)))
```

Repeating (selecting) 3 (for instance) sample:

```
replicate(3, random_prediction_summary())
```

```
##                  [,1]      [,2]      [,3]
## accuracy    0.5151925 0.5091757 0.5106799
## specificity 0.4402570 0.4333102 0.4350469
## sensitivity 0.5724330 0.5671266 0.5684532
```

So in order: 72%, 75%, and 71% for accuracy, specificity, and sensitivity of our model are much better than these measures in randomly selected samples.

Other Measures

- ***False omission rate*** (false negatives divided by all predicted negatives):

```
confusion_matrix[2,1] / sum(confusion_matrix[,1])
```

```
## [1] 0.3316007
```

- ***Negative predictive value*** (true negatives divided by all predicted negatives):

```
confusion_matrix[1,1] / sum(confusion_matrix[,1])
```

```
## [1] 0.6683993
```

- ***Positive predictive value***

```
confusion_matrix[2,2] / sum(confusion_matrix[,2])
```

```
## [1] 0.7948187
```

- ***False discovery rate***

```
confusion_matrix[1,2] / sum(confusion_matrix[,2])
```

```
## [1] 0.2051813
```

Interpretation of FDR (False Discovery Rate)

It means significance threshold in classical hypothesis testing, e.g. our guess that the pump is functional is wrong in 21% of cases.

Cross-Validation

We can use cross-validation method to divide up our data to k parts and then use each part as the test sample and other k - 1 parts as training samples. The algorithm is useful for finding the coefficients and because we would like to calculate the prediction power of our models, we will use it in the **Bayesian Networks' section.

Decision Trees

'Decision Trees' are used to divide the features based on their values that may be less or more than a threshold.

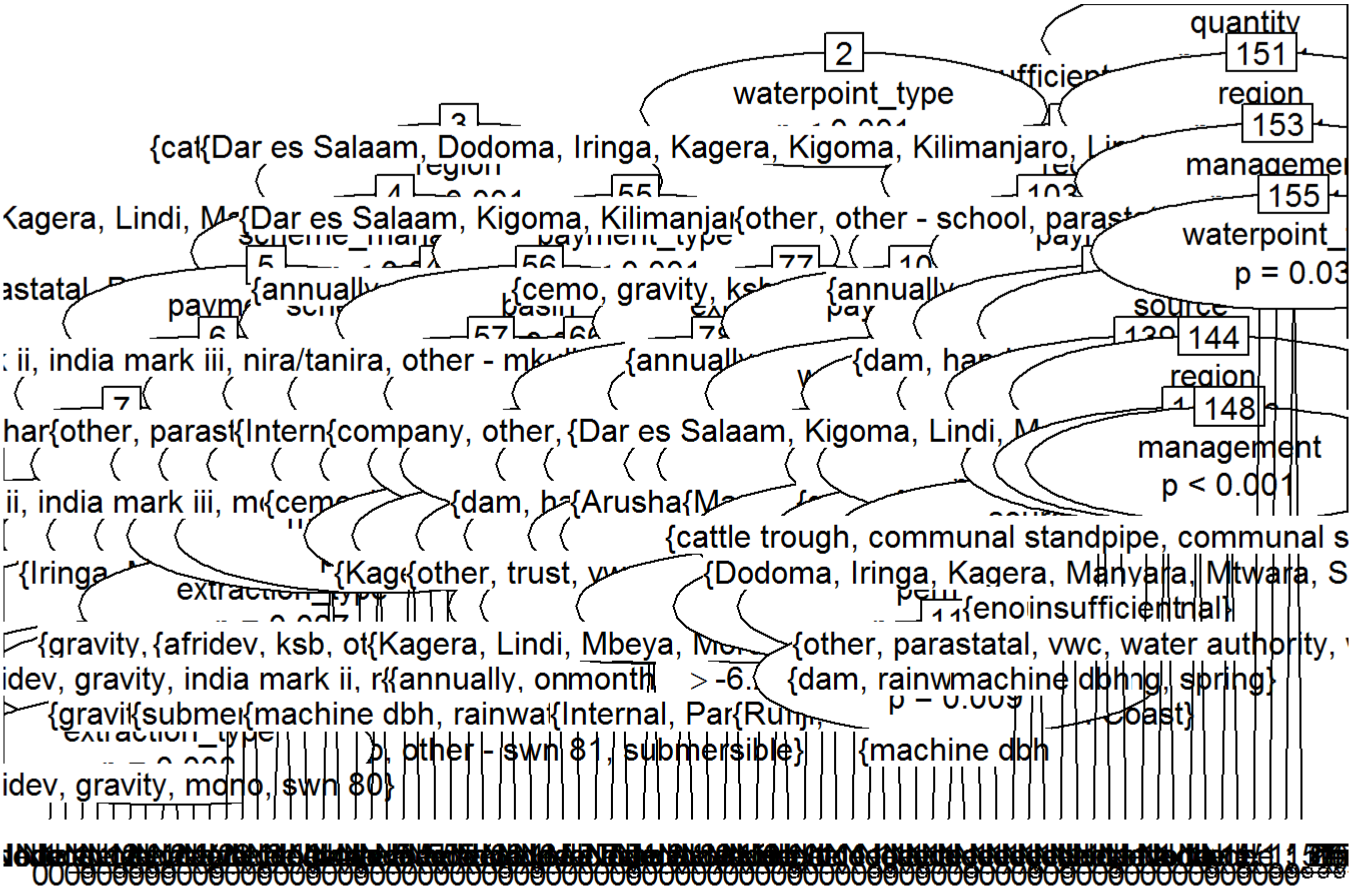
```
library(party)
decision_model <- train %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  ctree(isFunctional ~ latitude + longitude + basin + region +
          scheme_management + permit + extraction_type + management +
          payment_type + water_quality + quantity + source + waterpoint_type,
        data = .)

tree_pump <- classify(predict(decision_model, test))

(confusion_matrix <- table(formatted_data$isFunctional, tree_pump,
                           dnn=c("Data", "Predictions")))
```

```
##      Predictions
## Data    0    1
##    0 3798 1960
##    1 1045 6493
```

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7739922
```

```
formatted_data %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  ctree(isFunctional ~ latitude + longitude + basin + region +
          scheme_management + permit + extraction_type + management +
          payment_type + water_quality + quantity + source + waterpoint_type,
        data = .) %>% plot
```



The plot actually does not show much, but at least we know our decision tree model performs well.

And for ‘forward’ model:

```r
decision_model_forward <- train %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  ctree(isFunctional ~ region + scheme_management + extraction_type +
          payment_type + quantity + waterpoint_type,
        data = .)

tree_pump <- classify(predict(decision_model_forward, test))

(confusion_matrix <- table(formatted_data$isFunctional, tree_pump,
                           dnn=c("Data", "Predictions")))
```

```
##      Predictions
## Data    0    1
##    0 3447 2311
##    1  784 6754
```

```r
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7672232
```

Also here the prediction power of both models approximately are the same.

Because the 'sign()' function we had in linear regression, had a better perfromance in comparison with logistic ones; let's check 'decision tree' algorithm using the 'sign()' function:

```
library(party)
decision_model <- train %>%
  mutate(pump_sign = ifelse(status_group == "functional", 1, -1)) %>%
  ctree(pump_sign ~ latitude + longitude + basin + region +
          scheme_management + permit + extraction_type + management +
          payment_type + water_quality + quantity + source + waterpoint_type,
        data = .)

tree_pump <- reg_classify(predict(decision_model, test))
(confusion_matrix <- table(formatted_data$isFunctional, tree_pump,
                           dnn=c("Data", "Predictions")))
```

```
##     Predictions
## Data   -1    1
##    0 3800 1958
##    1 1050 6488
```

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7737665
```

So when we are using the 'sign()' function to classify our target variable, we have a little worse 'accuracy rate' than when we are using our categorical classify in decision trees - but we can ignore it.

Random Forests

'Random Forests' generalize decision trees by implementing them several times, and then combining them to use the best part of classification each decision tree has in its applying. So we expect we do better in terms of performance of prediction.

When we want to use 'Random Forests' we do not have to have missing values.

Also it needs much time to be done, so we use 'forward' selection of features model and hope it works as well as the model with 13 features.

```
library(randomForest)
forest_model <- train %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  randomForest(isFunctional ~ region + scheme_management + extraction_type +
                 payment_type + quantity + waterpoint_type,
               data = .)

forest_pump <- classify(predict(forest_model, test))

(confusion_matrix <- table(formatted_data$isFunctional, forest_pump,
                           dnn=c("Data", "Predictions")))
```

```
##      Predictions
## Data    0    1
##    0 3709 2049
##    1  829 6709
```

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7835439
```

Because the ‘Random Forest’ needs time to be done, we are not going to do it now. Also we have some null values and the model does not work with them.

