Restaurant Inspections

Contents

1. Introduction

In this data analysis project, I will use data on restaurant inspections to predict the number of days until the next inspection occurs, for a given restaurant.

The data set contains information on restaurant inspections in New York City carried out by the Department of Health and Hygiene (DOHMH) between 2010 and 2017. It contains the following variables:

- CAMIS: a unique identifier for the restaurant
- DBA: name of (doing business as) the restaurant
- BORO: borough in which the restaurant is located
- BUILDING: building number for the restaurant
- STREET: street name at which the restaurant is located
- ZIPCODE: Zip code as per the (mailing) address of the restaurant
- PHONE: phone number
- CUISINE DESCRIPTION: cuisine of the restaurant
- INSPECTION DATE: date of inspection
- ACTION: action associated with the given inspection
- VIOLATION CODE: violation code associated with the given inspection
- VIOLATION DESCRIPTION: description that corresponds to violation code
- CRITICAL FLAG: indication if violation is critical or not
- SCORE: total score for a particular inspection
- GRADE: grade issued for the given inspection
- GRADE DATE: date when the current grade was issued to the restaurant
- RECORD DATE: The date when the extract was run to produce this data set
- INSPECTION TYPE: The type of inspection. A combination of the program and inspection type

We read in the data:

```
(restaurants_raw <- read_excel("New_York_City_Restaurants.xlsx"))
```

```
## # A tibble: 376,945 x 18
##      CAMIS DBA     BORO  BUILDING STREET ZIPCODE  PHONE `CUISINE DESCRIPT~
##      <dbl> <chr>   <chr> <chr>    <chr>    <dbl>  <dbl> <chr>
##  1  5.00e7 EDEN W~ MANH~ 43       E 34T~   10016 2.13e9 Jewish/Kosher
##  2  4.09e7 RUBENS~ BROO~ 1533     70 ST~   11228 7.18e9 Pizza
##  3  5.01e7 KUNGFU~ MANH~ 805      8TH A~   10019 6.47e9 Chinese
##  4  5.00e7 MCDONA~ MANH~ 1499     3RD A~   10028 2.13e9 American
##  5  5.01e7 Brookl~ BROO~ 1218     UNION~   11225 7.18e9 American
##  6  4.16e7 ASELLI~ MANH~ 420      PARK ~   10016 2.12e9 Italian
##  7  5.00e7 TACO Y~ MANH~ 142      NAGLE~   10040 2.13e9 Latin (Cuban, Dom~
##  8  4.12e7 KAM SH~ BROO~ 1608     AVENU~   11230 7.18e9 Chinese
##  9  4.17e7 NEPALE~ QUEE~ 907      SENEC~   11385 7.18e9 Indian
## 10  4.14e7 BUTTER~ MANH~ 346      EAST ~   10128 2.13e9 American
## # ... with 376,935 more rows, and 10 more variables: `INSPECTION
## #   DATE` <dttm>, ACTION <chr>, `VIOLATION CODE` <chr>, `VIOLATION
## #   DESCRIPTION` <chr>, `CRITICAL FLAG` <chr>, SCORE <dbl>, GRADE <chr>,
## #   `GRADE DATE` <dttm>, `RECORD DATE` <dttm>, `INSPECTION TYPE` <chr>
```

For convenience, we rename the variables.

```
##  [1] "id"              "rest_name"       "boro"
##  [4] "building"        "street"          "zipcode"
##  [7] "phone"           "cuisine_descr"   "inspection_date"
## [10] "action"          "violation_code"  "violation_descr"
## [13] "critical_flag"   "score"           "grade"
## [16] "grade_date"      "record_date"     "inspection_type"
```

2. Exploratory Analysis

Let us turn to the exploration of the data..

2.1 Inspection Date

```
restaurants_raw$inspection_date <- ymd(restaurants_raw$inspection_date)
summary(restaurants_raw$inspection_date)
```

```
##         Min.      1st Qu.       Median         Mean      3rd Qu.
## "1900-01-01" "2015-09-09" "2016-08-10" "2016-03-26" "2017-07-18"
##         Max.
## "2018-03-19"
```

Apparently, there are some observations with an inspection date of "1900-01-01". Of course, these are wrong values. On closer inspection we see that values across all the variables relating to grading are missing for those observations

```
## # A tibble: 1 x 1
##   na_inspectionDate
##               <dbl>
## 1            0.0031
```

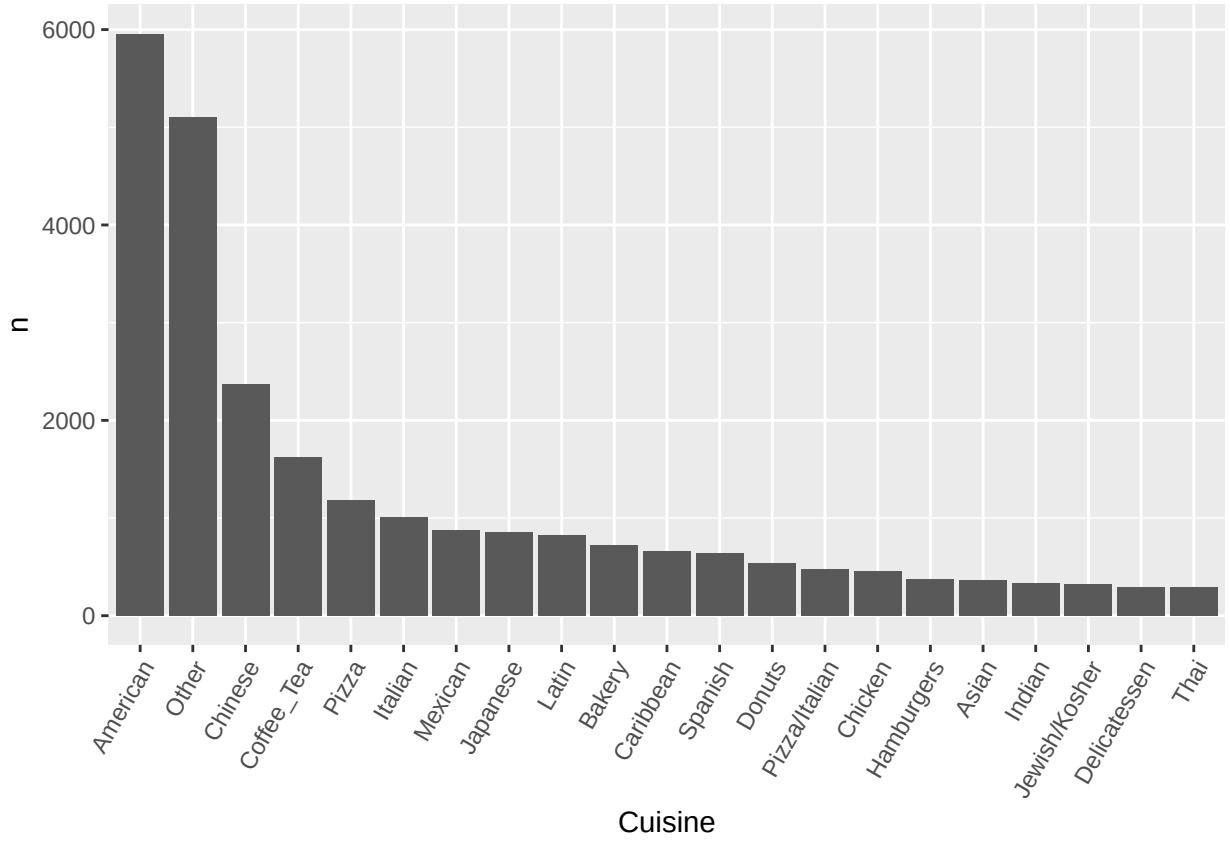
Those obervations amount to .03 % of all observations. For convenience, we will ignore them.

2.2 Cuisine description

```
restaurants_raw$cuisine_descr <- as.factor(restaurants_raw$cuisine_descr)

# Relabel level "CafÃfÂ©/Coffee/Tea" which is at position 14 in levels vector
cuis_ind <- levels(restaurants_raw$cuisine_descr) == levels(restaurants_raw$cuisine_descr)[14]
levels(restaurants_raw$cuisine_descr)[cuis_ind] <- "Coffee_Tea"

# Relabel level "Latin (..." to "Latin"
levels(restaurants_raw$cuisine_descr)[levels(restaurants_raw$cuisine_descr) == "Latin (Cuban, Dominican

# Lump together the least common categories, keeping the 20 most frequent.
restaurants_raw <- restaurants_raw %>%
  mutate(cuisine_descr = fct_lump(f = cuisine_descr, n = 20))
```

Let us look at the distribution of 'cuisine_descr':

```
restaurants_raw %>%
  group_by(cuisine_descr) %>%
  summarise(n = n_distinct(id)) %>%
  ungroup() %>%
  ggplot(aes(x = reorder(cuisine_descr, -n), y = n)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  xlab("Cuisine")
```



The most frequent cuisines are those labelled 'American', 'Other' and 'Chinese'.

