MSAN 593: Assignment 2

[redacted]

July 29, 2016

```
if (!require("tidyr")) {
  install.packages("tidyr",repos="http://cran.rstudio.com/")
}

if (!require("dplyr")) {
  install.packages("dplyr",repos="http://cran.rstudio.com/")
}

if (!require("knitr")) {
  install.packages("knitr",repos="http://cran.rstudio.com/")
}

if (!require("ggplot2")) {
  install.packages("ggplot2",repos="http://cran.rstudio.com/")
}

if (!require("lubridate")) {
  install.packages("lubridate",repos="http://cran.rstudio.com/")
}

if (!require("maps")) {
  install.packages("maps",repos="http://cran.rstudio.com/")
}
```

```
library(knitr)
library(tidyr)
library(dplyr)
library(ggplot2)
library(lubridate)
library(maps)
```

```
opts_chunk$set(tidy.opts=list(width.cutoff=60))
opts_chunk$set(tidy = TRUE)
```

Question 2

Section 1: Data Import and Initial Review

We have a data set that needs to be cleaned. It is imported below as **df**. Columns are renamed with more concise denominators for convenience.

```
# import data with '', 'NA', and 'na' as NA
df <- read.csv("hw2.csv", header = TRUE, stringsAsFactors = FALSE,
    na.strings = c("", "NA", "na"))
```

```
# Rename columns with especially long names
df <- df %>%
  rename(Opportunity.Thru=Opportunity..Purchased.Thru) %>%
  rename(Utility.Company=Opportunity..Utility.Company) %>%
  rename(Jurisdiction.Name=
          Opportunity..Jurisdiction..Jurisdiction.Name) %>%
  rename(System.Size=Proposal..System.Size.STC.DC) %>%
  rename(Service.Contract=
           Service.Contract..Service.Contract.Event..Using.Build.Partner.) %>%
  rename(Opportunity.Upgrade=Opportunity..Service.Panel.Upgrade) %>%
  rename(Opportunity.Reroof=Opportunity..Reroof.under.Array) %>%
  rename(Opportunity.HOA=Opportunity..HOA.) %>%
  rename(PEStamp.Required=Opportunity..PE.Stamp.Required.)
```

To get a quick sense of the data, we use the **dplyr** function **glimpse()**.

```
glimpse(df)
```

```
## Observations: 49,880
## Variables: 14
## $ Project.Name        <chr> "PR-1631525512", "PR-1727346069", "PR-1321...
## $ Created.Date        <chr> "11/19/15", "5/31/15", "1/25/16", "11/19/1...
## $ Project.Status      <chr> "Open", "Cancelled", "Open", "Open", "Comp...
## $ Opportunity.Thru    <chr> "Costco", NA, NA, "Costco", NA, "Standard ...
## $ Agreement.Type      <chr> "Customer Owned - Full Upfront", "Custom P...
## $ Install.Branch      <chr> "Sacramento", "Las Vegas", "MD - Columbia"...
## $ Utility.Company     <chr> "PG&E", "NV Energy South", "BG&E", "SCE", ...
## $ Jurisdiction.Name   <chr> "CA-CITY CHICO", "NV-COUNTY CLARK", "MD-CO...
## $ System.Size         <dbl> 5.565, 5.720, 8.480, 5.830, 7.540, 4.240, ...
## $ Service.Contract    <int> 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...
## $ Opportunity.Upgrade <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Opportunity.Reroof  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ Opportunity.HOA     <chr> "No", "Yes", "Yes", "No", NA, "No", "No", ...
## $ PEStamp.Required    <int> 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, ...
```

To further characterize the data set and gain insight into the type of data contained in each variable (e.g. continuous numerical, boolean factor, categorical factor, etc.) as well as the completeness of the data, we make Table 1 summarizing type, class, number of *NA* entries, and number of unique entries.

```
df.type <- summarise_each(df, funs(typeof))
df.class <- summarise_each(df, funs(class))
df.unique <- summarise_each(df, funs(n_distinct))
df.numNA <- summarise_each(df, funs(sum(is.na(.))))
cap1 = "Type, class, number of unique, and number of NA entries in original data set"
kable(cbind(variable = names(df.unique), type = t(df.type)[,
    1], class = t(df.class)[, 1], unique = t(df.unique)[, 1],
    na = t(df.numNA)[, 1]), row.names = FALSE, caption = cap1)
```

Table 1: Type, class, number of unique, and number of NA entries in original data set

variable	type	class	unique	na
Project.Name	character	character	49874	1
Created.Date	character	character	368	6
Project.Status	character	character	5	42
Opportunity.Thru	character	character	7	16372
Agreement.Type	character	character	14	7309
Install.Branch	character	character	38	231
Utility.Company	character	character	63	240
Jurisdiction.Name	character	character	1317	13719
System.Size	double	numeric	424	190
Service.Contract	integer	integer	3	6
Opportunity.Upgrade	integer	integer	3	6
Opportunity.Reroof	integer	integer	3	6
Opportunity.HOA	character	character	6	5083
PEStamp.Required	integer	integer	3	6

Based on the **glimpse()** result and Table 1, we now go variable-by-variable and discuss initial insights on the data set.

Section 1.1: Project.Name Based on the fact that there are nearly as many unique entries as there are rows of data, **Project.Name** is probably a unique identifier for each observation in the dataset. Looking at the output from **glimpse()**, it seems as though there is intended to be a standard format for each project name, i.e.*PR-1234567890*.

Section 1.2: Created.Date **Created.Date** has a large number of entries and few rows of missing data. As is implied by the name, it is likely intended to be a date-type varible specifying the chronological occurence of each observation.

Section 1.3: Project.Status **Project.Status** has only 5 unique entries and relatively few rows of missing data. Based on this information, and the example entries from **glimpse()**, we surmise that **Project.Status** is a classifier with a small set of possible states.

Section 1.4: Opportunity.Thru Based on the entries in **glimpse()** and the column name, **Opportunity.Thru** likely specifies how a project was referred to the company in question. Considering the large number of *NA* values in this data set, it could be the case that the **Opportunity.Thru** information is sometimes hard to ascertain.