Neural Networks

‘Neural Networks’ implement a model using the layers of nodes (neurals) that in each layer we can have a non-linear combination of the previous layer, and therefore the power of ‘Neural Networks’ in prediction is considerable.

```
library(nnet)
neural_model <- train %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  nnet(isFunctional ~ latitude + longitude + basin + region + scheme_management +
          permit + extraction_type + management + payment_type + water_quality +
          quantity + source + waterpoint_type,
        data = ., size = 5)
```

```
## # weights:  496
## initial  value 8649.687634
## iter  10 value 7213.765738
## iter  20 value 5878.501664
## iter  30 value 5519.910591
## iter  40 value 5289.060523
## iter  50 value 5186.675129
## iter  60 value 5139.665038
## iter  70 value 5131.691384
## iter  80 value 5098.654441
## iter  90 value 4991.194935
## iter 100 value 4932.966363
## final  value 4932.966363
## stopped after 100 iterations
```

```
nueral_pump <- classify(predict(neural_model, test))

(confusion_matrix <- table(formatted_data$isFunctional, nueral_pump,
                           dnn=c("Data", "Predictions")))
```

```
##      Predictions
## Data    0    1
##    0 3619 2139
##    1  928 6610
```

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7693291
```

The ‘size’ parameters identifies the number of hidden layers in ‘neural network’.

Support Vector Machines

‘Support Vector Machines’ is a discriminative classifier formally defined by a separating hyperplane. They best do the classification for new data points, even though they may have not classify all data points correctly. It is because they do not tend to have over-fitting.

```
library(kernlab)
svm_model <- train %>%
  mutate(isFunctional = ifelse(status_group == "functional", 1, 0)) %>%
  ksvm(isFunctional ~ latitude + longitude + basin + region + scheme_management +
          permit + extraction_type + management + payment_type + water_quality +
          quantity + source + waterpoint_type,
        data = .)

svm_pump <- classify(predict(svm_model, test))

(confusion_matrix <- table(formatted_data$isFunctional, svm_pump,
                           dnn=c("Data", "Predictions")))
```

```
##      Predictions
## Data    0    1
##     0 3705 2053
##     1  814 6724
```

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.7843712
```

Naive Bayes

‘Naive Bayes’ essentially assumes that each explanatory variable is independent of the others and uses the distribution of these for each category of data to construct the distribution of the response variable given the explanatory variables.

We will continue with the ‘Bayesian Networks’ more specifically both to use their power (and their special way) in prediction, and also to interpret their structure for identifying more affecting variables on our target.

```
library(e1071)
naive_model <- train %>%
  naiveBayes(status_group ~ latitude + longitude + basin + region +
               scheme_management + permit + extraction_type + management +
               payment_type + water_quality + quantity + source +
               waterpoint_type,
        data = .)
```

```
(confusion_matrix <- table(formatted_data$status_group,
                           predict(naive_model, test),
                          dnn=c("Data", "Predictions")))
```

```
##                            Predictions
## Data                        functional functional needs repair
##   functional                      5880                     436
##   functional needs repair          491                     178
##   non functional                  1682                     199
##                            Predictions
## Data                        non functional
##   functional                          1222
##   functional needs repair              199
##   non functional                      3009
```

```
(accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix))
```

```
## [1] 0.6819344
```

I was not successful in changing the levels of target variable in ‘Naive Bayes’. We will see we can use different structure in ‘Bayesian Networks’ to get better accuracy rates.

The best model in terms of accuracy rate is: SVM

Model	Accuracy Rate
SVM	0.7843712
Random Forests	0.7835439
Decision Trees	0.7739922
Decision Trees (sign)	0.7737665
Neural Networks (5)	0.7693291
Decision Trees (fwd)	0.7672232
Polynomial	0.7548135
Linear	0.7520307
Linear (fwd)	0.7444344
Logistic	0.7326264
Linear (bwd)	0.7251805
Logistic (fwd)	0.7166817
Naive Bayes	0.6819344

Bayesian Networks

'Bayesian Networks' (BN) are models that see the cause and effect relationships of features in a directed acyclic graph (DAG), so we are going to use BNs in two ways:

1. Learning the structure of the network: By having the structure of relationships among features, we can evaluate the strength of each arc (the path between two node or feature), and then see which variables have the most effect on each other and also on target variable (here: functionality of pumps).

2. Building a model with a power in prediction Just like other models we have used so far, we can learn from data (train) and then evaluate our BN model in terms of prediction power (test).

Learning the structure using BN algorithms

We have three types of learning in BNs:

1. **Using the target variable** As we have seen it already, 'naive bayes' assumes each feature has an effect on the target, after that it can be calculate the chaging the status of target variable by changing the values in explanatory variables. Another similar algorithm assumes that all features can also effect on each other, so it tries to increase the power of prediction, and also a better sense of relationships among explanatory variables - not 'cause and effect' ones here.

2. **Constrained-based algorithms** These kinds of algorithms use *conditional independence tests* to learn the structure of the network. This means if changing in the status of one node (feature) can affect the status of another node, having the status of other nodes. Actually it means how many degrees two node (feature) are independent of each other. Unlike the *Naive Bayesian* algorithms, they are useful to discover causalities. One of the 'constrained-based' algorithms is *Grow-shrink* that tries to reduce unnecessary computations.

3. **Score-based algorithms** All possible DAGs are given the same probability of occurrence. And then given the data, the DAG with the highest network score (reflecting its goodness of fit) is chosen. The discovered relationships may not identify causalities; but we expect the power of prediction be high. As an 'score-based' algorithms, *Hill-climbing* tries to find a better solution by incrementally changing a single element of the solution.

TAN

We select the **status_group** as our target value in TAN structure, and convert it into a Boolean variable.

TAN needs all variables are categorical variables, so we have to discretize them. We have to discretize data using all data we have.

Also for having more recognizable graphs, we are going to rename the variables.

```
library(bnlearn)
library(Rgraphviz)

bn_drop <- drop

bn_drop$stat <- ifelse(bn_drop$stat == "functional", 1, 0)
bn_drop <- subset(bn_drop, select = -c(status_group))

colnames(bn_drop)[1] <- "long"
colnames(bn_drop)[2] <- "lat"
colnames(bn_drop)[3] <- "bas"
colnames(bn_drop)[4] <- "rgn"
colnames(bn_drop)[5] <- "schm"
colnames(bn_drop)[6] <- "prmt"
colnames(bn_drop)[7] <- "ext"
```

```
colnames(bn_drop)[8] <- "mgmt"
colnames(bn_drop)[9] <- "pay"
colnames(bn_drop)[10] <- "qlty"
colnames(bn_drop)[11] <- "qnty"
colnames(bn_drop)[12] <- "src"
colnames(bn_drop)[13] <- "wtr"

bn_drop$stat <- as.factor(bn_drop$stat)
```

Separating the ‘train’ and the ‘test’ samples for discretized data:

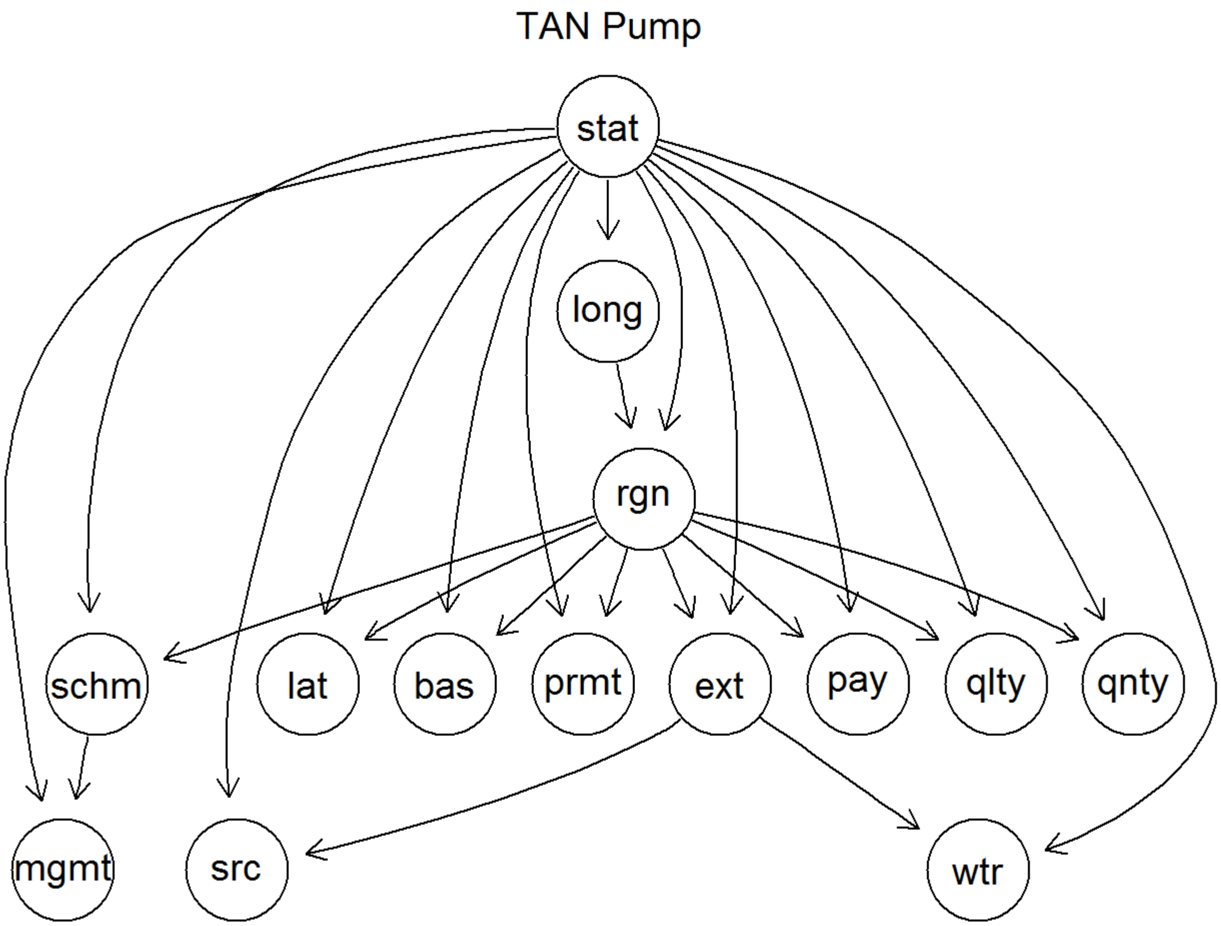
```
d_drop <- discretize(bn_drop, breaks = 3, method = "interval")

smp_size <- floor(0.7 * nrow(d_drop))

set.seed(123456)
train_ind <- sample(seq_len(nrow(d_drop)), size = smp_size)

d_train <- d_drop[train_ind, ]
d_test <- d_drop[-train_ind, ]
```

Separating the ‘train’ and the ‘test’ samples for normal data:

```
smp_size <- floor(0.7 * nrow(bn_drop))

set.seed(123456)
train_ind <- sample(seq_len(nrow(bn_drop)), size = smp_size)

train <- bn_drop[train_ind, ]
test <- bn_drop[-train_ind, ]
```

Learning TAN structure:

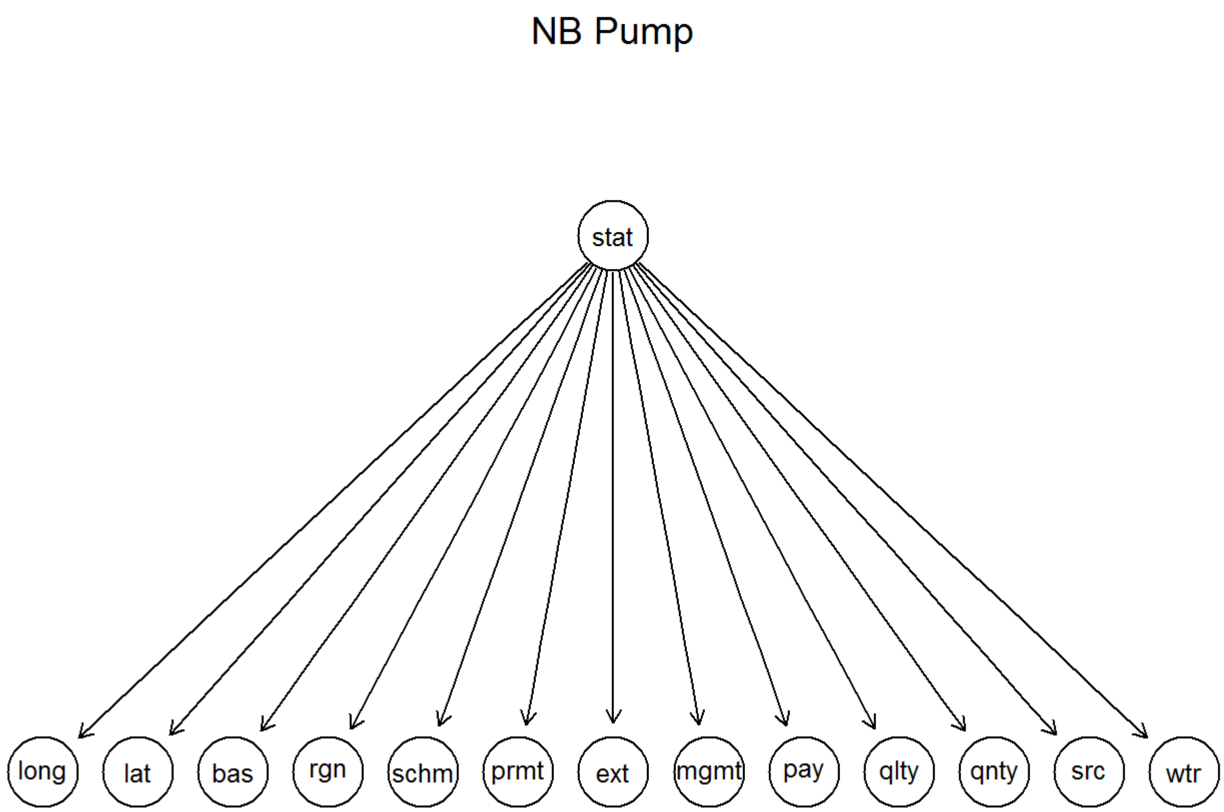
```
tan <- tree.bayes(d_train, "stat")
graphviz.plot(tan, main = "TAN Pump")
```



Naive Bayes (NB)

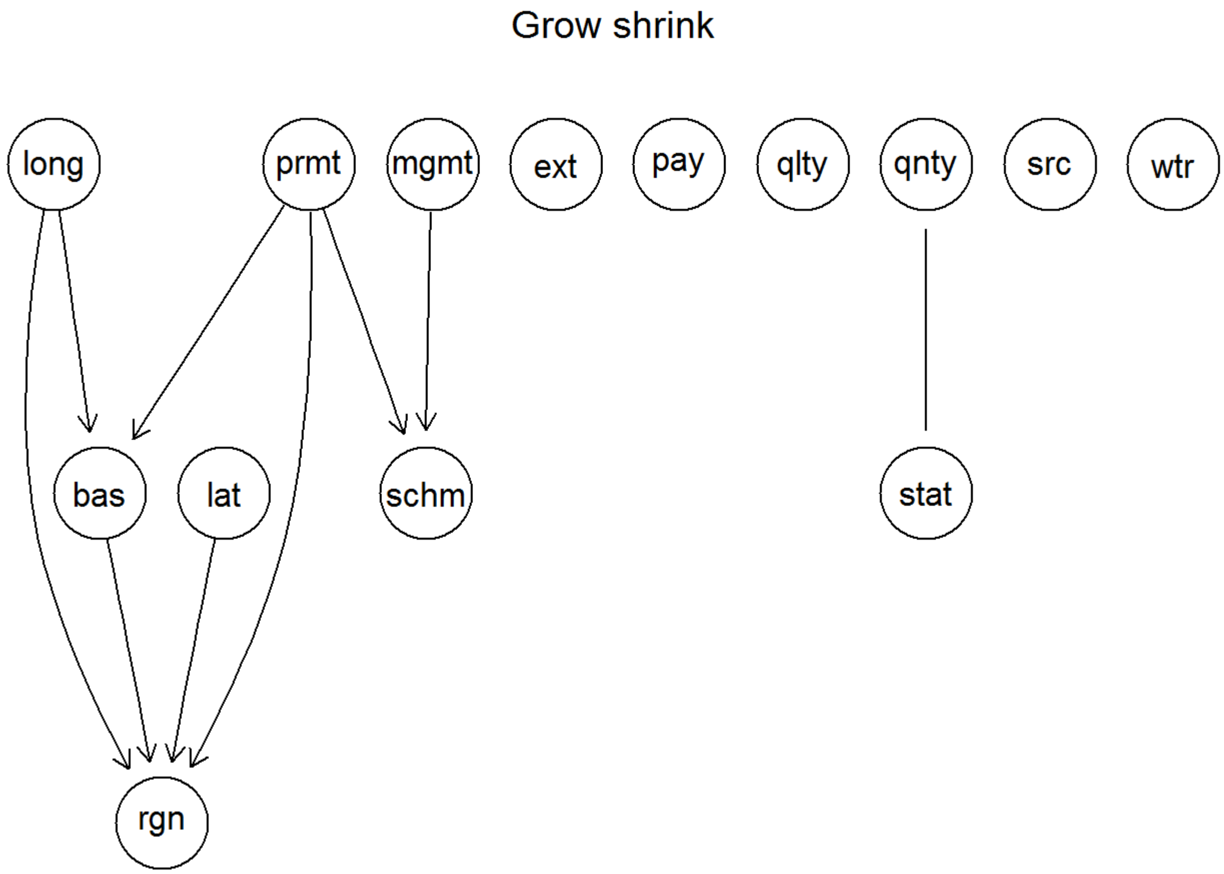
Just like the TAN structure, but this time the child cannot have relationships with each other.

```
nb <- naive.bayes(d_train, "stat")
graphviz.plot(nb, main = "NB Pump")
```