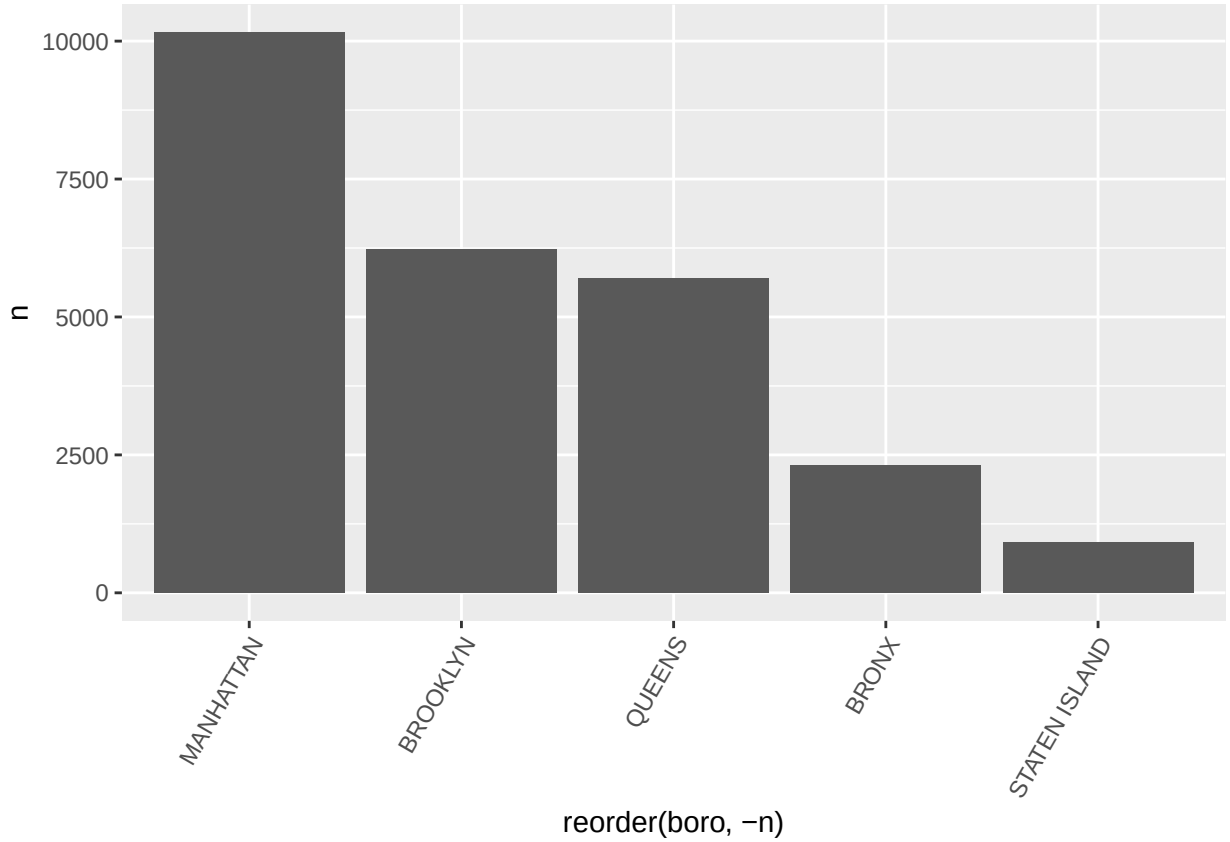
2.3 Boros

Next, let us take a look at the distribution of 'Boro'.

```
restaurants_raw$boro <- as.factor(restaurants_raw$boro)
levels(restaurants_raw$boro)[levels(restaurants_raw$boro) == "Missing"] <- NA
summary(restaurants_raw$boro)
```

```
##         BRONX      BROOKLYN     MANHATTAN        QUEENS STATEN ISLAND
##         33105         94772        149821         85191         12900
```

```
restaurants_raw %>%
  group_by(boro) %>%
  summarize(n = n_distinct(id)) %>%
  ggplot(aes(x = reorder(boro, -n), y = n)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle=60, hjust=1))
```



Most of the eating establishments are located in Manhattan, followed by Brooklyn and Queens.

2.4 Actions taken

Next, we look at the variable 'actions taken'. The variable takes on the following values:

```
## [1] "Establishment Closed by DOHMH.  Violations were cited in the following area(s) and those requir
## [2] "Establishment re-closed by DOHMH"
## [3] "Establishment re-opened by DOHMH"
## [4] "No violations were recorded at the time of this inspection."
## [5] "Violations were cited in the following area(s)."
```

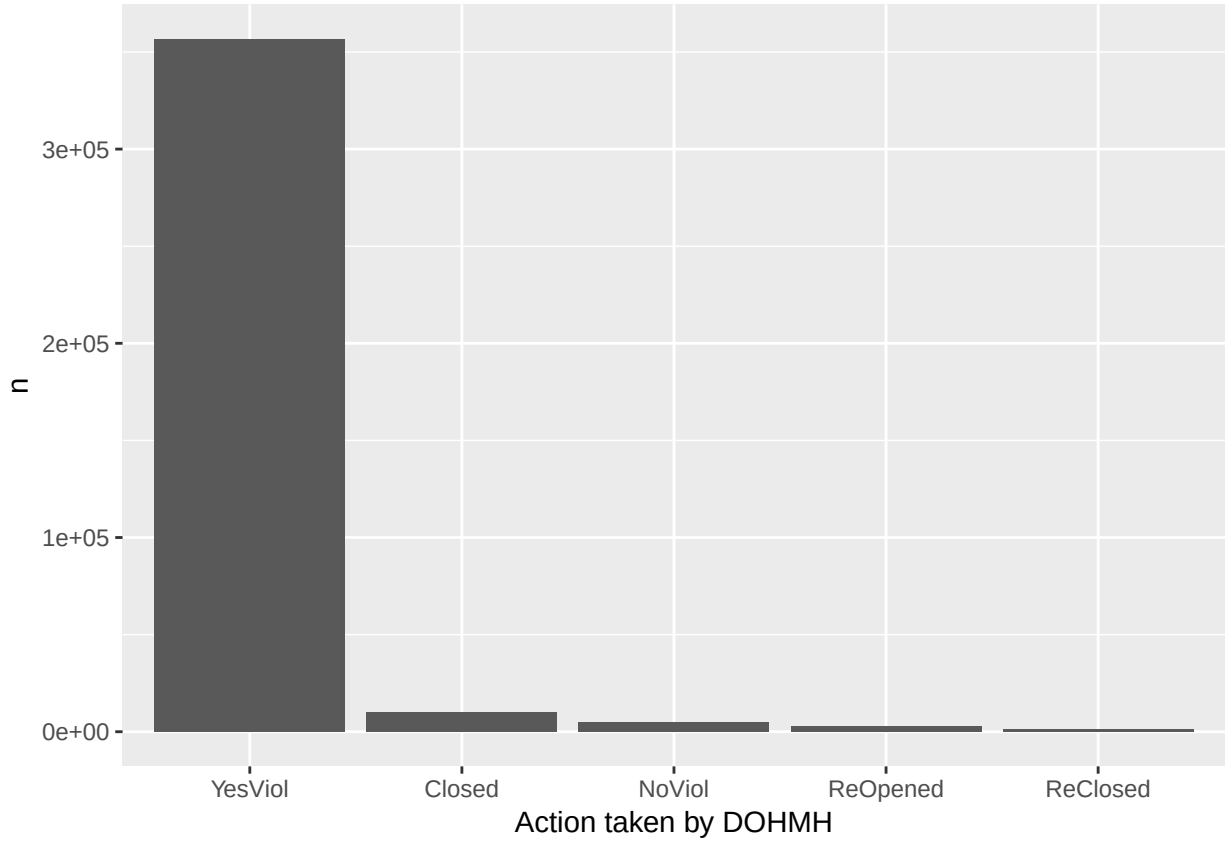
We convert it into a factor and (re)name its levels for convenience:

```
restaurants_raw$action <- as.factor(restaurants_raw$action)
levels(restaurants_raw$action) <- c("Closed", "ReClosed", "ReOpened", "NoViol", "YesViol")
summary(restaurants_raw$action)
```

```
##   Closed ReClosed ReOpened   NoViol  YesViol
##    10145     1116     2973     4898   356657
```

Now we can see what the most frequent categories are:

```
## # A tibble: 5 x 2
##   action        n
##   <fct>     <int>
## 1 YesViol  356657
## 2 Closed    10145
## 3 NoViol     4898
## 4 ReOpened   2973
## 5 ReClosed   1116
```



The most frequent category is 'YesViol'.