Section 1.5: Agreement.Type Similar to **Project.Status**, **Agreement.Type** appears to be a classifier with a small number of possible values. The large number of *NA* values could suggest that the choice of categories for this column need improvement to make collecting complete data easier.

Section 1.6: Install.Branch Reviewing the entries in **glimpse()**, it seems that **Install.Branch** specifies a location associated with the project. It has a large number of unique entries.

Section 1.7: Utility.Company **Utility.Company** appears to be similar to **Install.Branch** except that it specifies a company associated with the project.

Section 1.8: Jurisdiction.Name **Jurisdiction.Name** seems to specify another location associated with the project, but unlike **Install.Branch**, it follows a standard format, i.e. *AB-CITYNAME*. Interestingly, **Jurisdiction.Name** has a large number of NA entries. It's unclear whether this is due to incomplete data collection or if it's because there is not an associated jurisdiction in all cases.

Section 1.9: System.Size **System.Size** is a numeric class column with a large number of unique entries. It is almost certainly a measurement of size on a continuous numeric scale.

Section 1.10: Service.Contract **Service.Contract** is integer type with only three unique entries (including *NA*). In **glimpse()** we see only entries of *0* or *1*. We therefore surmise that it is a boolean.

Section 1.11: Opportunity.Upgrade Using the same reasoning as Section 1.10, we conclude that **Opportunity.Upgrade** is a boolean.

Section 1.12: Opportunity.Reroof Again, using the same reasoning as Section 1.10, we conclude that **Opportunity.Reroof** is a boolean.

Section 1.13: Opportunity.HOA **Opportunity.HOA** is an interesting case. All the **glimpse()** entries are *0* or *1*, but it has 6 unique entries. This suggests that it is intended to be a boolean, but contains some erroneous entries.

Section 1.14: PEStamp.Required. Using the same reasoning as Section 1.10, we conclude that **PEStamp.Required** is a boolean.

Section 2: Data Cleaning

Now that we have a cursory understanding of the data, we begin the cleaning process. Again, we go variable-by-varable. With the exception of **Created.Date**, we follow the same order as **Section 1**. **Created.Date** is inspected and cleaned first because we are only interested in observations after September 01, 2015 and we don't want to expend effort cleaning data we are not interested in.

Most of the data manipulation is done with **dplyr**. Visualizations are done with **ggplot2**.

Section 2.1: Created.Date We begin by using the package **lubridate** to convert all values in the **Created.Date** column to a *yyyy-mm-dd* format. **lubridate** will produce a warning if it can not parse one or more of the values as valid dates.

```
# convert all date data to date format
df <- mutate(df, Created.Date = mdy(Created.Date))  # no warning message, all data good
```

No warning message is produced, so we conclude that all non-missing values can be translated to a valid date format.

Now that we have a clean and reformatted data column, we filter all the rows of **df** by a **Created.Date** value of on or after *2015-09-01*. We also keep the rows of data with *NA* values in **Created.Date**.

```
df <- filter(df, (Created.Date >= "2015-09-01") | is.na(Created.Date))
```

Finally, in Figure 1, we create a histogram of the values in **Created.Date** to make sure they look reasonable. There appears to be a cyclic pattern to the data in Figure 1 with a period length of approximately 4 months. Some periodicity in the data is not surprising given that each value in **Created.Date** likely corresponds to a new project and often sales in a business are tied to financial quarters. However, it does seem unusual that the period length is 4 months rather than 3 months, as the length of each quarter is 3 months. Perhaps data covering a longer span would make the pattern and underlying drivers more evident.

```
Plot <- ggplot(df, aes(x = Created.Date)) + geom_histogram() +
    scale_x_date(date_breaks = "1 month") + theme(axis.text.x = element_text(angle = 45))
print(Plot)
```