Grow-Shrink

```
gs_d <- gs(d_train, alpha = 0.05, test = "x2") # alpha = the sig. level
graphviz.plot(gs_d, main = "Grow shrink")
```

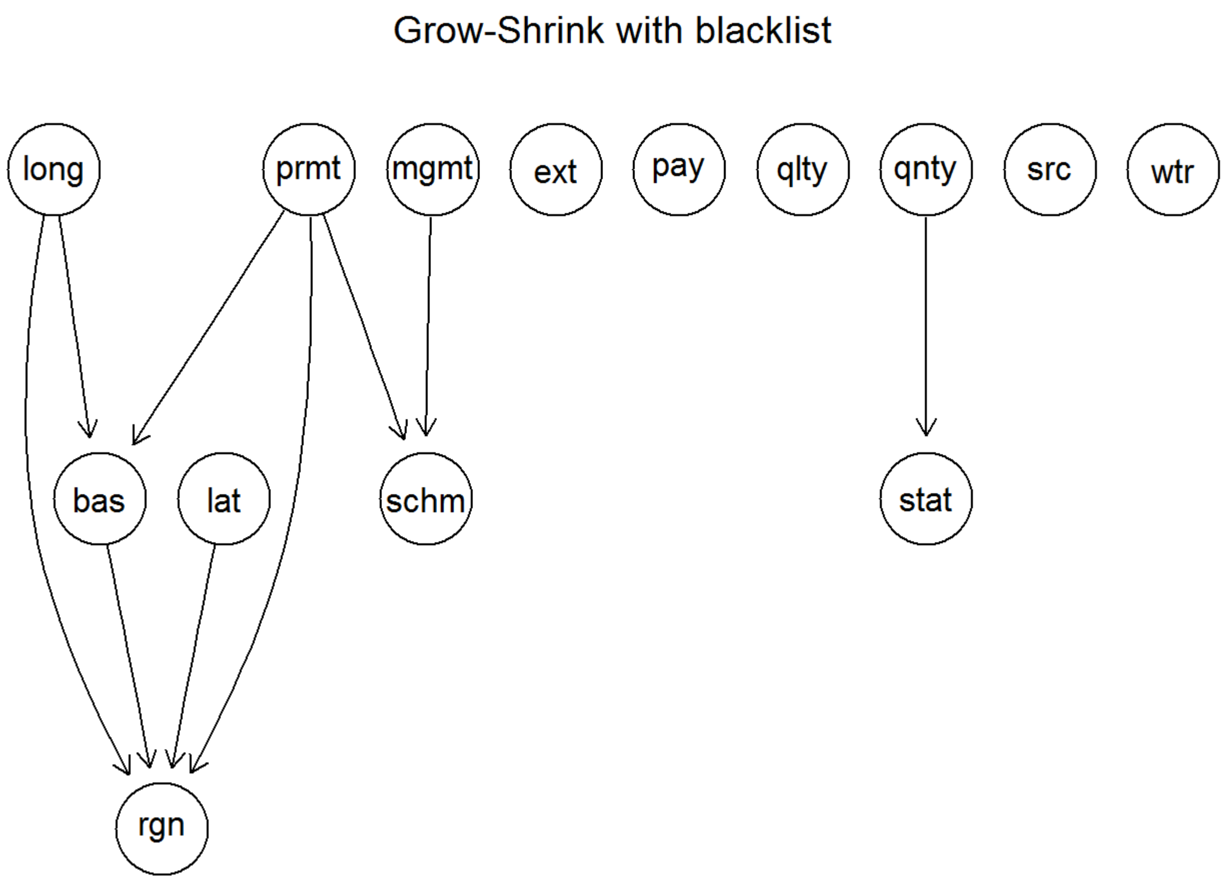


In the constrained-based graphs some links are undirected (the software cannot establish the direction of the causality). We see some relationships do not have meaning, and because the structure is directed, then we have to correct them.

```
undirected.arcs(gs_d)
```

```
##      from   to
## [1,] "qnty" "stat"
## [2,] "stat" "qnty"
```

```
blacklist = data.frame(from = c("stat"),
                       to =   c("qnty"))
gs_db <- gs(d_train, alpha = 0.05, test = "x2", blacklist = blacklist)
graphviz.plot(gs_db, main = "Grow-Shrink with blacklist")
```



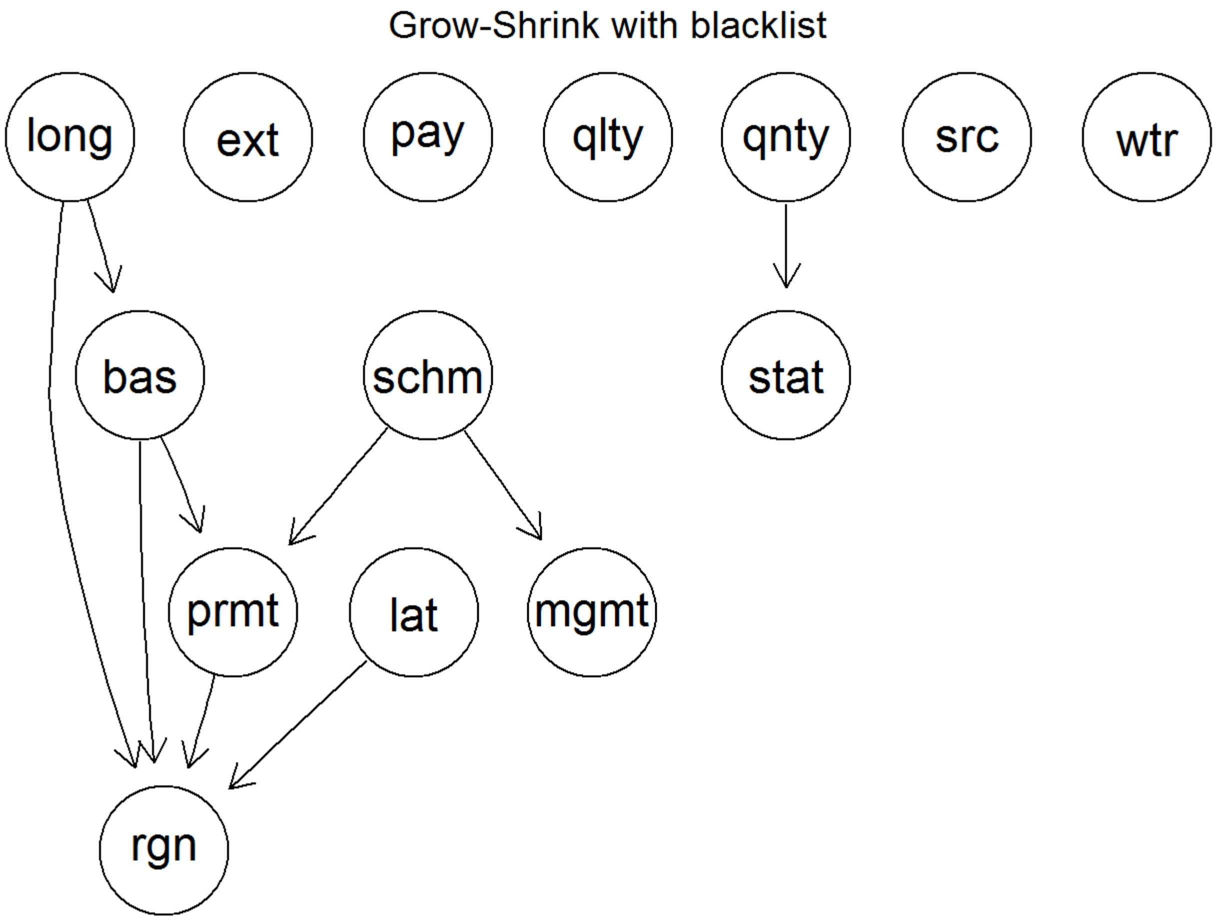
We have to correct other undirected arcs:

```
undirected.arcs(gs_db)
```

```
##      from   to
## [1,] "long" "bas"
## [2,] "bas"  "long"
## [3,] "schm" "mgmt"
## [4,] "mgmt" "schm"
```