2.5 Violation types

Next, we will consider the different types of violations recorded. The most common violation codes are listed below:

```
restaurants_raw$violation_code <- as.factor(restaurants_raw$violation_code)
restaurants_raw %>%
  group_by(violation_code) %>%
  summarise(number = n()) %>%
  mutate(rank = rank(desc(number))) %>%
  filter(rank <= 10) %>%
  arrange(rank)
```

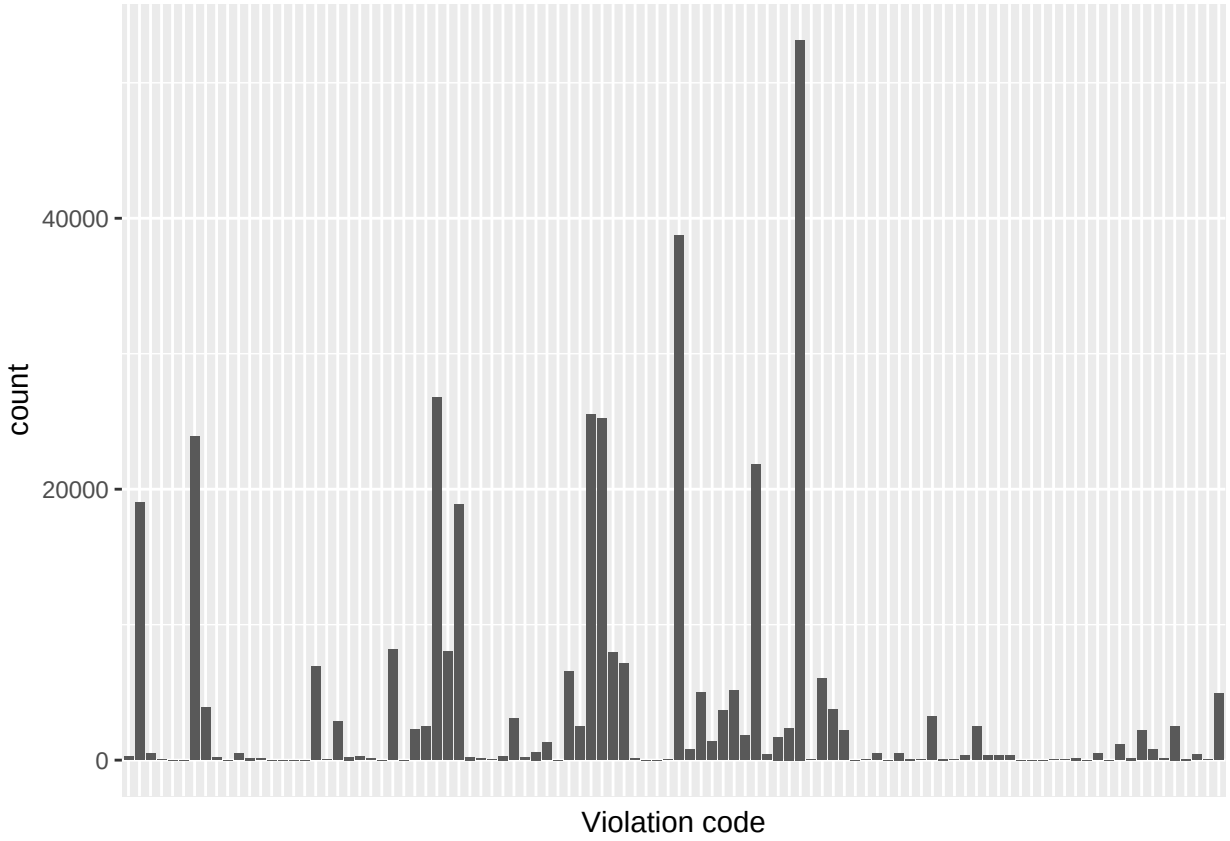
```
## # A tibble: 10 x 3
```

```
##    violation_code number  rank
##    <fct>           <int> <dbl>
##  1 10F             53154     1
##  2 08A             38706     2
##  3 04L             26768     3
##  4 06C             25521     4
##  5 06D             25215     5
##  6 02G             23866     6
##  7 10B             21819     7
##  8 02B             19023     8
##  9 04N             18882     9
## 10 04H              8174    10
```

10F (general violation pertaining to non-food contact surfaces) is the most common violation type. followed by 08A (facility not vermin proof), 04L (Evidence of mice).

Here is a quick plot of the distribution:

```
restaurants_raw %>%
  ggplot(aes(x = violation_code)) +
  geom_bar() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank()) +
  xlab("Violation code")
```



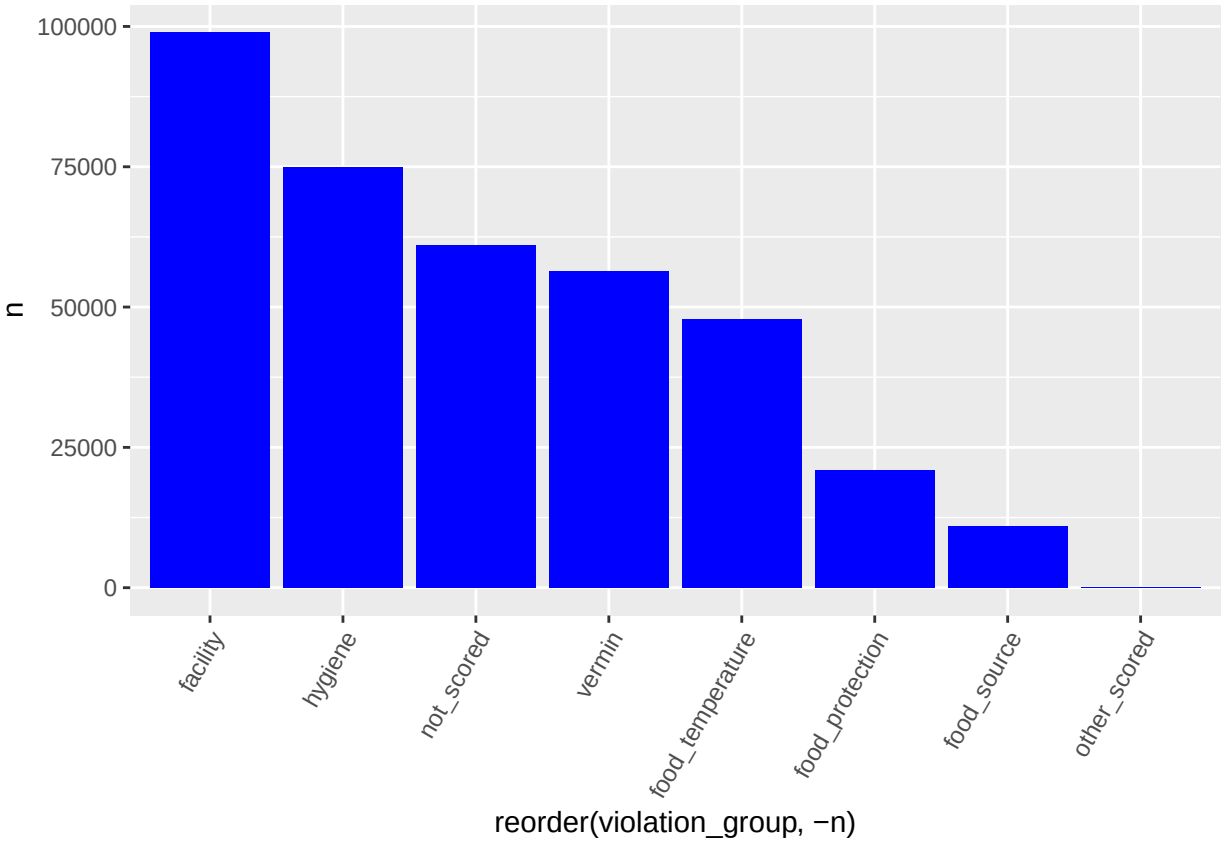
To get a better sense of what is going on, we group the values in the range of violation_code into broader categories. To do this, we use information from the provided by the authorities. Accordingly, violations

are either scored or unscored. Among scored violations, we have to distinguish between critical and general violations.

```
# New variable violation_group:
restaurants_raw <-
  restaurants_raw %>%
  mutate(violation_group = case_when(
    violation_code %in% str_c("02", LETTERS[1:10]) ~ "food_temperature",
    violation_code %in% c(str_c("03", LETTERS[1:7]), str_c("09", LETTERS[1:3])) ~ "food_source",
    violation_code %in% str_c("04", LETTERS[1:10]) ~ "food_protection",
    violation_code %in% c(str_c("05", LETTERS[1:9]), str_c("10", LETTERS[1:10])) ~ "facility",
    violation_code %in% str_c("06", LETTERS[1:9]) ~ "hygiene",
    violation_code %in% str_c("04", LETTERS[11:15]) ~ "vermin",
    violation_code %in% c("07A", "99B") ~ "other_scored",
    !is.na(violation_code) ~ "not_scored"
  )
)
```