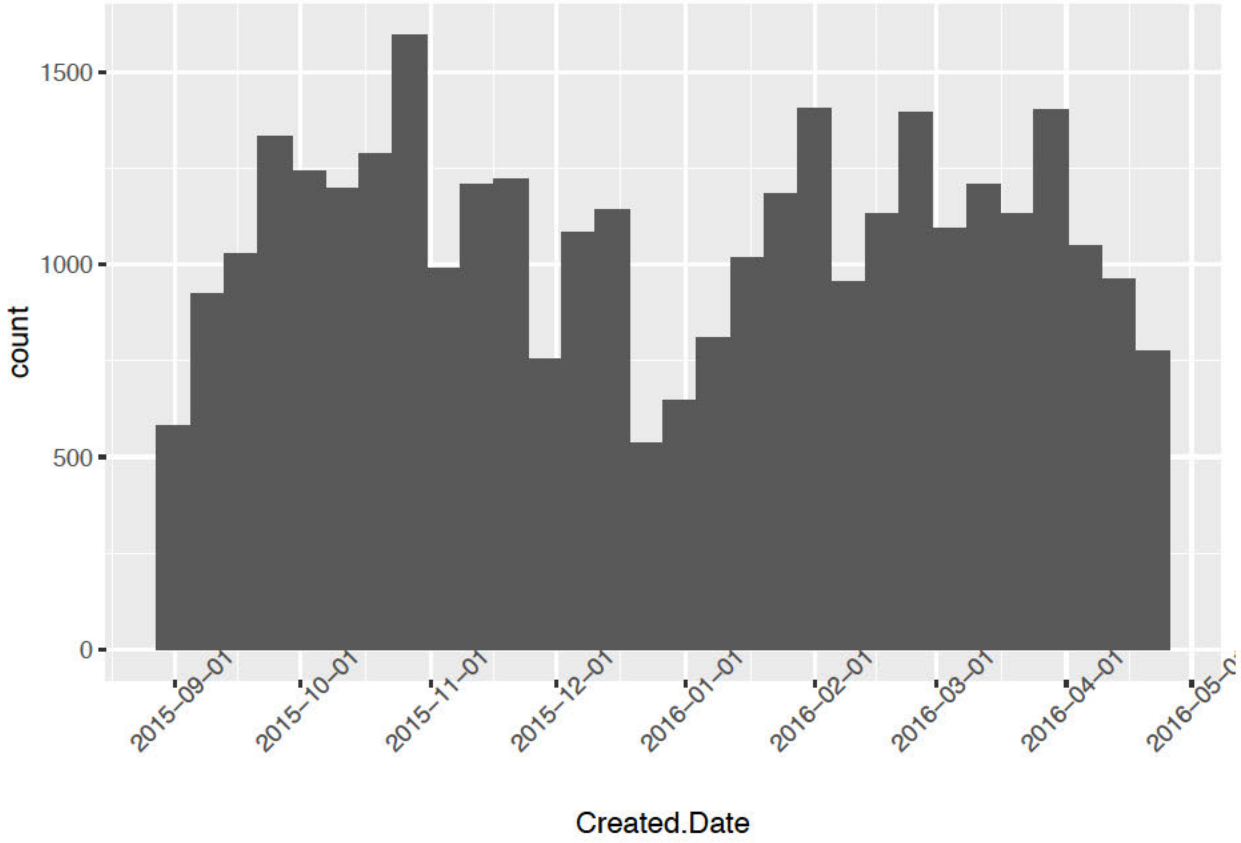

Figure 1: Histogram of Created.Date values

Section 2.2: Project.Name Since it looks like all the values in **Project.Name** should be following the format *PR-1234567890*, we begin by using regex to search for all entries that do not follow that format. In addition, we look at the number of *NA*s and zeros in each row to gain insight into whether there is merit in keeping these non-standard entries. The result is printed in Table 2, below.

```
project.name.filter <- df %>% filter(!grepl("PR-[0-9]{10}", Project.Name)) %>%
    mutate(numNA = rowSums(is.na(.))) %>% mutate(numZero = rowSums(. ==
    0, na.rm = TRUE)) %>% mutate(numZeroNA = numZero + numNA) %>%
    select(one_of(c("Project.Name", "numNA", "numZero", "numZeroNA")))
cap2 = "Non-standard project names with number of NAs and number of zeros in row"
kable(project.name.filter, caption = cap2)
```

Table 2: Non-standard project names with number of NAs and number of zeros in row

Project.Name	numNA	numZero	numZeroNA
test	8	4	12
Dmas	8	4	12
202R-043KONA/ Konadu	8	4	12
2143773610	8	4	12
Rubina Needles	8	4	12
austin	8	4	12
Laura ford	8	4	12
gonzalez maria 115r441gonz 0915	8	4	12
PK1VALVR9K3C:002-H	8	4	12
gonzalez maria 115r441gonz 0915	8	4	12
gonzalez maria 115r441gonz 0915	8	4	12
gonzalez maria 115r441gonz 0915	8	4	12
5403 Mount Vernon Ave-91710-3540	4	4	8
231R-024HARW	8	4	12
pr1732531655	8	4	12
Sweeney,Merrilee-	4	4	8
NA	14	0	14
Project_attributes	13	0	13
Copyright (c) 2000-2016 salesforce.com, inc. All rights reserved.	13	0	13
Confidential Information - Do Not Distribute	13	0	13
Generated By: Nathaniel Cook 4/22/2016 4:14 PM	13	0	13
Sunrun	13	0	13

Looking at the project names in Table 2, one possibility is to keep only the rows that have project names that can easily be put into the correct format, namely *2143773610* and *pr1732531655*. However, in these rows, all but two columns have data that is NA or zero. The lack of data is suspect, so we will keep these rows out of our data set.

Two rows, *Sweeney,Merrilee-* and *5403 Mount Vernon Ave-91710-3540* contain data that is potentially useful. In a real world situation, we would probably want to track down the correct name for these projects. However, for our case, we will just generate two new unique project identifiers by sorting the existing project names and adding 1 and 2, respectively, to the last project name in the list.

```
# Generate new unique project name according to standard
# format
all.project.names <- df %>% select(Project.Name) %>% filter(grepl("PR-[0-9]{10}",
    Project.Name)) %>% arrange(desc(Project.Name))
lastProjectNum = as.numeric(gsub("PR-([0-9]{10})", "\\1", all.project.names[1,
    1]))
newProjectName1 = paste0("PR-", lastProjectNum + 1)
newProjectName2 = paste0("PR-", lastProjectNum + 2)
print(newProjectName1)
```

```
## [1] "PR-2147451137"
```

```
print(newProjectName2)
```

```
## [1] "PR-2147451138"
```

```
# Rename projects
df <- df %>% mutate(Project.Name = replace(Project.Name, Project.Name ==
    "Sweeney,Merrilee-", newProjectName1)) %>% mutate(Project.Name = replace(Project.Name,
    Project.Name == "5403 Mount Vernon Ave-91710-3540", newProjectName2))
```

Now that we've renamed the projects we want to keep, we remove all project names that are in the non-standard format.

```
df <- df %>% filter(grepl("PR-[0-9]{10}", Project.Name) | is.na(Project.Name))
```

Given the assumption that each project name should be a unique identifier, we will proceed by searching for duplicates. Duplicate project names along with **Created.Date** value and number of missing row entries are printed below in Table 3.

```
project.name.dup <- df %>% group_by(Project.Name) %>% filter(n() >
    1) %>% ungroup() %>% mutate(numNA = rowSums(is.na(.))) %>%
    select(Project.Name, Created.Date, numNA) %>% arrange(Project.Name,
    Created.Date)
cap3 = "Rows of duplicate project names with created date and number of NA entries"
kable(project.name.dup, caption = cap3)
```

Table 3: Rows of duplicate project names with created date and number of NA entries

Project.Name	Created.Date	numNA
PR-2132921988	2015-09-20	0
PR-2132921988	2015-10-26	8

From Table 3 we remark that the duplicate project name has one row with many more *NA* entries. We choose to remove the row with more *NA* entries.

```
# remove rows that have duplicates
df <- df %>% group_by(Project.Name) %>% filter(n() == 1) %>%
    ungroup()

# take the duplicates and order them by number of NAs, remove
# the second half of that data
project.name.dup.clean <- project.name.dup %>% mutate(numNA = rowSums(is.na(.))) %>%
    arrange(numNA) %>% slice(1:(nrow(project.name.dup)/2)) %>%
    select(-numNA)  # remove numNA column

# re-attach the cleaned duplicate rows
df <- bind_rows(df, project.name.dup.clean)
```