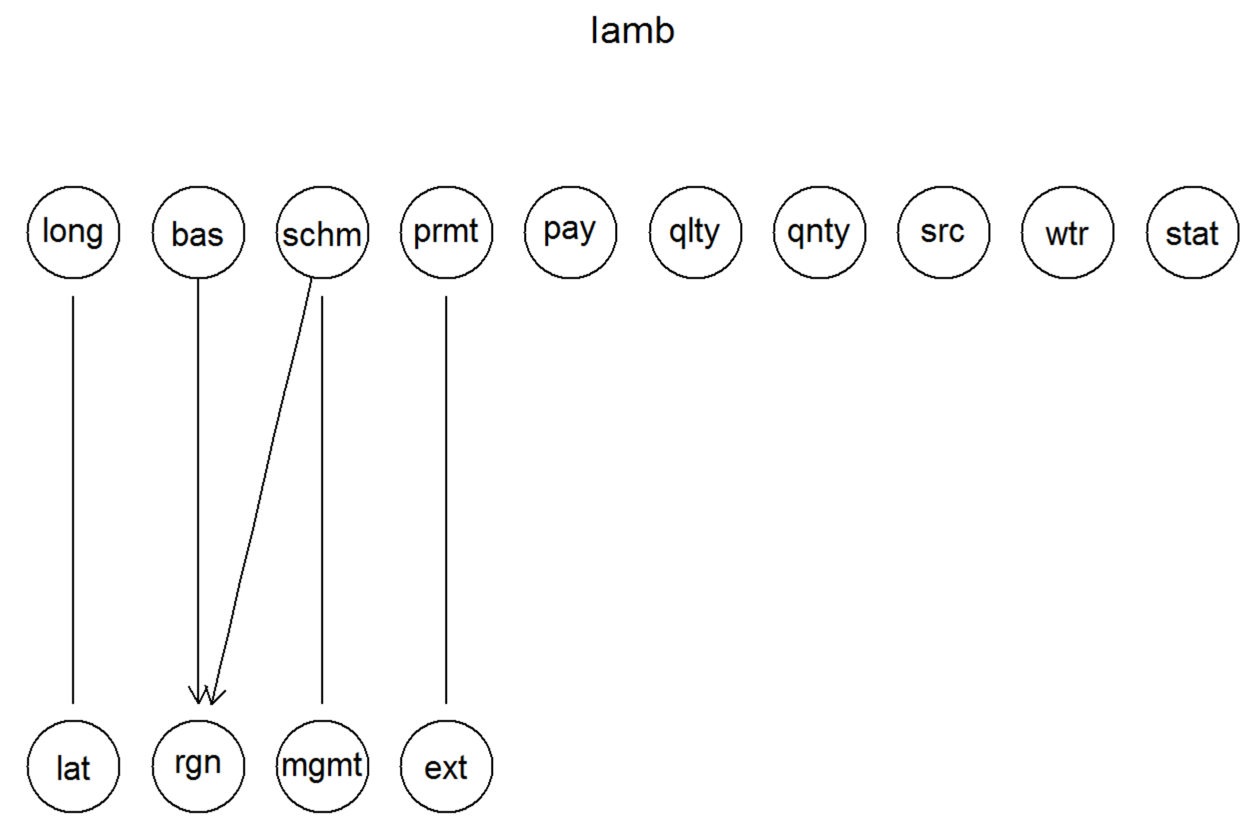
We assume the ‘scheme_management’ affects on ‘management’ of pumps. And also ‘longitude’ affects on ‘basin’.

```
blacklist = data.frame(from = c("stat", "bas", "mgmt"),
                       to =   c("qnty", "long", "schm"))
gs_db <- gs(d_train, alpha = 0.05, test = "x2", blacklist = blacklist)
graphviz.plot(gs_db, main = "Grow-Shrink with blacklist")
```



Incremental Association Markov Blanket (IAMB)

```
iamb <- iamb(train)
graphviz.plot(iamb, main = "Iamb")
```

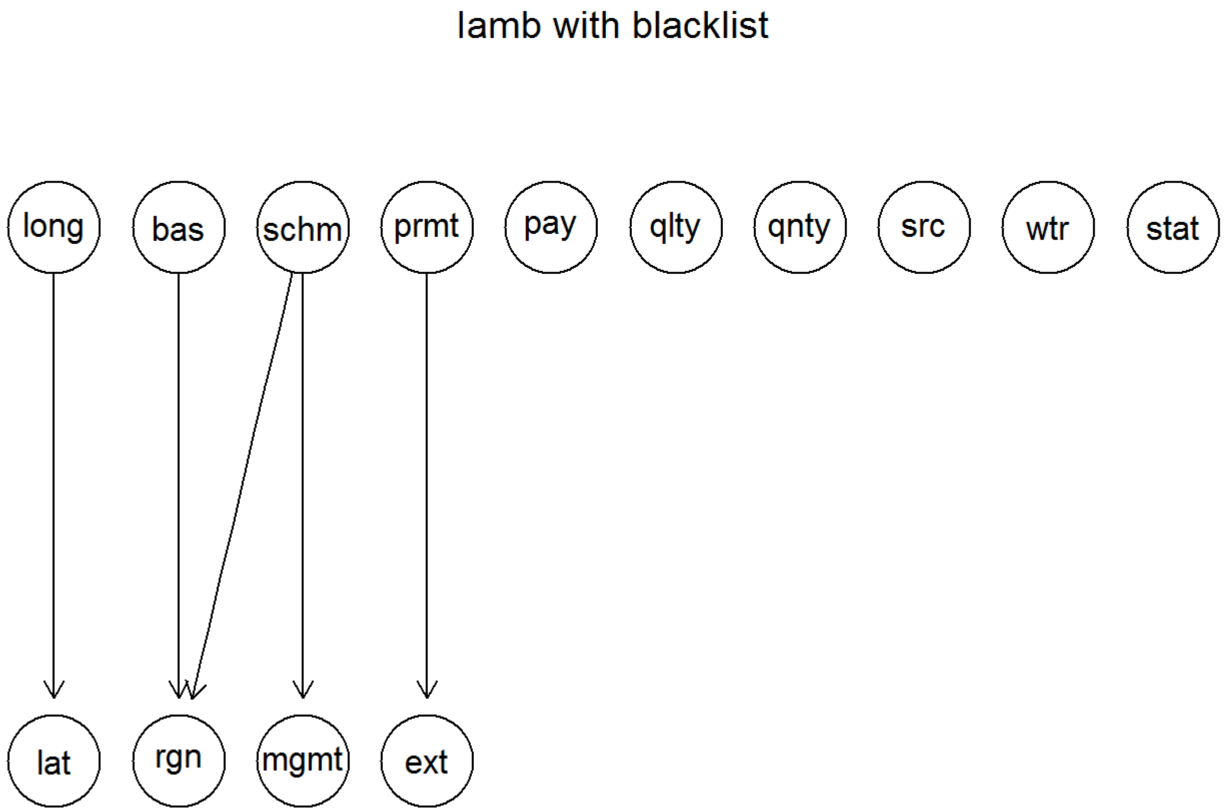



```
undirected.arcs(iamb)
```

```
##      from   to    
## [1,] "long" "lat" 
## [2,] "lat"  "long"
## [3,] "schm" "mgmt"
## [4,] "prmt" "ext" 
## [5,] "ext"  "prmt"
## [6,] "mgmt" "schm"
```

We assume that ‘permit’ affects on ‘extraction_type’.

```
blacklist = data.frame(from = c("lat",  "ext",  "mgmt"),
                        to =   c("long", "prmt", "schm"))
iamb_b <- iamb(train, blacklist = blacklist)
graphviz.plot(iamb_b, main = "Iamb with blacklist")
```