Here is a bar plot displaying the distribution

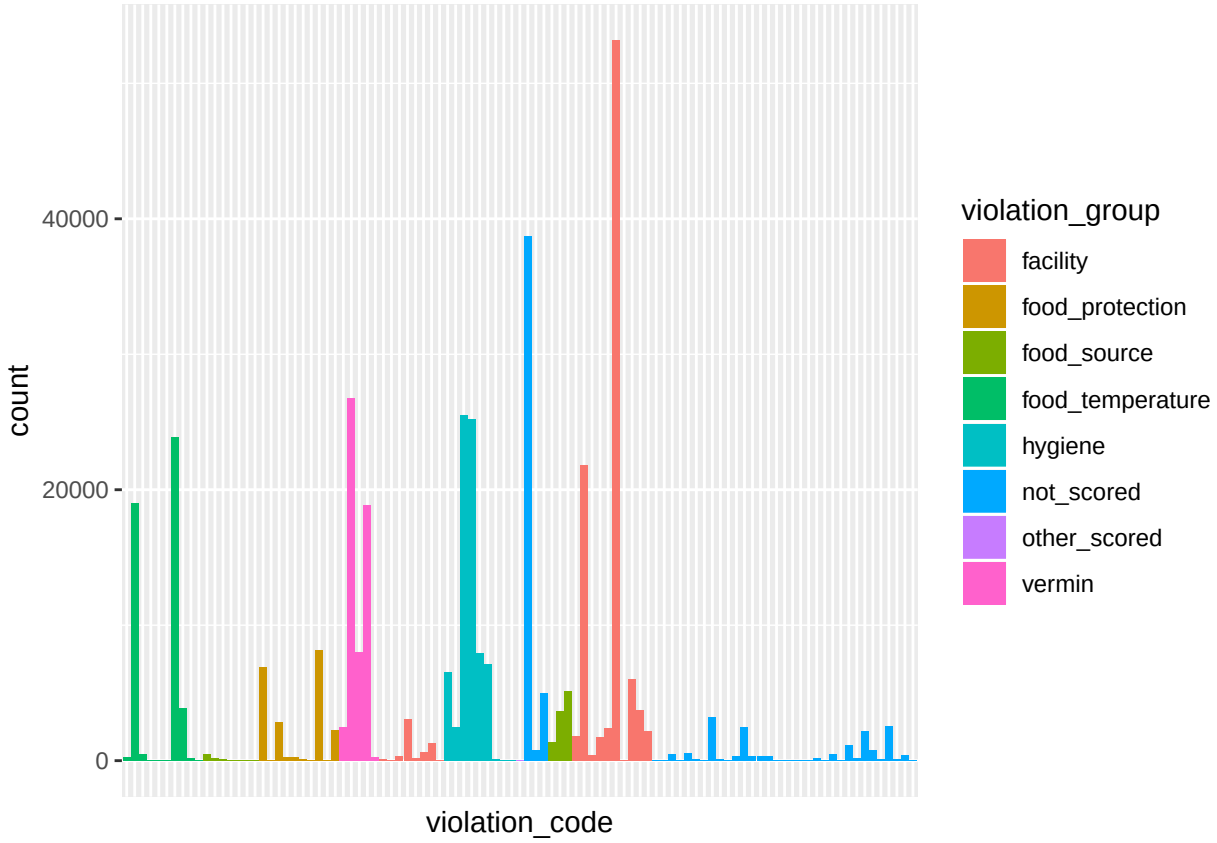
```
restaurants_raw %>%
  group_by(violation_group) %>%
  filter(!is.na(violation_group)) %>%
  summarize(n = n()) %>%
  ggplot(aes(reorder(violation_group, -n), n)) +
  geom_bar(stat = "identity", fill = "blue") +
  theme(axis.text.x = element_text(angle=60, hjust=1))
```



Some categories barely contain any observations. For the time being, we will keep the chosen binning. We are going to revisit this point in the context of feature engineering.

Here is the distribution of violation code again. This time we incorporate the variable violation group.

```
restaurants_raw %>%
  filter(!is.na(violation_group)) %>%
  ggplot(aes(x = violation_code, fill = violation_group)) +
  geom_bar() +
  theme(axis.text.x = element_blank(),
        axis.ticks.x = element_blank())
```



For later use, we construct indicator variables for the different violation groups:

```
for(level in unique(restaurants_raw$violation_group)){
  if (is.na(level)) {
    next
  } else {
    restaurants_raw[paste("viol", level, sep = "_")] <-
    ifelse(restaurants_raw$violation_group == level, 1, 0)
  }
}
names(restaurants_raw)
```

```
##  [1] "id"                    "rest_name"
##  [3] "boro"                  "building"
##  [5] "street"                "zipcode"
##  [7] "phone"                 "cuisine_descr"
##  [9] "inspection_date"       "action"
## [11] "violation_code"        "violation_descr"
## [13] "critical_flag"         "score"
## [15] "grade"                 "grade_date"
## [17] "record_date"           "inspection_type"
## [19] "violation_group"       "viol_vermin"
## [21] "viol_not_scored"       "viol_facility"
## [23] "viol_food_temperature" "viol_hygiene"
## [25] "viol_food_protection"  "viol_food_source"
## [27] "viol_other_scored"
```

2.6 Inspection types

The variable inspection type takes on 34 distinct values:

```
restaurants_raw %>%
     summarise(n_distinct(inspection_type))
```

```
## # A tibble: 1 x 1
##   `n_distinct(inspection_type)`
##                           <int>
## 1                            34
```

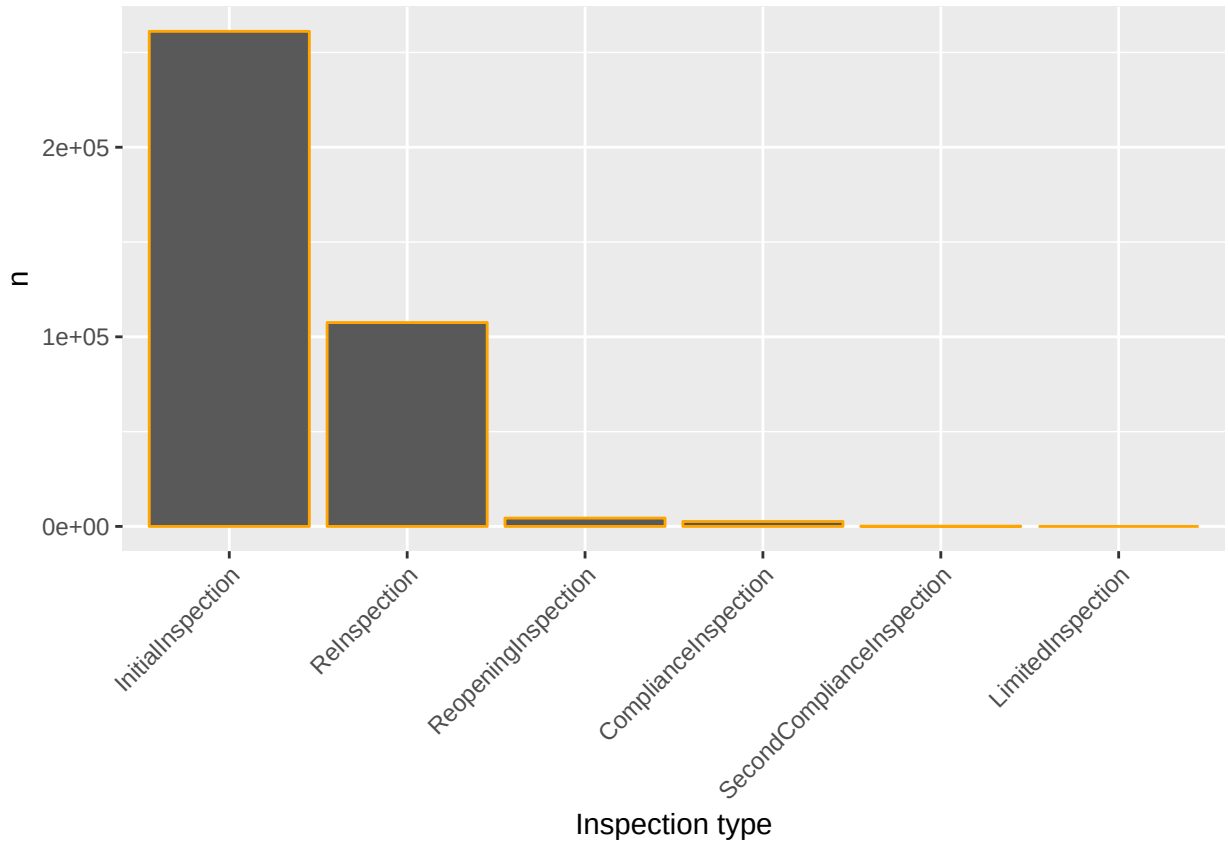
These are the values:

```
## # A tibble: 34 x 1
##    inspection_type
##    <chr>
##  1 Administrative Miscellaneous / Compliance Inspection
##  2 Administrative Miscellaneous / Initial Inspection
##  3 Administrative Miscellaneous / Re-inspection
##  4 Administrative Miscellaneous / Reopening Inspection
##  5 Administrative Miscellaneous / Second Compliance Inspection
##  6 Calorie Posting / Compliance Inspection
##  7 Calorie Posting / Initial Inspection
##  8 Calorie Posting / Re-inspection
##  9 Calorie Posting / Second Compliance Inspection
## 10 Cycle Inspection / Compliance Inspection
## # ... with 24 more rows
```

We collapse these values into a smaller range:

```
## [1] "ReInspection"               "InitialInspection"
## [3] "ComplianceInspection"       "ReopeningInspection"
## [5] "SecondComplianceInspection" "LimitedInspection"
```

Here is a display of the distribution:



Again, there a are categories that are 'almost empty'. The vast majority of observations are initial and re-inspections:

```
## # A tibble: 6 x 3
##   inspection_type2               n    prop
##   <chr>                      <int>   <dbl>
## 1 InitialInspection         261126 0.695
## 2 ReInspection              107536 0.286
## 3 ReopeningInspection         4388 0.0117
## 4 ComplianceInspection        2601 0.00692
## 5 SecondComplianceInspection   134 0.00036
## 6 LimitedInspection              4 0.00001
```