Section 2.3: Project.Status According to Table 1, there are only five unique entries in **Project.Status**. We can make a histogram of the distribution of entries in those categories (see Figure 2). We define a function to do this task so that it can be used for other columns as well.

```
# define general function to avoid duplicating code
barplot_fun <- function(dataSet, colName) {
    Plot <- dataSet %>% select(category = contains(colName)) %>%
        transmute(category = replace(category, is.na(category),
            "NA")) %>% ggplot(aes(x = category)) + geom_bar() +
        geom_text(stat = "count", aes(label = ..count..), vjust = -0.1) +
        ggtitle(colName) + theme(axis.text.x = element_text(angle = 90))
    return(Plot)
}
print(barplot_fun(df, "Project.Status"))
```

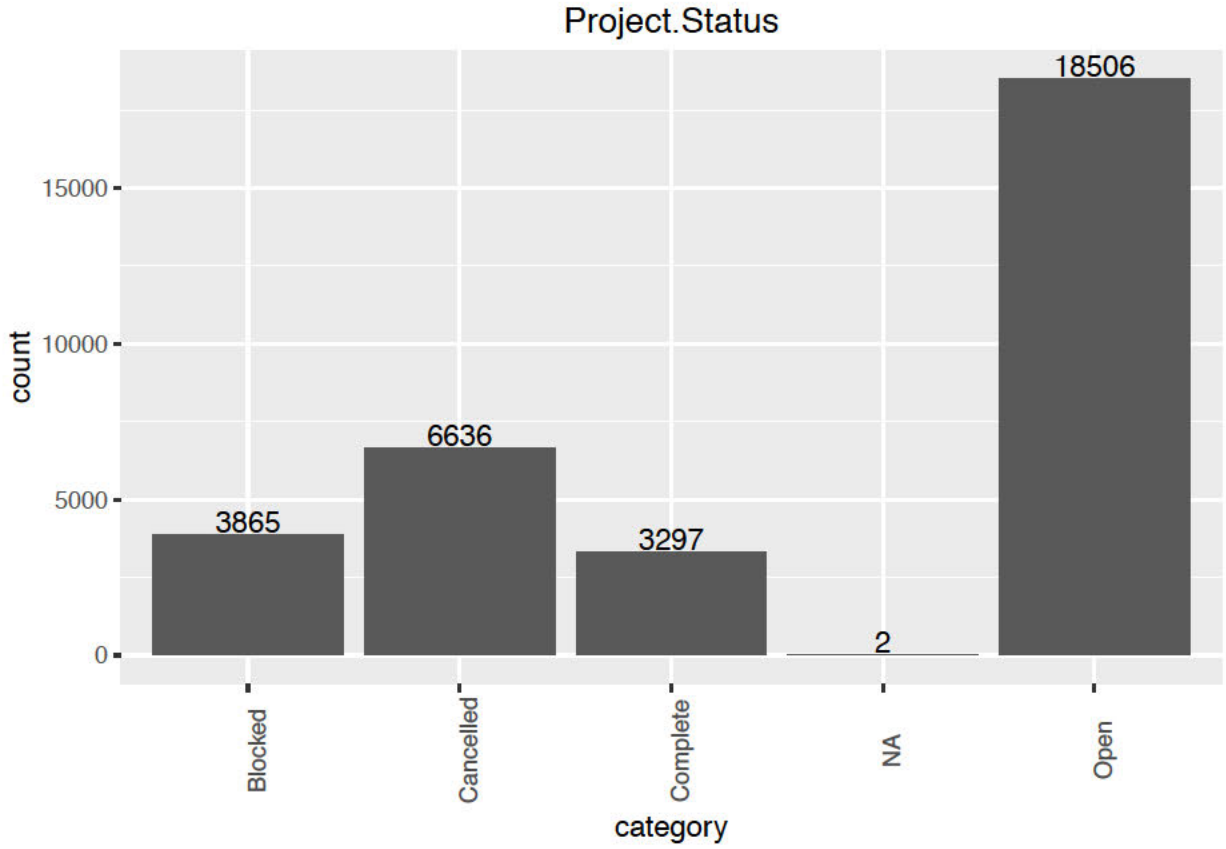

Figure 2: Histogram of entries in categories of Project.Status

The categories and distribution in Figure 2 look reasonable. The only remaining step is to change **Project.Status** to a factor.

```
# this data looks good just want to make it a factor
df <- df %>% mutate(Project.Status = factor(Project.Status))
```

Section 2.4: Opportunity.Thru According to Table 1, there are only seven unique entries in **Opportunity.Thru**, so we generate a histogram, shown in Figure 3.

```
print(barplot_fun(df, "Opportunity.Thru"))
```