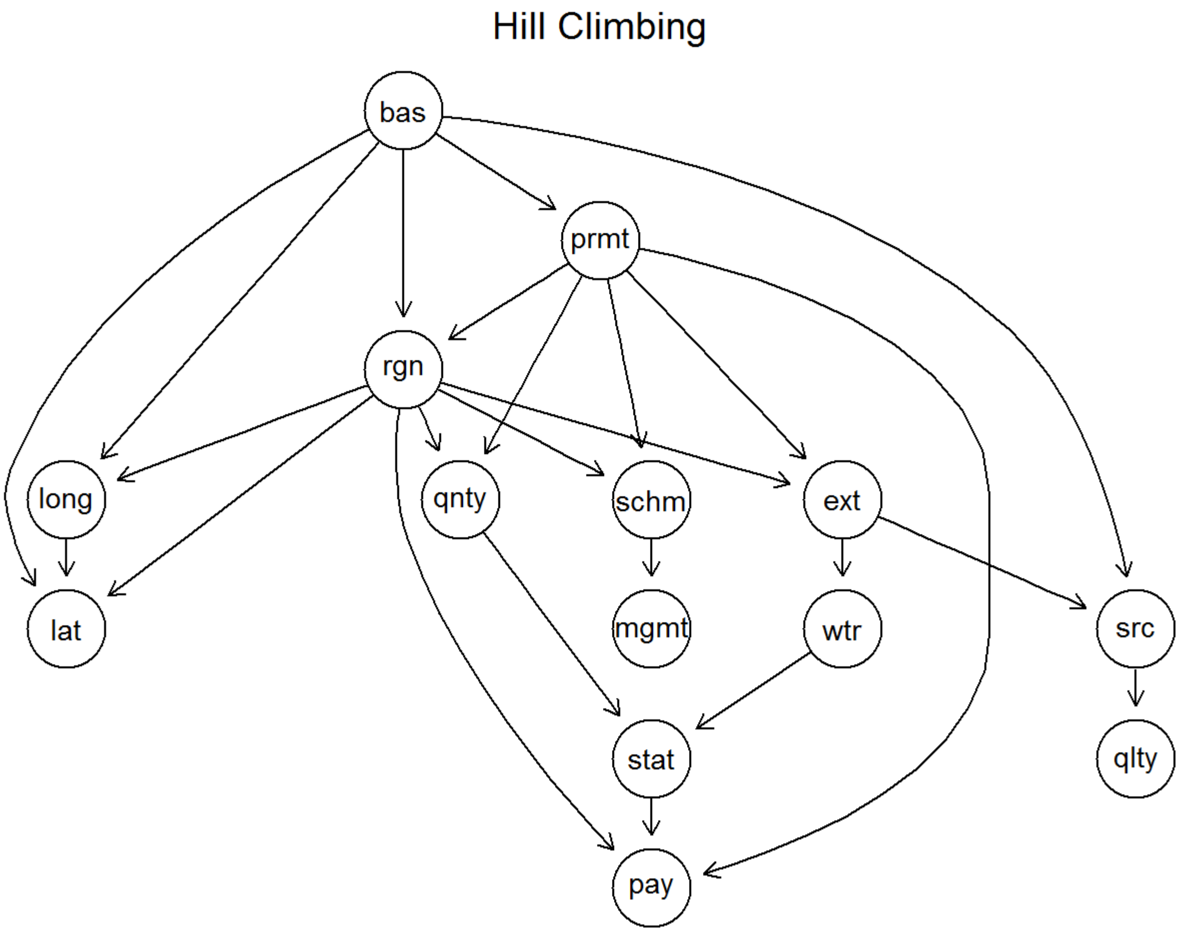


We assume the ‘basin’ affects on ‘region’.

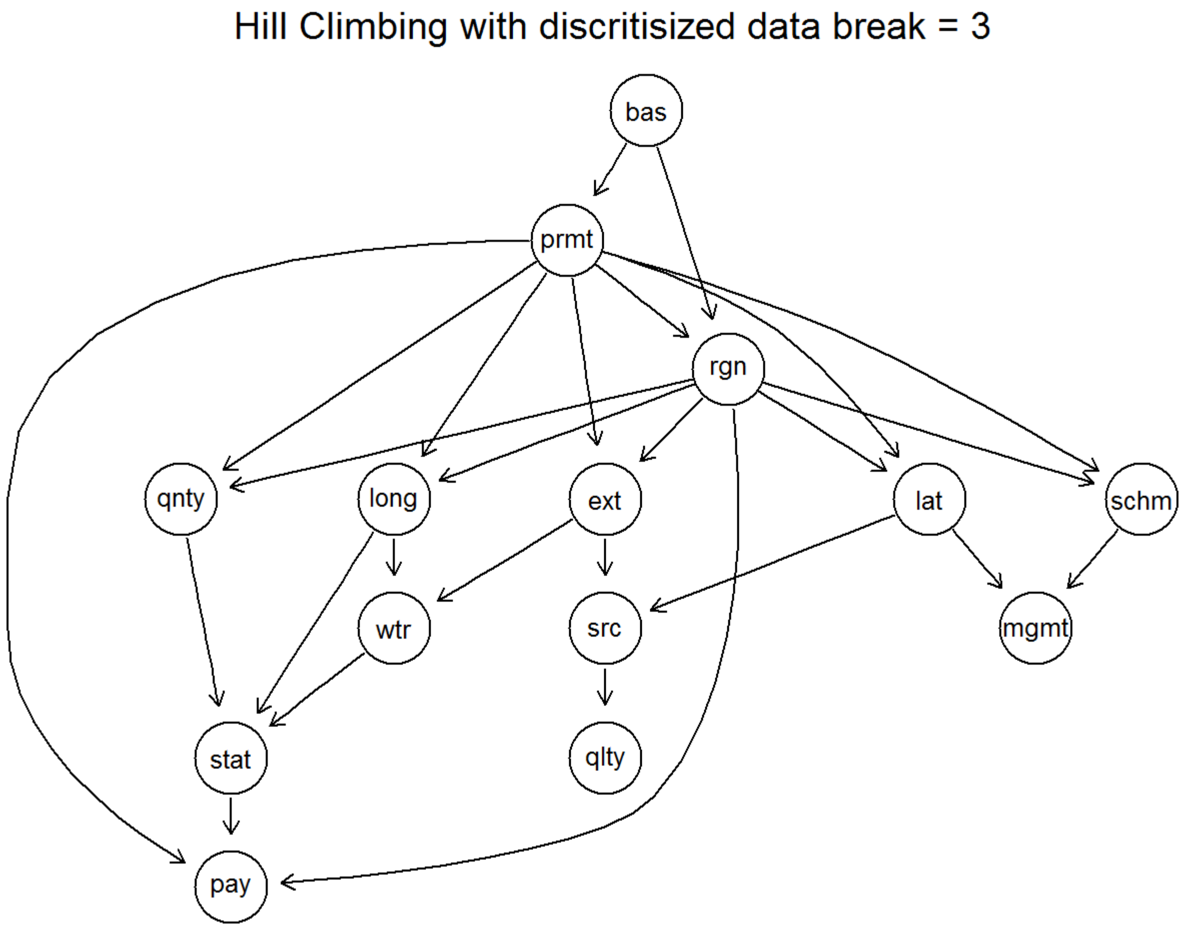
Hill-Climbing

A hill-climbing greedy search on the space of the directed graphs. The optimized implementation uses score caching, score decomposability and score equivalence to reduce the number of duplicated tests.

```
hc <- hc(train)
graphviz.plot(hc, main = "Hill Climbing")
```




```
hc_d <- hc(d_train, score = "bic")
graphviz.plot(hc_d, main = "Hill Climbing with discritisized data break = 3")
```




```
all.equal(hc, hc_d)
```

```
## [1] "Different number of directed/undirected arcs"
```

Comparison among structures

Now we are going to test and compare structures of TAN, NB, GS, IAMB, and HC.