For later purposes, we construct a dummy variable for the category 'InitialInspection'.

```
##  [1] "id"                    "rest_name"
##  [3] "boro"                  "building"
##  [5] "street"                "zipcode"
##  [7] "phone"                 "cuisine_descr"
##  [9] "inspection_date"       "action"
## [11] "violation_code"        "violation_descr"
## [13] "critical_flag"         "score"
## [15] "grade"                 "grade_date"
## [17] "record_date"           "inspection_type"
## [19] "violation_group"       "viol_vermin"
## [21] "viol_not_scored"       "viol_facility"
```

```
## [23] "viol_food_temperature"   "viol_hygiene"
## [25] "viol_food_protection"    "viol_food_source"
## [27] "viol_other_scored"       "inspection_type2"
## [29] "dummy_InitialInspection"
```

2.7 Grade

Now on to 'grade'.

```
## # A tibble: 7 x 2
##   grade                n
##   <chr>            <int>
## 1 <NA>            187702
## 2 A               149897
## 3 B                24696
## 4 C                 6318
## 5 Z                 3510
## 6 Not Yet Graded    1915
## 7 P                 1751
```

There is a massive amount of NA's. We are going to investigate how the number of NAs for 'grade' relates to 'score' and 'inspection_type.' More precisely, we know that on initial inspections no grade is issued, if the corresponding score is above 14. It seems plausible that this would lead to a value of NA for grade.

```
restaurants_raw %>%
  select(inspection_type2, score, grade) %>%
  filter(inspection_type2 == "InitialInspection", score > 14) %>%
  group_by(grade) %>%
  summarise(n = n())
```

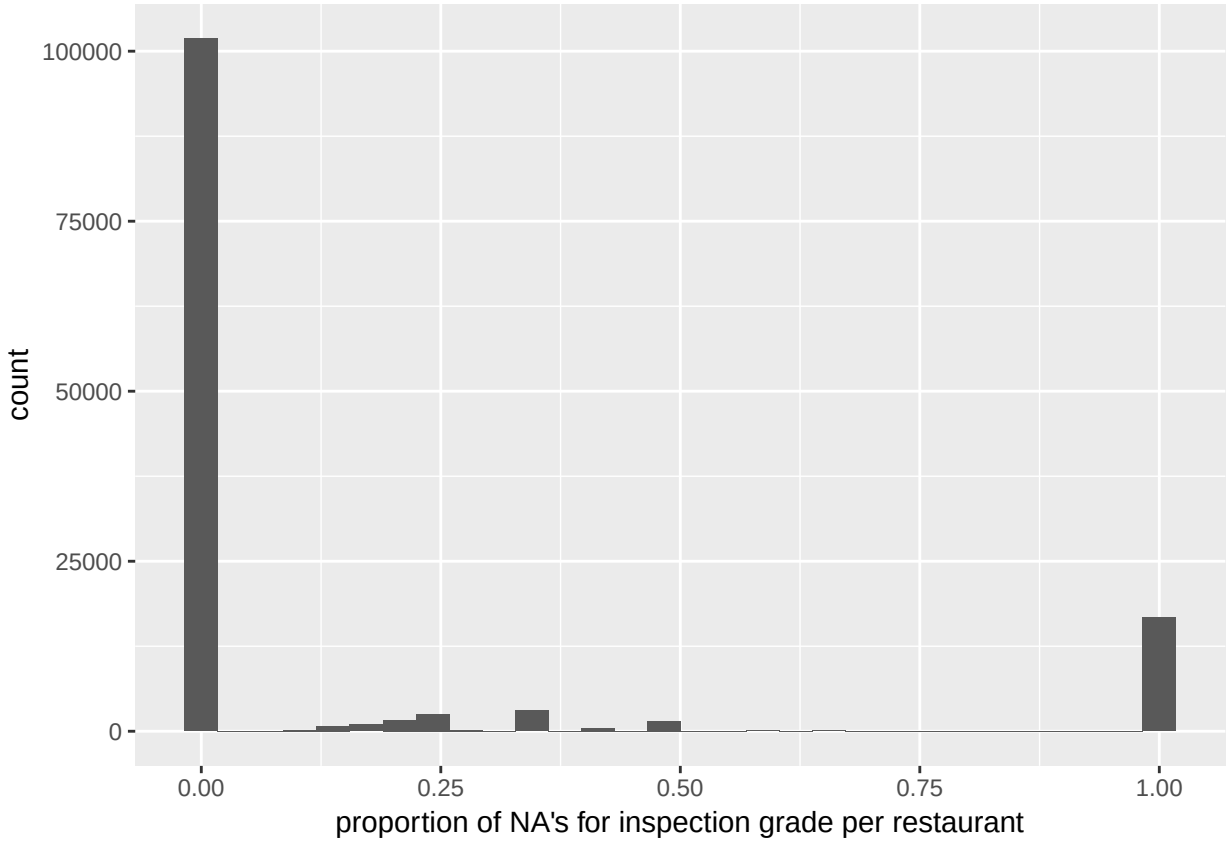
```
## # A tibble: 2 x 2
##   grade                n
##   <chr>            <int>
## 1 Not Yet Graded    1548
## 2 <NA>            137410
```

So about 73 % of the missing values are due to a score of 14 or higher. Furthermore, out of the 1915 observations with value 'Not yet graded', 1548 are the result of a high score (above 13) in the initial inspection. We will recode these NA's to "Not Yet Graded" to reduce the number of missing values for 'grade'. This reduces the number of NA's significantly:

```
restaurants_raw %>%
  summarise(prop_na = round(mean(is.na(grade)), 2),
            n_not_graded = sum(grade == "Not Yet Graded", na.rm = T))
```

```
## # A tibble: 1 x 2
##   prop_na n_not_graded
##     <dbl>        <int>
## 1    0.13       139325
```

Here is the distribution of the proportion of inspections with an NA for grade per restaurant:



It seems that for most restaurants there is no inspection with a missing value for grade. On the other hand, there are some restaurants for whom all values for grade are NA.

```
## # A tibble: 39 x 2
##    prop_gradeNA proportion_of_restaurants
##           <dbl>                     <dbl>
##  1        0                         0.781
##  2        1                         0.128
##  3        0.333                     0.024
##  4        0.25                      0.02
##  5        0.2                       0.012
##  6        0.5                       0.011
##  7        0.167                     0.008
##  8        0.143                     0.004
##  9        0.4                       0.003
## 10        0.125                     0.002
## # ... with 29 more rows
```

It would be worth to further investigate the nature of the NA's related grade, something which will not pursue any further at this point. For later purposes, we convert 'grade' into a factor:

```
restaurants_raw <- restaurants_raw %>%
  mutate(grade = as.factor(grade))
```

2.8 Score

Let us turn to 'score'.

```
summary(restaurants_raw$score)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   -2.00   11.00   14.00   18.93   24.00  151.00   19514
```

There are 19514 NA's. So let us look at the proportion of inspections that exhibit an NA for 'score'.

```
## # A tibble: 1 x 1
##    n_na
##   <dbl>
## 1  0.15
```

15 % of all inspections have a missing value for 'score'.

We know that there are 16530 observations that refer to violations that are not scored:

```
## # A tibble: 1 x 2
##   violation_group     n
##   <chr>           <int>
## 1 not_scored      61028
```

These observations are expected to have NA for 'score':

```
## # A tibble: 1 x 3
##       n n_na_score prop_na_score
##   <int>      <int>         <dbl>
## 1 61028      16086          0.26
```

Hence, these observations account for 82 % of all of the 19514 NA's for 'score'. What about the remaining 3428 missing values for 'score'? Let us consider the relationship with violation group:

```
## # A tibble: 1 x 2
##       n n_na_score
##   <int>      <int>
## 1  4921       3428
```

Hence, the rest of the NA's for score are associated with NA's on 'violation_group'. So let us turn to the NA's for the variable 'violation_group'. One cause for an NA here may be that on a given inspection, there was no violation at all. So what is the number of NA's for violation group among observations that do not exhibit a violation?

```
## # A tibble: 1 x 1
##   viol_group_NA
##           <int>
## 1          4634
```