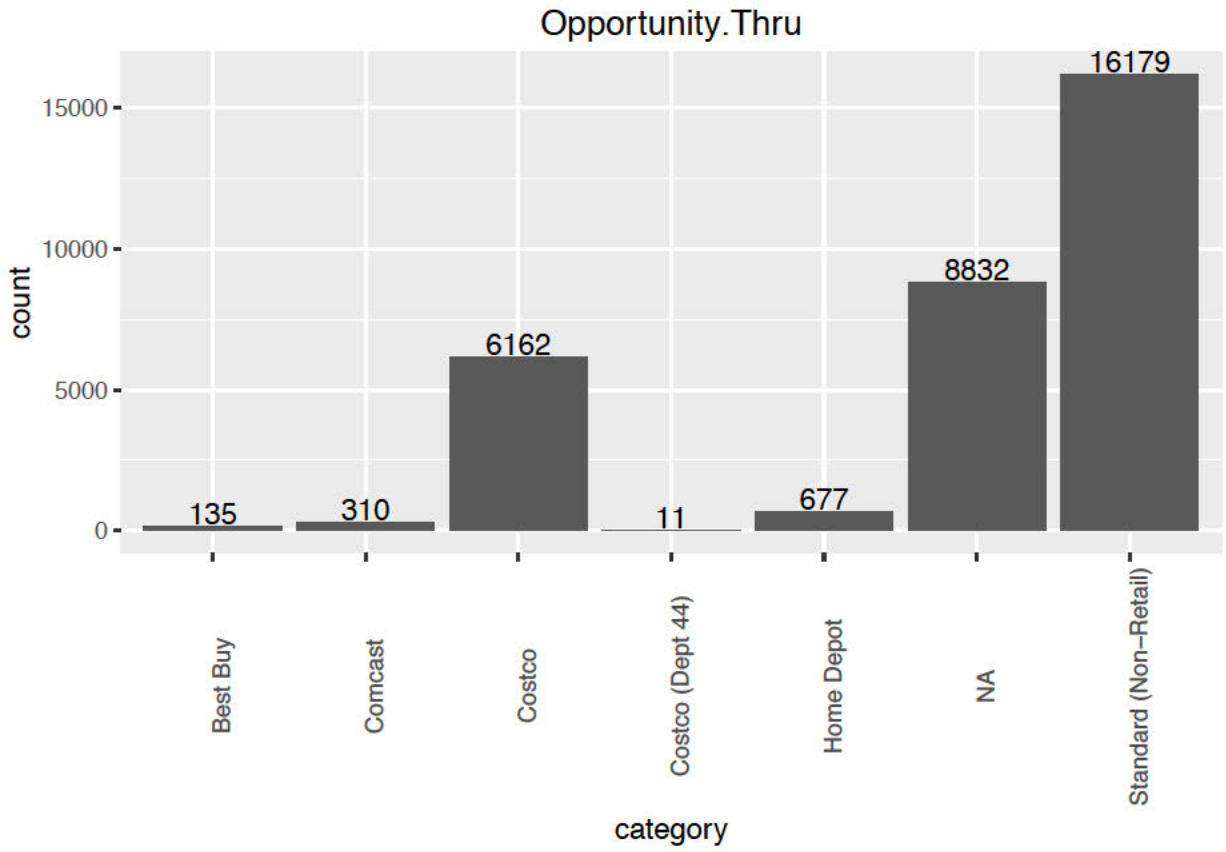

Figure 3: Histogram of entries in categories of Opportunity.Thru

Based on how few entries are in *Costco Dept 44* versus *Costco*, and the fact that it sounds like the former is a subset of the latter, we make the decision to merge these categories together. Then we change **Opportunity.Thru** to a factor.

```
df <- df %>% mutate(Opportunity.Thru = replace(Opportunity.Thru,
    Opportunity.Thru == "Costco (Dept 44)", "Costco")) %>% mutate(Opportunity.Thru = factor(Opportunity
```

Section 2.5: Agreement.Type Once again, there are few enough unique entires in **Agreement.Type** to generate a histogram (Figure 4).

```
print(barplot_fun(df, "Agreement.Type"))
```

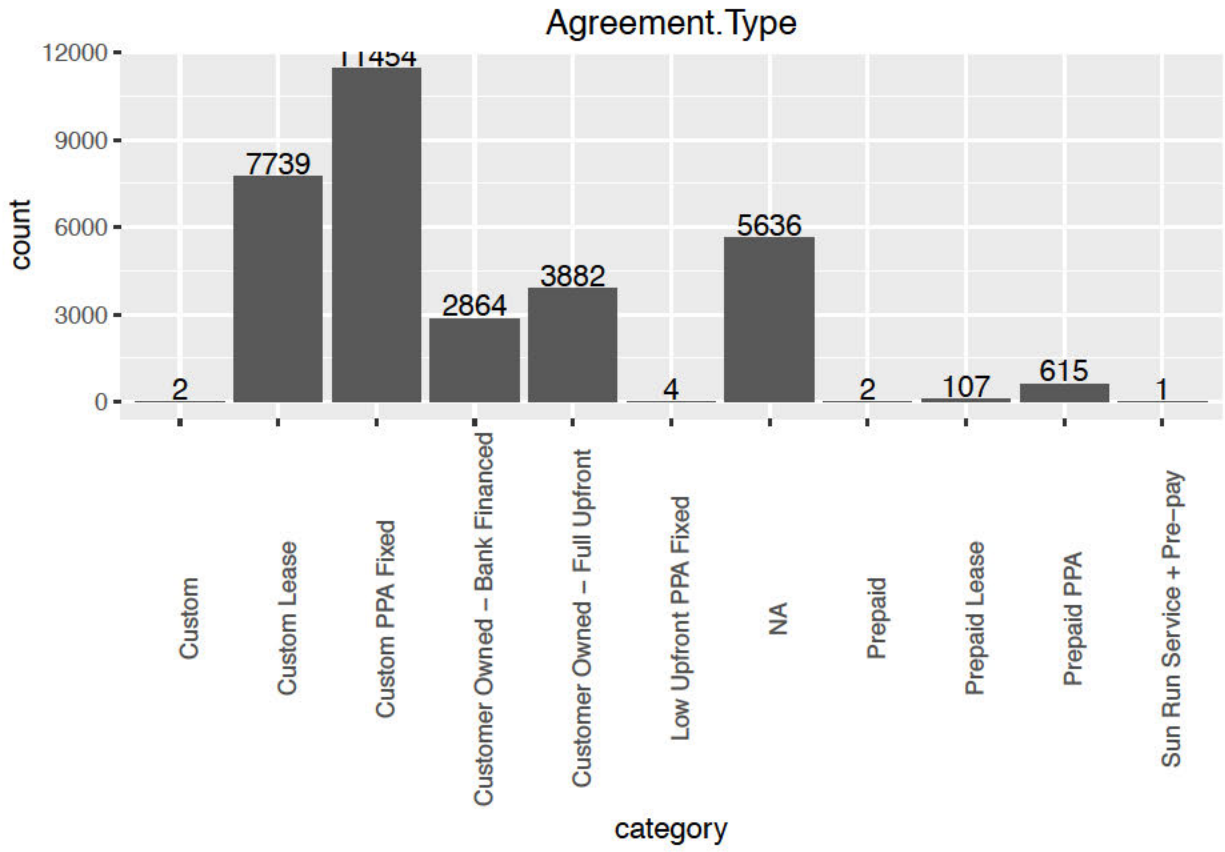

Figure 4: Histogram of entries in categories of Agreement.Type

It seems as though **Agreement.Type** has two categories of data that are being combined:

1. the general type of owernership agreement e.g. PPA, Lease, Customer Owned, Sun Run Service
2. refinement of the agreement within the general category e.g. Custom, Bank Financed, Prepaid, etc.