Number and power of arcs

It may does not mean for comparison between naive and TAN, because it is trivial that TAN has more arcs.

```
arc.strength(tan, data = d_train, criterion = "x2")
```

```
##    from   to      strength
## 1  stat long 7.175184e-122
## 2  stat  lat  1.800154e-27
## 3  stat  bas  7.937755e-20
## 4  stat  rgn 1.722065e-212
## 5  stat schm 8.782034e-262
## 6  stat prmt  3.225148e-71
## 7  stat  ext  0.000000e+00
## 8  stat mgmt  3.310034e-95
## 9  stat  pay  0.000000e+00
## 10 stat qlty  4.281160e-39
```

```
## 11 stat qnty  0.000000e+00
## 12 stat  src 1.406415e-153
## 13 stat  wtr 1.346442e-241
## 14 long  rgn  0.000000e+00
## 15  rgn  lat  0.000000e+00
## 16  rgn  bas  0.000000e+00
## 17  rgn schm  0.000000e+00
## 18  rgn prmt  0.000000e+00
## 19  rgn  ext  0.000000e+00
## 20  rgn  pay  0.000000e+00
## 21  rgn qlty  0.000000e+00
## 22  rgn qnty  0.000000e+00
## 23 schm mgmt  0.000000e+00
## 24  ext  src  0.000000e+00
## 25  ext  wtr  0.000000e+00
```

```
arc.strength(nb, data = d_train, criterion = "x2")
```

```
##      X1   X2      strength
## 1  stat long 7.175184e-122
## 2  stat  lat  1.859293e-27
## 3  stat  bas 4.824184e-169
## 4  stat  rgn  0.000000e+00
## 5  stat schm 5.856931e-126
## 6  stat prmt  2.550164e-14
## 7  stat  ext  0.000000e+00
## 8  stat mgmt 4.289832e-177
## 9  stat  pay  0.000000e+00
## 10 stat qlty  2.481566e-23
## 11 stat qnty  0.000000e+00
## 12 stat  src  3.581694e-92
## 13 stat  wtr  0.000000e+00
```

```
arc.strength(gs_db, data = d_train, criterion = "x2")
```

```
##   from   to strength
## 1 long  bas        0
## 2 long  rgn        0
## 3  lat  rgn        0
## 4  bas  rgn        0
## 5  bas prmt        0
## 6 schm prmt        0
## 7 schm mgmt        0
## 8 prmt  rgn        0
## 9 qnty stat        0
```

```
arc.strength(iamb_b, data = train)
```

```
##   from   to strength
## 1 long  lat        0
```

```
## 2  bas  rgn        0
## 3 schm  rgn        0
## 4 schm mgmt        0
## 5 prmt  ext        0
```

Less p-values show more power for arc, and this means the dependency between two variables is more, and therefore more ‘variance reduction’ we have.

However, GS shows the **Quantity** or the volume of the water has the most effect on ‘functionality’ of the pumps.

```
arc.strength(hc, data = train)
```

```
##    from   to     strength
## 1   rgn long -33506.26003
## 2   rgn  lat -21110.92076
## 3   bas  rgn -49582.19065
## 4  schm mgmt -24760.24749
## 5   ext  src -16184.62205
## 6   ext  wtr -20552.16561
## 7   rgn schm -12675.67543
## 8   rgn  ext -13200.52395
## 9   rgn  pay  -9302.12019
## 10  bas  lat  -9164.90027
## 11  bas long  -6186.79000
## 12 long  lat  -6091.80842
## 13 prmt  rgn  -3202.38301
## 14  rgn qnty  -3319.48230
## 15 qnty stat  -2250.11802
## 16  src qlty  -2116.10283
## 17 prmt  ext  -1464.53314
## 18  wtr stat  -1417.92483
## 19 prmt  pay   -792.10603
## 20 prmt qnty   -950.65693
## 21 prmt schm   -779.00912
## 22  bas prmt   -689.83411
## 23 stat  pay   -245.00973
## 24  bas  src    -80.86357
```

The results reflect that BIC becomes worse if we remove any of the arcs.

Again here we can see the **Quantity** and **Water_point** are the most affecting variables on ‘functionality’ of the pumps. ‘Water point’ means the way the water is being exploited e.g. if it is exploited through a dam, or it is in the way of cattle, etc.

```
arc.strength(hc_d, data = d_train, criterion = "bic")
```

```
##    from   to     strength
## 1   bas  rgn -49582.19065
## 2   rgn  lat -27170.81835
## 3   rgn long -27132.61401
## 4  schm mgmt -23069.76012
## 5   ext  src -21403.74403
```

```
## 6   ext  wtr -19272.19422
## 7   rgn schm -12675.67543
## 8   rgn  ext -13200.52395
## 9   rgn  pay  -9302.12019
## 10 prmt  rgn  -3202.38301
## 11  rgn qnty  -3319.48230
## 12 qnty stat  -2223.19970
## 13  src qlty  -2116.10283
## 14 prmt  ext  -1464.53314
## 15  wtr stat  -1302.19041
## 16 prmt  pay   -792.10603
## 17 prmt qnty   -950.65693
## 18 prmt long   -871.18016
## 19 prmt  lat   -781.52852
## 20 prmt schm   -779.00912
## 21  lat mgmt   -730.01812
## 22  lat  src   -696.27934
## 23  bas prmt   -689.83411
## 24 long stat   -290.00922
## 25 stat  pay   -245.00973
## 26 long  wtr    -27.74195
```

Again here we can see the **Quantity**, **Water_point**, and **longitude** have the most effects. Maybe we can relate the 'longitude' to the volume of the water or even the 'water point'.

Apparently, Hill-climbing with discretized data can detect more relationships between features. This investigation depends only on the number of discovered relationships and therefore we cannot relate it to the power of model in prediction.

We have other measures (for evaluating the including features in our model), like 'influence analysis' and 'approximate inference'.

Score function

We evaluate the scores of our networks by using 'test' data.

- **Naive Bayes**

```
score(nb, data = d_test, type = "bic")
```

```
## [1] -243070.5
```

- **TAN**

```
score(tan, data = d_test, type = "bic")
```

```
## [1] -161849
```

- **Grow Shrink**

```
score(gs_db, data = d_test, type = "bic")
```

```
## [1] -214630.6
```

- **IAMB**

```
score(iamb_b, data = d_test, type = "bic")
```

```
## [1] -219231.5
```

- **Hill Climbing**

```
score(hc, data = d_test, type = "bic")
```

```
## [1] -165796.1
```

- **Hill Climbing (discretized data)**

```
score(hc_d, data = d_test, type = "bic")
```

```
## [1] -159839
```

The results show ‘hill-climbing with discretized data’ has the most matching points with our data.

Power of prediction

K-fold cross-validation

We are going to evaluate the ‘misclassification rate’ using k-fold cross-validation. We are going to perform a 5-fold cross validation for all structures, using the classification error for ‘status_group’ as a loss function.

- **Naive Bayes**

```
(nbcv = bn.cv(d_test, nb, loss = "pred", k = 5,
              loss.arg = list(target = "stat")))
```

```
##
##   k-fold cross-validation for Bayesian networks
##
##   target network structure:
##    [Naive Bayes Classifier]
##   number of subsets:                     5
##   loss function:                         Classification Error
##   training node:                         stat
##   expected loss:                         0.2908394
```

- **TAN**

```
(tancv = bn.cv(d_test, tan, loss = "pred", k = 5,
               loss.arg = list(target = "stat")))
```

```
##
##   k-fold cross-validation for Bayesian networks
```

```
##
##   target network structure:
##    [stat][long|stat][rgn|stat:long][lat|stat:rgn][bas|stat:rgn]
##    [schm|stat:rgn][prmt|stat:rgn][ext|stat:rgn][pay|stat:rgn]
##    [qlty|stat:rgn][qnty|stat:rgn][mgmt|stat:schm][src|stat:ext]
##    [wtr|stat:ext]
##   number of subsets:                       5
##   loss function:                           Classification Error
##   training node:                           stat
##   expected loss:                           0.257145
```

- **Grow Shrink**

```
(gscv = bn.cv(d_test, gs_db, loss = "pred", k = 5,
              loss.arg = list(target = "stat")))
```

```
##
##   k-fold cross-validation for Bayesian networks
##
##   target network structure:
##    [long][lat][schm][ext][pay][qlty][qnty][src][wtr][bas|long][mgmt|schm]
##    [stat|qnty][rgn|long:lat:bas][prmt|bas:rgn:schm]
##   number of subsets:                       5
##   loss function:                           Classification Error
##   training node:                           stat
##   expected loss:                           0.3440132
```

- **IAMB**

```
(iambcv = bn.cv(d_test, iamb_b, loss = "pred", k = 5,
                loss.arg = list(target = "stat")))
```

```
##
##   k-fold cross-validation for Bayesian networks
##
##   target network structure:
##    [long][bas][schm][prmt][pay][qlty][qnty][src][wtr][stat][lat|long]
##    [rgn|bas:schm][ext|prmt][mgmt|schm]
##   number of subsets:                       5
##   loss function:                           Classification Error
##   training node:                           stat
##   expected loss:                           0.4330626
```

- **Hill Climbing**

```
(hccv = bn.cv(d_test, hc, loss = "pred", k = 5,
              loss.arg = list(target = "stat")))
```

```
##
##   k-fold cross-validation for Bayesian networks
##
```

```
##   target network structure:
##    [bas][prmt|bas][rgn|bas:prmt][long|bas:rgn][schm|rgn:prmt][ext|rgn:prmt]
##    [qnty|rgn:prmt][lat|long:bas:rgn][mgmt|schm][src|bas:ext][wtr|ext]
##    [qlty|src][stat|qnty:wtr][pay|rgn:prmt:stat]
##   number of subsets:                      5
##   loss function:                          Classification Error
##   training node:                          stat
##   expected loss:                          0.2905385
```

- **Hill Climbing (discretized data)**

```
(hccv_d = bn.cv(d_test, hc_d, loss = "pred", k = 5,
                loss.arg = list(target = "stat")))
```

```
##
##   k-fold cross-validation for Bayesian networks
##
##   target network structure:
##    [bas][prmt|bas][rgn|bas:prmt][long|rgn:prmt][lat|rgn:prmt]
##    [schm|rgn:prmt][ext|rgn:prmt][qnty|rgn:prmt][mgmt|lat:schm][src|lat:ext]
##    [wtr|long:ext][qlty|src][stat|long:qnty:wtr][pay|rgn:prmt:stat]
##   number of subsets:                      5
##   loss function:                          Classification Error
##   training node:                          stat
##   expected loss:                          0.2809116
```

We can see that TAN works here better.