3. Predicting time until the next inspection

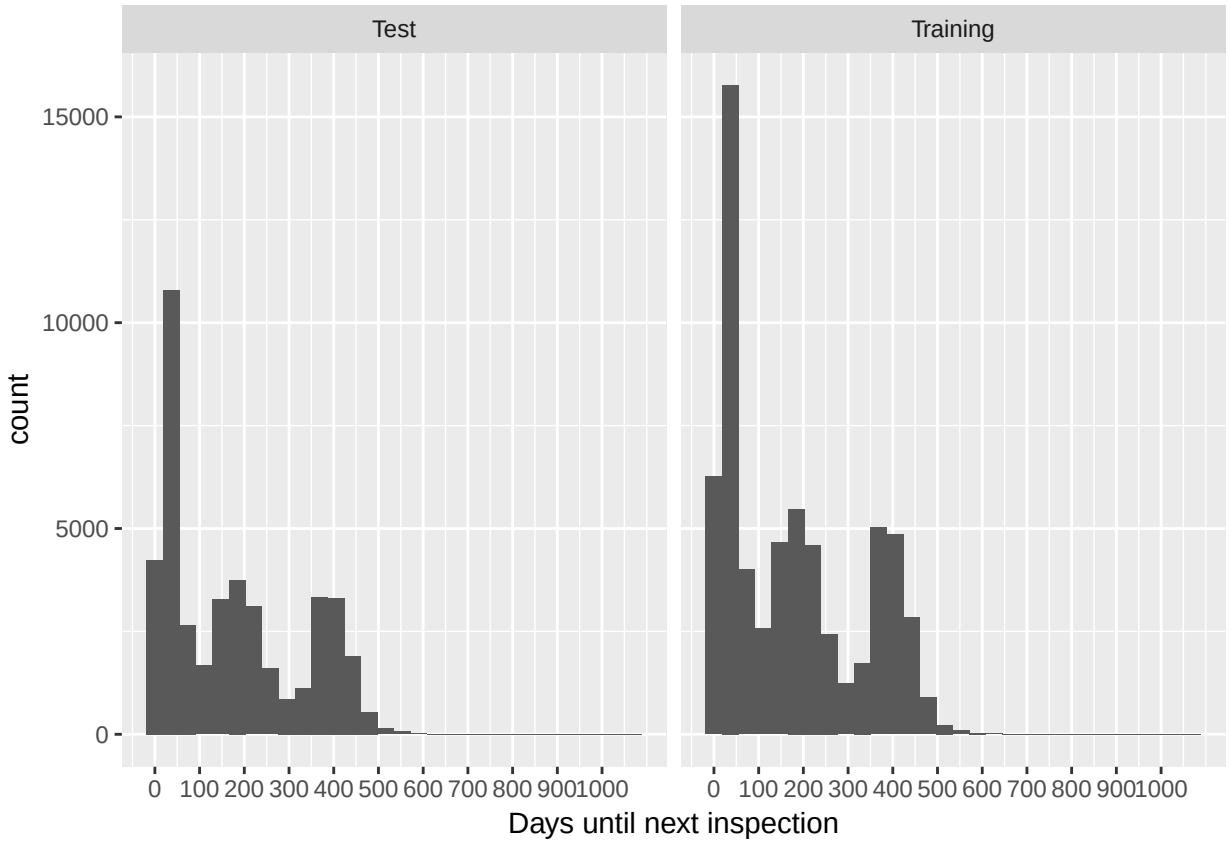
Finally, we now return to the question posed in the beginning, namely, whether it is possible to predict the number of days until the next inspection, using features of the data, described above.

3.1 Split into training and test data

As a first step, we will divide our data set into a training and a test data set, putting 2/3 of the restaurants into the training set.

```
set.seed(1)
train_ids <- sample(unique(restaurants_raw$id), 0.6*length(unique(restaurants_raw$id)))
test_ids <- setdiff(unique(restaurants_raw$id), train_ids)
```

Next, let us see if the distributions of the target feature in training and test data are similar:



The distribution of the target seems reasonably similar across training and test data.

3.2 Features

In the following we will use regression trees to predict the number of days until the next inspection, using the score and grade on the last inspection, the type of cuisine and number of violations in the different violation groups on the last inspection. Furthermore, we will impute missing values for score and grade using regression trees, including only those observations for which the respective values are not missing.

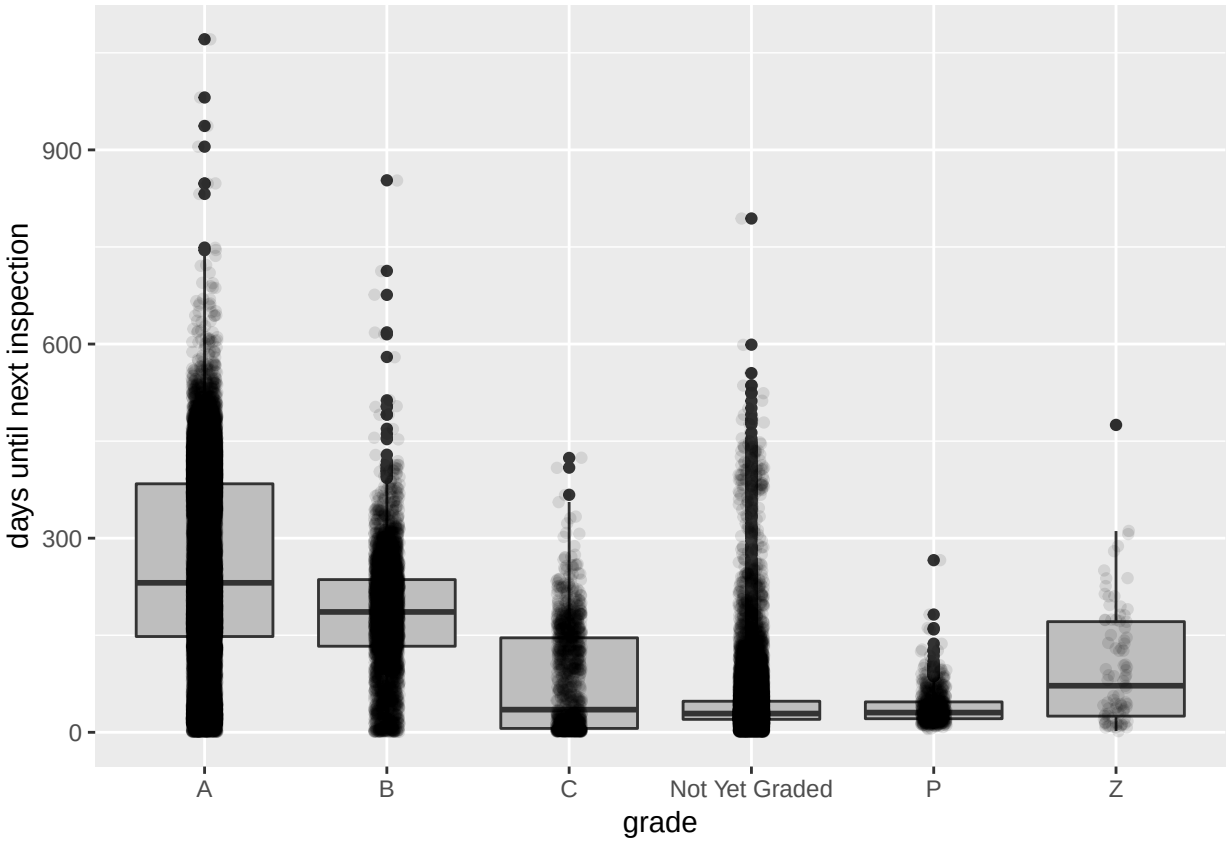
```
library(rpart)
df_features <- make_feature_set(restaurants_raw, is_train = TRUE) %>% impute_features()

df_features
```

```
## # A tibble: 105,166 x 17
##         id inspection_date grade    dummy_InitialInsp~ score cuisine_descr
##      <dbl> <date>          <fct>    <fct>              <dbl> <fct>
##  1  3.01e7 2015-02-09      A        Initial                6 Bakery
##  2  3.01e7 2016-02-18      A        Initial               10 Bakery
##  3  3.01e7 2015-05-07      A        Initial               12 Hamburgers
##  4  3.01e7 2016-04-12      A        Initial                0 Hamburgers
##  5  3.01e7 2016-04-30      A        Not_Initial           13 Hamburgers
##  6  3.01e7 2016-10-03      Not Yet~ Initial               48 Hamburgers
##  7  3.01e7 2016-10-27      A        Not_Initial           11 Hamburgers
##  8  3.01e7 2017-06-26      A        Initial                7 Hamburgers
##  9  3.01e7 2017-10-06      A        Not_Initial           10 Hamburgers
## 10  3.02e7 2015-08-31      A        Initial               12 Other
## # ... with 105,156 more rows, and 11 more variables: viol_vermin <dbl>,
## #   viol_not_scored <dbl>, viol_facility <dbl>,
## #   viol_food_temperature <dbl>, viol_hygiene <dbl>,
## #   viol_food_protection <dbl>, viol_food_source <dbl>,
## #   viol_other_scored <dbl>, critical_flag <dbl>, days_until_next <time>,
## #   days_until_next_categ <fct>
```

```
df_train <- df_features %>% filter(id %in% train_ids)
df_test <- df_features %>% filter(id %in% test_ids)
```