It may be useful to create another column called **Agreement.Category** with just the first category in order to enable data analysis that treats only the general agreement type as a factor. We create the new column and change both **Agreement.Type** and **Agreement.Category** to factors.

```
df <- df %>% mutate(Agreement.Category = ifelse(grepl("PPA",
    Agreement.Type), "PPA", ifelse(grepl("Lease", Agreement.Type),
    "Lease", ifelse(grepl("Customer Owned", Agreement.Type),
        "Customer Owned", ifelse(grepl("Sun Run", Agreement.Type),
            "Sun Run", NA)))))
# Change both variables to factor
df <- df %>% mutate(Agreement.Type = factor(Agreement.Type)) %>%
    mutate(Agreement.Category = factor(Agreement.Category))
```

In Figure 5 we show a histogram of the new column **Agreement.Category** for reference.

```
print(barplot_fun(df, "Agreement.Category"))
```

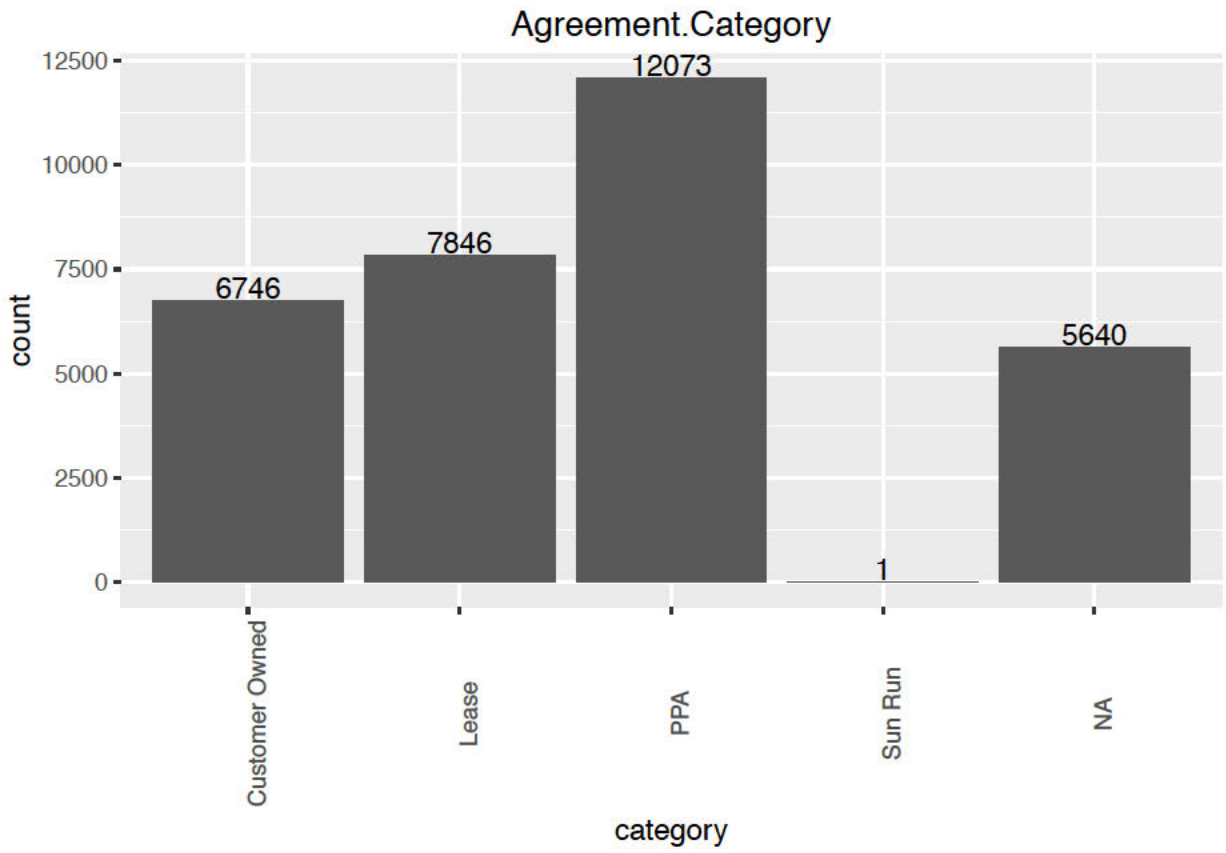

Figure 5: Histogram of entries in categories of Agreement.Category

Section 2.6: Install.Branch As with the other columns that are categorical in nature, we make a histogram of **Install.Branch** in Figure 6.

```
print(barplot_fun(df, "Install.Branch"))
```

As seen in Figure 6, the categories in **Install.Branch** are an unfortunate and inconsistent combination of state names, city names, and region names. To make the meaning of this column more consistent, we should seperate it into two seperate columns – one with the state named by standard abbreviation, and another with the area inside that state (when available).