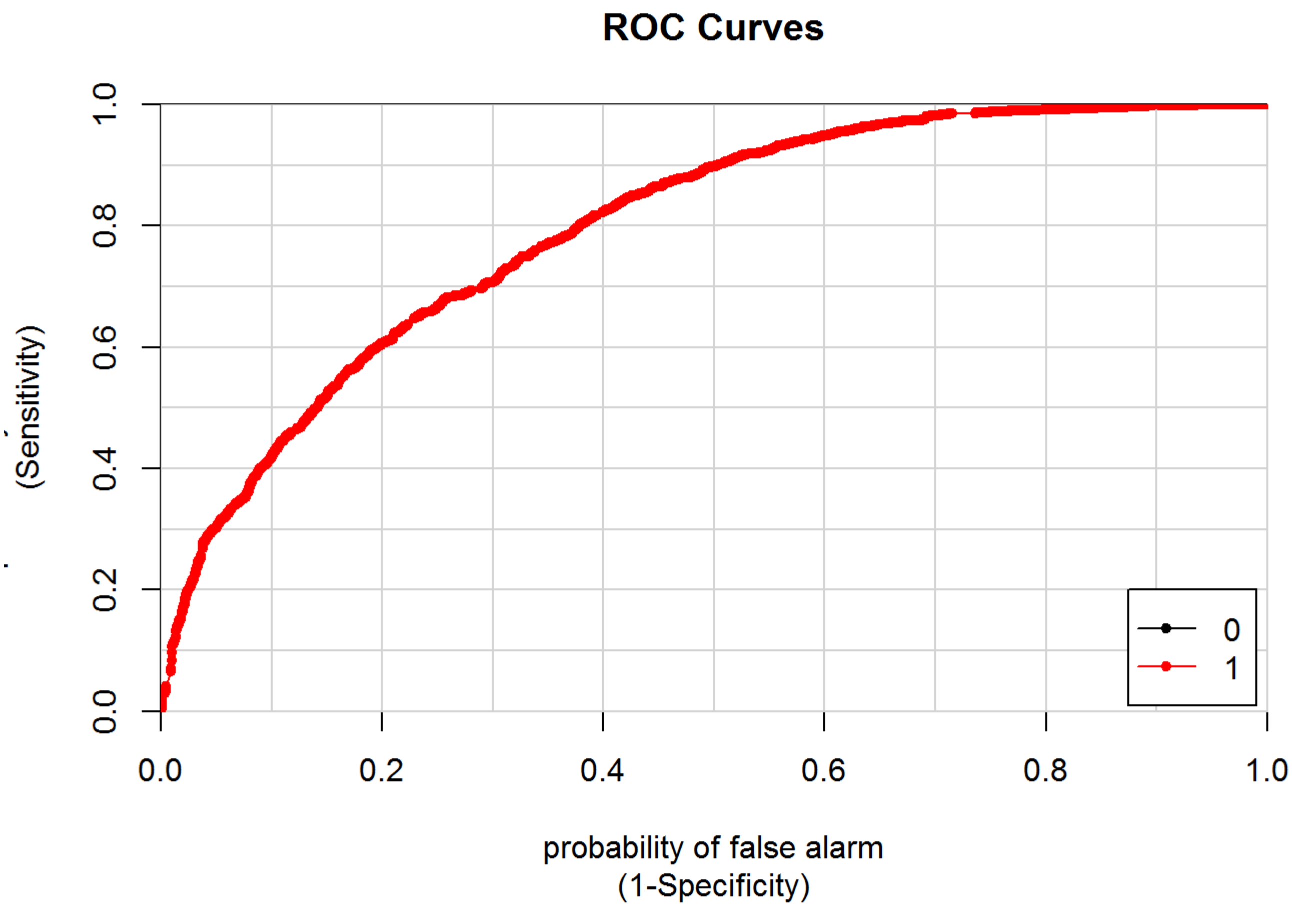
AUC and ROC curve

We use predict function for the best cross-validation structure as well as all structures we have (Naive, GS, and HC)

We use ‘Receiver operating characteristic’ (**ROC Curve**) as another measure to evaluate the power of prediction for our models. ROC curve is based on ‘specificity’ and ‘sensitivity’ - we have seen before. Both measures must be close to one to have a better model. By applying the ‘1 - Specificity’ in the x-axis, we can conclude more area under curve (AUC) results in having better model.

- **Naive Bayes**

```
library(caTools)
fitted_nb = bn.fit(nb, d_train)
nb_fit <- as.grain(fitted_nb)
pred_nb = predict(nb_fit, response = c("stat"), newdata = d_test,
                  predictors = names(d_test)[-14], type = "distribution")
colAUC(pred_nb$pred$stat, d_test[, 14], plotROC = TRUE)
```




```
##                    0         1
## 0 vs. 1 0.7957041 0.7957041
```

Unfortunately, all attempts to calculate the AUC and ROC curve fails. So we compare the models in other measures.

Conclusion

We evaluate the BN models in two ways: Best fitting with data (that also was useful to detect most affecting variables on target), and the prediction power:

Measure	NB	TAN	GS	IAMB	HC	HC (dis)
Number of Arcs	NA	2	4	5	3	1
Power of Arcs	NA	NA	3	4	2	1
Score function	6	2	4	5	3	1
K-fold cross validation	4	1	5	6	3	2

Here in this problem, the most important measure for us is misclassification rate. So Tan structure has the best performance in it.

Final ranking based on misclassification rate is: The best model in terms of accuracy rate is: SVM

Model	Accuracy Rate

SVM	0.7843712
Random Forests	0.7835439
Decision Trees	0.7739922
Decision Trees (sign)	0.7737665
Neural Networks (5)	0.7693291
Decision Trees (fwd)	0.7672232
Polynomial	0.7548135
Linear	0.7520307
Linear (fwd)	0.7444344
TAN	0.7428550
Logistic	0.7326264
Linear (bwd)	0.7251805
Hill Climbing (disc)	0.7190884
Logistic (fwd)	0.7166817
Hill Climbing	0.7094615
Naive Bayes (bnlearn)	0.7091606
Naive Bayes (e1071)	0.6819344
Grow Shrink	0.6559868
IAMB	0.5669374

Also we can see the most affecting features on functionality of pumps:

- **forward selection**

‘region’, ‘scheme_management’, ‘extraction_type’, ‘payment_type’, ‘quantity’, and ‘waterpoint_type’

- **backward selection**

‘payment_type’, ‘quantity’, ‘source’ and ‘waterpoint_type’

So **payment_type**, **quantity**, and **waterpoint_type** have most effect on target.

- **TAN**

‘quantity’, ‘etraction_type’, and ‘payment_type’

- **Naive Bayes**

‘quantity’, ‘etraction_type’, ‘region’, ‘waterpoint_type’ and ‘payment_type’

- **Grow Shrink**

‘quantity’

- **Hill Climbing**

‘quantity’, ‘waterpoint_type’, ‘extracton_type’, ‘region’ and ‘permit, and ’basin’

- **Hill Climbing (discretized data)**

‘longitude’, ‘quantity’, ‘waterpoint_type’, ‘extracton_type’, ‘region’, ‘permit’, and ‘basin’

Feature	Rank
Quantity	7
Waterpoint_type	5
Extraction_type	5
Region	4
Payment_type	4
Permit	2
Basin	2
Scheme_management	1
Source	1
Longitude	1