3.3 Feature Exploration

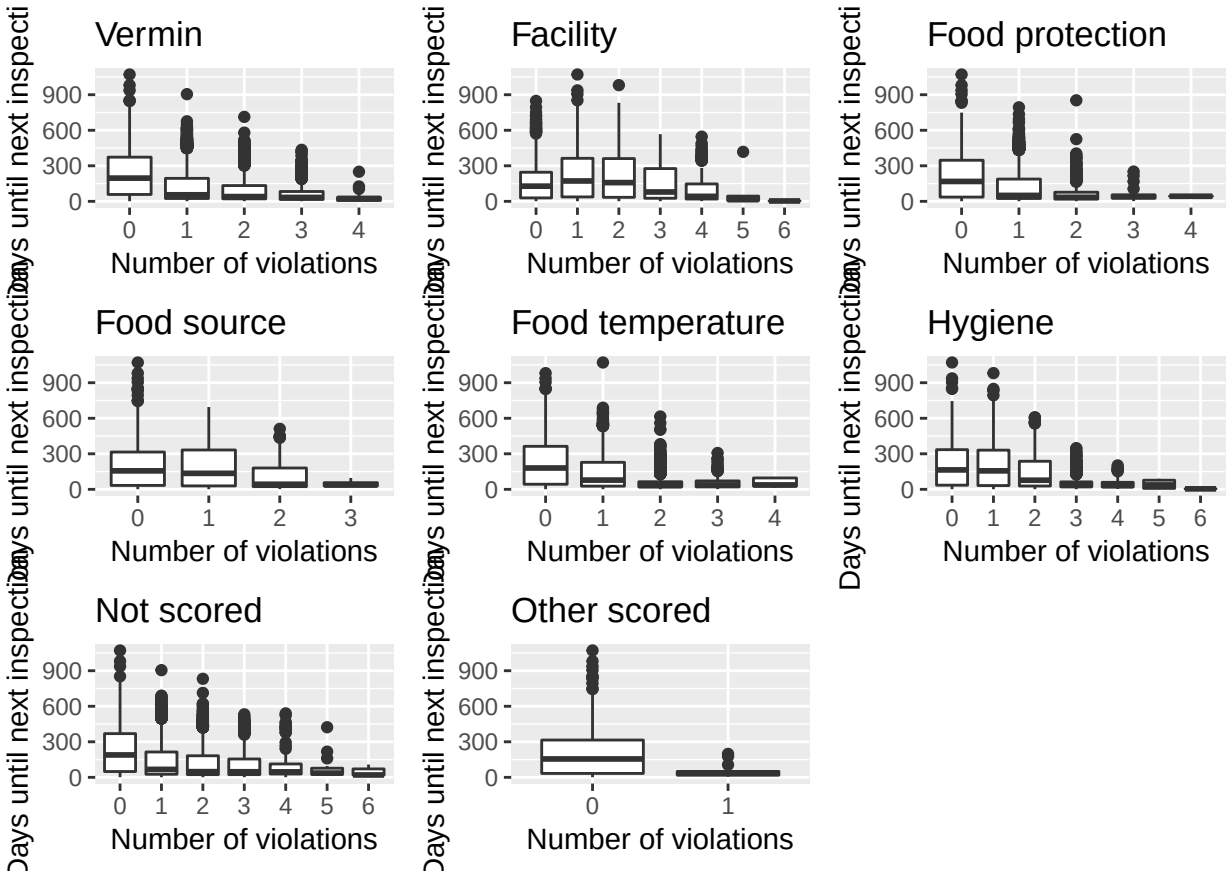
Let us first explore a couple of bivariate relationships between the predictors and the target feature. We will begin with the relationship between grade and days until the next inspection



As we can see, the median number of days until the next inspection decreases when moving from A to C. This is not suprising, since a good grade on the initial inspection reduces the likelihood of being subjected to a re-inspection the same inspection cycle.

Let us move on to the relationship between violation types and the target feature:

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```



Here also, as expected, the median number of days until the next inspection decreases as the number of violations per inspection increases.

3.4 Linear Regression model

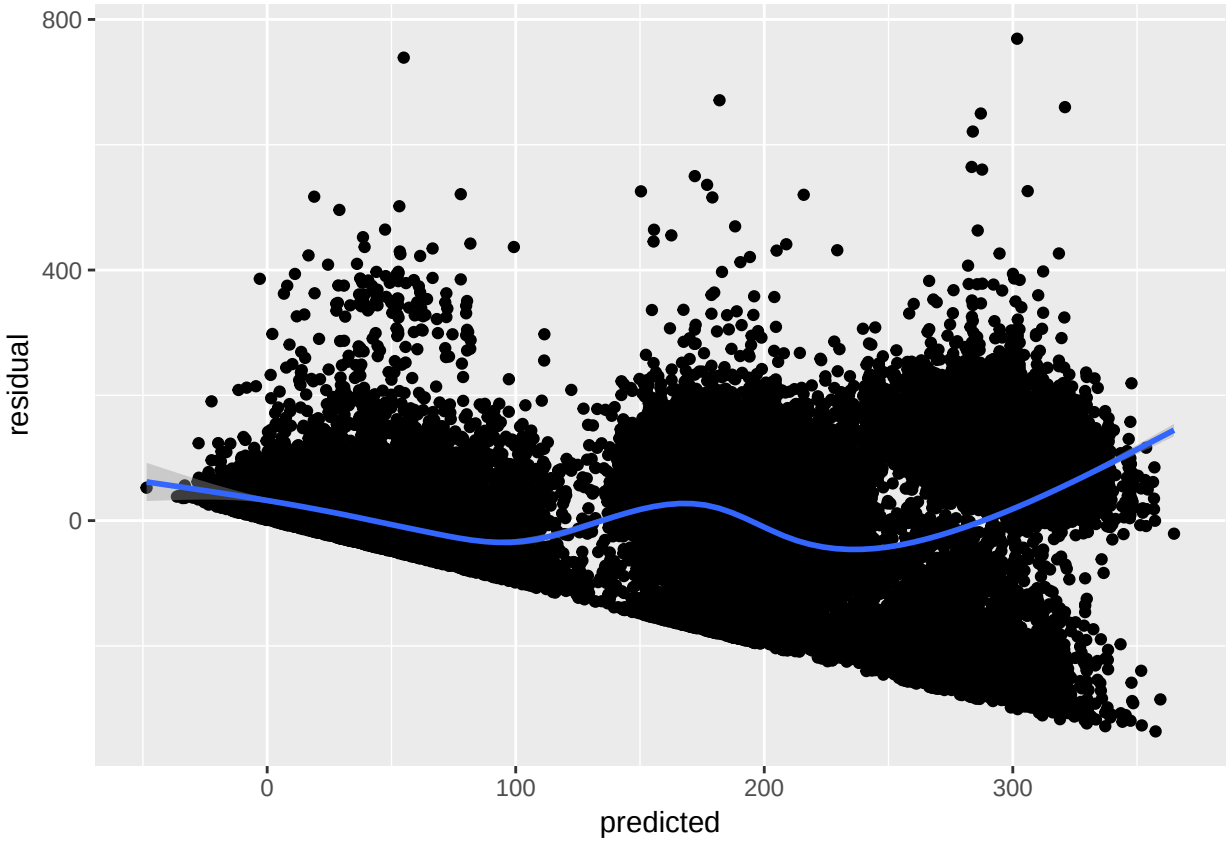
We start with a simple linear regression model, using all features.

```
## Loading required package: lattice

##
## Attaching package: 'caret'

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

## Linear Regression
##
## 62767 samples
##    13 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 62767, 62767, 62767, 62767, 62767, 62767, ...
## Resampling results:
```

```
##
##   RMSE      Rsquared   MAE
##   109.2895  0.4761192  79.92761
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

We see that the R squared associated with the model is 0.476, i.e., 47.6 percent of the variation in the data can be 'explained' in terms of the linear regression model. Let us look at the residual plot:

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



There are a few curious things to notice about the plot. First, apparently there is a set of restaurants for which the model predicts a negative duration until the next inspection, which does not make sense. Second, there seems to be some pattern in the residuals. In the presence of nonlinearities, the conclusions drawn from the fit are questionable. At this point, one could investigate how these patterns in the residuals come about. Instead, in the next section, we will fit a nonlinear regression model to the data.

3.5 Second Model: Regression tree

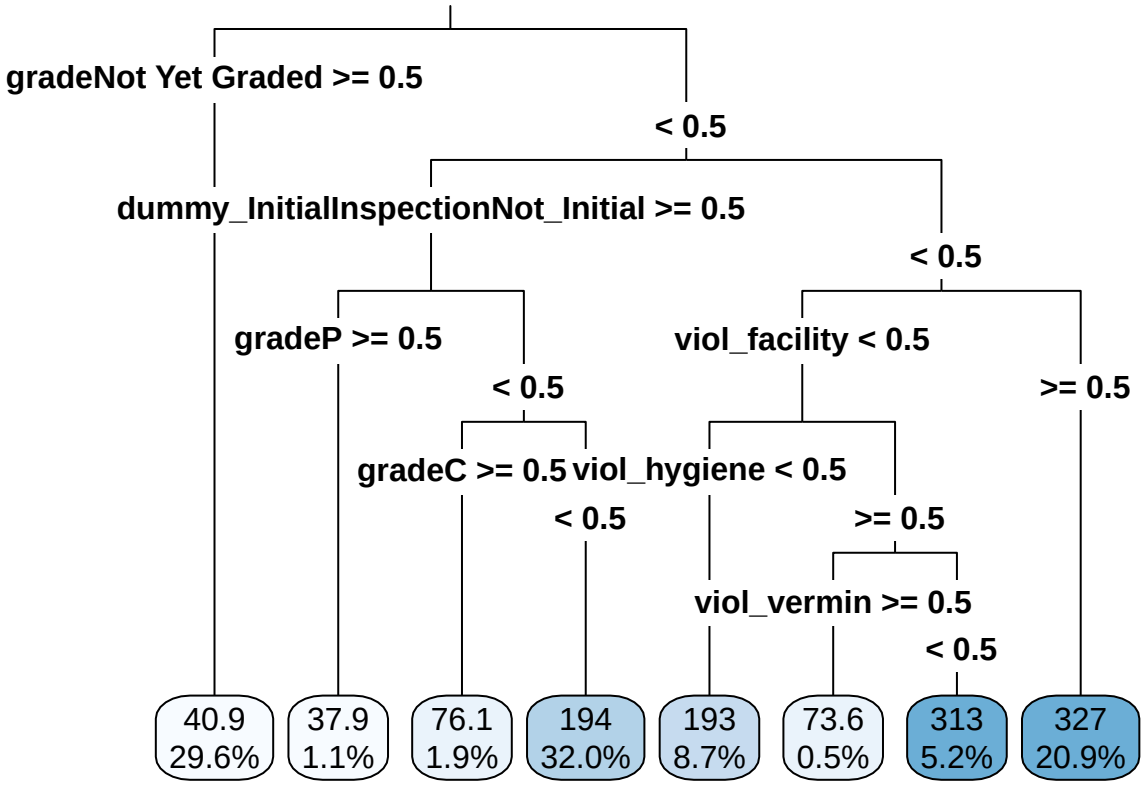
So far, so good. Let us fit at a cross-validated regression tree model, using all the features. First, we set the hyperparameters:

```
# Set hyperparameters for cross-validated classification and regression tree
set.seed(42)
myControl <- trainControl(
```

```
  method = "cv",
  number = 10

)
cp.grid <- expand.grid(.cp = (1:10)*0.01)
```

Then we fit the model, employing 10-fold cross validation:

```
## CART
##
## 62767 samples
##    13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 56490, 56489, 56491, 56491, 56490, 56492, ...
## Resampling results across tuning parameters:
##
##   cp    RMSE      Rsquared   MAE
##   0.01  105.5183  0.5123524  72.32293
##   0.02  110.5336  0.4648770  77.94083
##   0.03  110.5336  0.4648770  77.94083
##   0.04  113.9228  0.4315671  82.14456
##   0.05  113.9228  0.4315671  82.14456
##   0.06  113.9228  0.4315671  82.14456
##   0.07  113.9228  0.4315671  82.14456
##   0.08  113.9228  0.4315671  82.14456
##   0.09  122.2085  0.3458682  93.62115
##   0.10  122.2085  0.3458682  93.62115
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.01.
```



The model has a slightly higher R squared than the linear regression model. Nevertheless, the fit of the regression to the training data is quite poor.

3.6 Reformulating the problem: Classification

As we have seen, linear regression and regression trees did a poor job of predicting the number of days until the next inspection. This is disappointing, but we can rephrase the problem. Instead of predicting the number of days, we could give a time interval into which the next inspection will fall.

3.7 Classification model

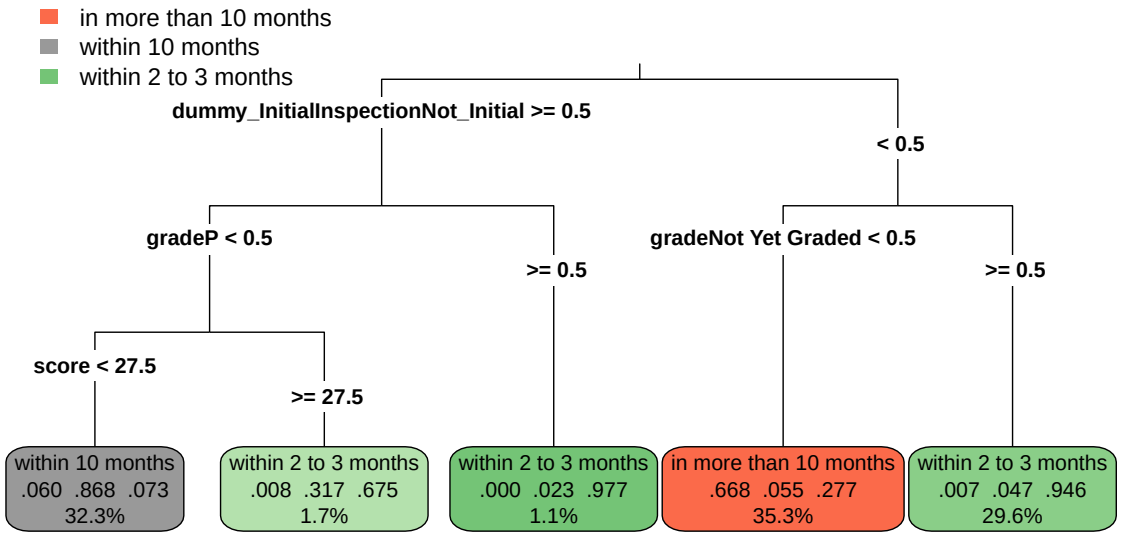
As a next step, we will turn the continuous target into a categorical variable, by dividing the range into bins. The first bin will be the 0 - 100 days, the second 101 - 300 and the third bin more than 300 days. Then we will fit a classification tree to the data, using 10-fold cross validation:

```
set.seed(3333)
(class_tree_model <- df_train %>% select(-days_until_next, -id, -inspection_date) %>%
    train(days_until_next_categ ~ .,
          data = .,
          method = "rpart",
          trControl = myControl,
          tuneGrid = cp.grid)
)
```

```
## CART
```

```
##
## 62767 samples
##    13 predictor
##     3 classes: 'in more than 10 months', 'within 10 months', 'within 2 to 3 months'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 56491, 56490, 56490, 56490, 56490, 56491, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy   Kappa
##   0.01  0.8190449  0.7292110
##   0.02  0.8022049  0.7055543
##   0.03  0.8022049  0.7055543
##   0.04  0.8022049  0.7055543
##   0.05  0.8022049  0.7055543
##   0.06  0.8022049  0.7055543
##   0.07  0.8022049  0.7055543
##   0.08  0.8022049  0.7055543
##   0.09  0.8022049  0.7055543
##   0.10  0.8022049  0.7055543
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01.
```

```
rpart.plot ( class_tree_model$finalModel, type = 3, digits = 3, fallen.leaves = TRUE )
```



As we can see, the accuracy of the model is 0.819. Let us look at additional measures of performance:

```
## Confusion Matrix and Statistics
##
##                          Reference
## Prediction                in more than 10 months within 10 months
##   in more than 10 months                  14806             1212
##   within 10 months                         1209            17593
##   within 2 to 3 months                      130             1219
##                          Reference
## Prediction                within 2 to 3 months
##   in more than 10 months                 6138
##   within 10 months                       1471
##   within 2 to 3 months                  18989
##
## Overall Statistics
##
##                Accuracy : 0.8187
##                  95% CI : (0.8157, 0.8217)
##     No Information Rate : 0.4238
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.729
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: in more than 10 months Class: within 10 months
## Sensitivity                                 0.9171                  0.8786
## Specificity                                 0.8423                  0.9373
## Pos Pred Value                              0.6683                  0.8678
## Neg Pred Value                              0.9670                  0.9428
## Prevalence                                  0.2572                  0.3190
## Detection Rate                              0.2359                  0.2803
## Detection Prevalence                        0.3530                  0.3230
## Balanced Accuracy                           0.8797                  0.9079
##                      Class: within 2 to 3 months
## Sensitivity                               0.7139
## Specificity                               0.9627
## Pos Pred Value                            0.9337
## Neg Pred Value                            0.8207
## Prevalence                                0.4238
## Detection Rate                            0.3025
## Detection Prevalence                      0.3240
## Balanced Accuracy                         0.8383
```

Sensitivity and specificity of all models are at least 0.84 for all classes, except for the third class, for which sensitiviy is at 0.71. The model

3.8 Evaluation of final model

Finally we evaluate the final model on the test data:

```
## Confusion Matrix and Statistics
##
##                          Reference
## Prediction                in more than 10 months within 10 months
##   in more than 10 months                   9879              824
##   within 10 months                          743            12039
##   within 2 to 3 months                       86              825
##                          Reference
## Prediction                within 2 to 3 months
##   in more than 10 months                 4174
##   within 10 months                        944
##   within 2 to 3 months                  12885
##
## Overall Statistics
##
##                Accuracy : 0.8208
##                  95% CI : (0.8172, 0.8245)
##     No Information Rate : 0.4246
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7321
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: in more than 10 months Class: within 10 months
## Sensitivity                                 0.9226                 0.8795
## Specificity                                 0.8423                 0.9412
## Pos Pred Value                              0.6640                 0.8771
## Neg Pred Value                              0.9699                 0.9425
## Prevalence                                  0.2526                 0.3228
## Detection Rate                              0.2330                 0.2839
## Detection Prevalence                        0.3509                 0.3237
## Balanced Accuracy                           0.8824                 0.9104
##                      Class: within 2 to 3 months
## Sensitivity                               0.7157
## Specificity                               0.9627
## Pos Pred Value                            0.9340
## Neg Pred Value                            0.8211
## Prevalence                                0.4246
## Detection Rate                            0.3039
## Detection Prevalence                      0.3254
## Balanced Accuracy                         0.8392
```

Hence, the performance of the model on the test data is similar than on the training data.

3.9 Conclusion

Our initial goal to predict the number of days until the next inspection could not be attained. Instead, we rephrased the problem and turned it into a classification problem. Then, we fit a classification tree on the training data, employing cross validation. The resulting model also performed well on the test data. So eventually, we have provided our client with a device that predicts a time window the next inspection is likely to occur in.