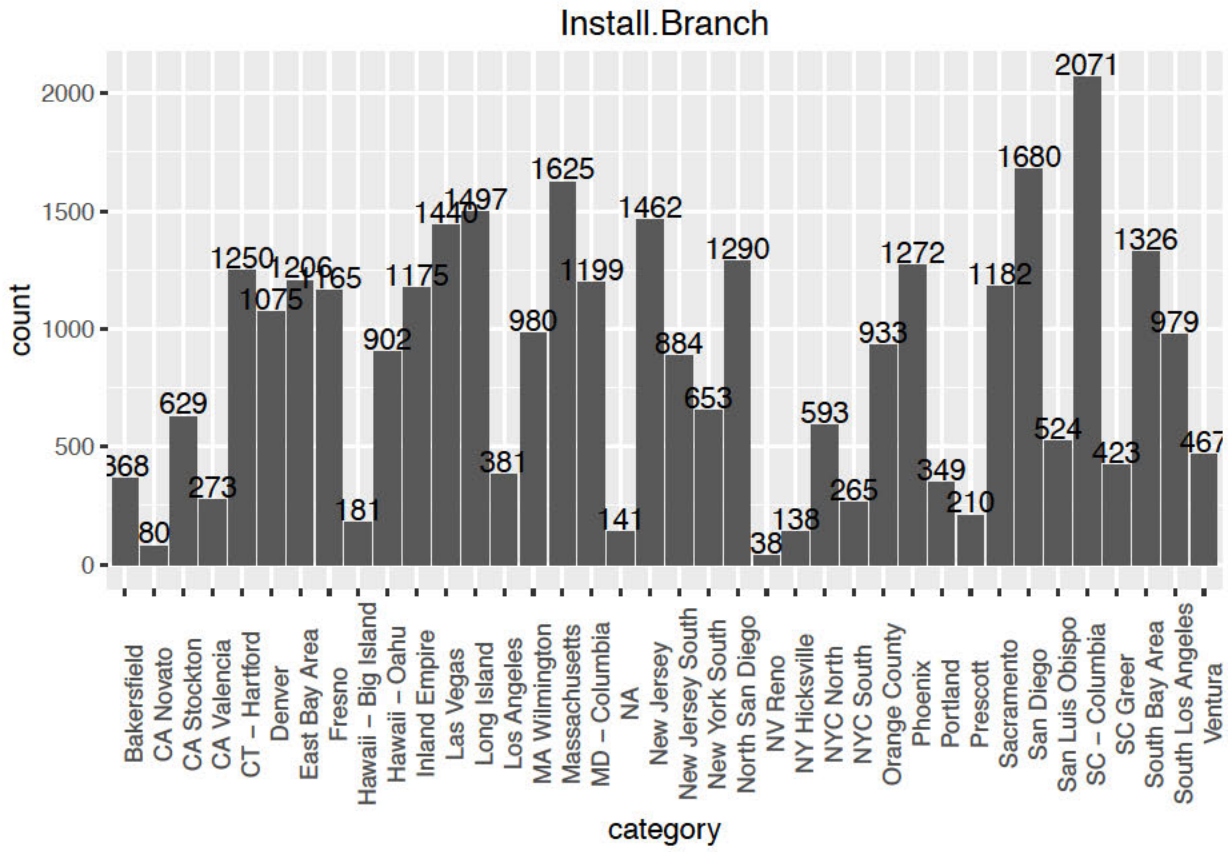

Figure 6: Histogram of entries in categories of Install.Branch

To accomplish this, we will do as much as possible programmatically. However, given the inconsistencies in the data reporting, some cleaning will have to be done by hand. In this case, there are actually few enough unique categories that they all could be fixed by hand, but that may not be the case in a larger dataset since **Install.Branch** can theoretically have a huge number of possible values. For this reason, it is still worthwhile to implement some automated cleaning functionality.

We begin by attempting to separate states from cities in the cases where a state is named. We load the data sets **state.name** and **state.abb** to search for state names and abbreviations against. To keep track of the category renaming, we create a data frame called **branch.mapping**.

```
branch.mapping <- data.frame(original = unique(df$Install.Branch))

## find and strip state abbreviations create regex search
## pattern
state.abb.gsearch <- paste(state.abb, collapse = " |")
state.abb.gsearch <- paste(c("(", state.abb.gsearch, "|)(.*)"),
    collapse = "")
branch.mapping <- branch.mapping %>% mutate(original.filter1 = gsub(state.abb.gsearch,
    "\\2", original)) %>% mutate(state.abbrev = gsub(state.abb.gsearch,
    "\\1", original)) %>% mutate(state.abbrev = gsub("(.+) ",
    "\\1", state.abbrev))

# replace NYC with New York City and strip '-' and ' ' space
# from beginning of rows
branch.mapping <- branch.mapping %>% mutate(original.filter2 = gsub("NYC",
    "New York City", original.filter1)) %>% mutate(original.filter2 = gsub(" *-* *(.+)",
    "\\1", original.filter2))

## Find and strip state names Create regex search pattern Note
## this causes a problem for New York City We make a specific
## exception for New York City, since we know there are lots
## of things are there
state.name.gsearch <- paste(state.name[state.name != "New York"],
    collapse = "|")
state.name.gsearch <- paste(c("(", state.name.gsearch, "|New York(?! City)|)(.*)"),
    collapse = "")
branch.mapping <- branch.mapping %>% mutate(original.filter3 = gsub(state.name.gsearch,
    "\\2", original.filter2, perl = T)) %>% mutate(state.names = gsub(state.name.gsearch,
    "\\1", original.filter2, perl = T))

# Clean up mapping data and strip '-' and ' ' from beginning
# of rows
branch.mapping <- branch.mapping %>% mutate(cities = gsub(" *-* *(.+)",
    "\\1", original.filter3)) %>% mutate(cities = ifelse(cities ==
    "", NA, cities)) %>% select(c(original, cities, state.abbrev,
    state.names))

# Match state names to abbreviations to merge names and
# abbreviations columns
branch.mapping <- branch.mapping %>% left_join(data.frame(state.abbrev.match = state.abb,
    state.names = state.name, stringsAsFactors = FALSE), by = "state.names") %>%
    mutate(state.abbrev = ifelse(is.na(state.abbrev) | state.abbrev ==
        "", state.abbrev.match, state.abbrev)) %>% select(-state.abbrev.match)
```

Table 4 shows the result of our effort so far.

```
cap4 = "Install.Branch category name mapping data frame, first pass"
branch.mapping.tab <- mutate(branch.mapping, state.abbrev = replace(state.abbrev,
    is.na(state.abbrev), ""))
kable(branch.mapping.tab, caption = cap4, row.names = TRUE)
```

Table 4: Install.Branch category name mapping data frame, first pass

	original	cities	state.abbrev	state.names
1	Sacramento	Sacramento		
2	MD - Columbia	Columbia	MD	
3	Inland Empire	Inland Empire		
4	NYC North	New York City North		
5	San Diego	San Diego		
6	Massachusetts	NA	MA	Massachusetts
7	East Bay Area	East Bay Area		
8	SC - Columbia	Columbia	SC	
9	SC Greer	Greer	SC	
10	Las Vegas	Las Vegas		
11	North San Diego	North San Diego		
12	South Bay Area	South Bay Area		
13	CT - Hartford	Hartford	CT	
14	Ventura	Ventura		
15	Bakersfield	Bakersfield		
16	Hawaii - Big Island	Big Island	HI	Hawaii
17	Orange County	Orange County		
18	Los Angeles	Los Angeles		
19	Long Island	Long Island		
20	New Jersey South	South	NJ	New Jersey
21	Portland	Portland		
22	New Jersey	NA	NJ	New Jersey
23	Fresno	Fresno		
24	Hawaii - Oahu	Oahu	HI	Hawaii
25	South Los Angeles	South Los Angeles		
26	New York South	South	NY	New York
27	Denver	Denver		
28	NYC South	New York City South		
29	MA Wilmington	Wilmington	MA	
30	CA Stockton	Stockton	CA	
31	Phoenix	Phoenix		
32	NA	NA		NA
33	San Luis Obispo	San Luis Obispo		
34	Prescott	Prescott		
35	CA Valencia	Valencia	CA	
36	NV Reno	Reno	NV	
37	NY Hicksville	Hicksville	NY	
38	CA Novato	Novato	CA	

Next we use the dataset **us.cities** from the package **maps** to attempt to match city names to states in the cases where **Install.Branch** does not specify a state. We use the rule that we select the most populous city when there are multiple cities by the same name.

```
# Create data frame of cities and states with population
city.state = data.frame(cities = gsub("(.*) [A-Z]{2}", "\\1",
    us.cities$name), states = us.cities$country.etc, popul = us.cities$pop,
    stringsAsFactors = FALSE)
# Sort by population and remove the less populous ones
city.state <- city.state %>% arrange(desc(popul)) %>% distinct(cities,
    .keep_all = TRUE)
# Do join
branch.mapping <- branch.mapping %>% left_join(city.state, by = "cities") %>%
    mutate(state.abbrev = ifelse(is.na(state.abbrev) | state.abbrev ==
        "", states, state.abbrev)) %>% select(-states, -state.names,
    -popul)
```

Table 5 shows the result of the programmatic city to state matching.

```
cap5 = "Install.Branch category name mapping data frame, second pass"
branch.mapping.tab <- mutate(branch.mapping, state.abbrev = replace(state.abbrev,
    is.na(state.abbrev), ""))
kable(branch.mapping.tab, caption = cap5, row.names = TRUE)
```

Table 5: Install.Branch category name mapping data frame, second pass

	original	cities	state.abbrev
1	Sacramento	Sacramento	CA
2	MD - Columbia	Columbia	MD
3	Inland Empire	Inland Empire	
4	NYC North	New York City North	
5	San Diego	San Diego	CA
6	Massachusetts	NA	MA
7	East Bay Area	East Bay Area	
8	SC - Columbia	Columbia	SC
9	SC Greer	Greer	SC
10	Las Vegas	Las Vegas	NV
11	North San Diego	North San Diego	
12	South Bay Area	South Bay Area	
13	CT - Hartford	Hartford	CT
14	Ventura	Ventura	
15	Bakersfield	Bakersfield	CA
16	Hawaii - Big Island	Big Island	HI
17	Orange County	Orange County	
18	Los Angeles	Los Angeles	CA
19	Long Island	Long Island	
20	New Jersey South	South	NJ
21	Portland	Portland	OR
22	New Jersey	NA	NJ
23	Fresno	Fresno	CA
24	Hawaii - Oahu	Oahu	HI
25	South Los Angeles	South Los Angeles	
26	New York South	South	NY
27	Denver	Denver	CO
28	NYC South	New York City South	

	original	cities	state.abbrev
29	MA Wilmington	Wilmington	MA
30	CA Stockton	Stockton	CA
31	Phoenix	Phoenix	AZ
32	NA	NA	
33	San Luis Obispo	San Luis Obispo	CA
34	Prescott	Prescott	AZ
35	CA Valencia	Valencia	CA
36	NV Reno	Reno	NV
37	NY Hicksville	Hicksville	NY
38	CA Novato	Novato	CA

At this point, we fill in the remaining missing state names by hand. Hand cleaning is required for 10 of the original 38 categories.

```
# just fill in the rest by hand
branch.mapping$state.abbrev[3] <- "CA"
branch.mapping$state.abbrev[4] <- "NY"
branch.mapping$state.abbrev[7] <- "CA"
branch.mapping$state.abbrev[11] <- "CA"
branch.mapping$state.abbrev[12] <- "CA"
branch.mapping$state.abbrev[14] <- "CA"
branch.mapping$state.abbrev[17] <- "CA"
branch.mapping$state.abbrev[19] <- "CA"
branch.mapping$state.abbrev[25] <- "CA"
branch.mapping$state.abbrev[28] <- "NY"
```

Finally, we use **branch.mapping** to split **Install.Branch** into two columns, **Install.Branch.Area** and **Install.Branch.State** with the new, more consistent category names.

```
# rename columns to prepare for join with data table
branch.mapping <- transmute(branch.mapping, Install.Branch = original,
    Install.Branch.State = state.abbrev, Install.Branch.Area = cities)

# now do the mapping
df <- df %>% left_join(branch.mapping, by = "Install.Branch") %>%
    select(-Install.Branch)

# and change state and area to factors
df <- df %>% mutate(Install.Branch.Area = factor(Install.Branch.Area)) %>%
    mutate(Install.Branch.State = factor(Install.Branch.State))
```

In Figure 7 we show a histogram of the new column **Install.Branch.State** for reference.

```
print(barplot_fun(df, "Install.Branch.State"))
```

Section 2.7: Utility.Company Again, we make a histogram of the categories in **Utility.Company**. Result is shown in Figure 8.

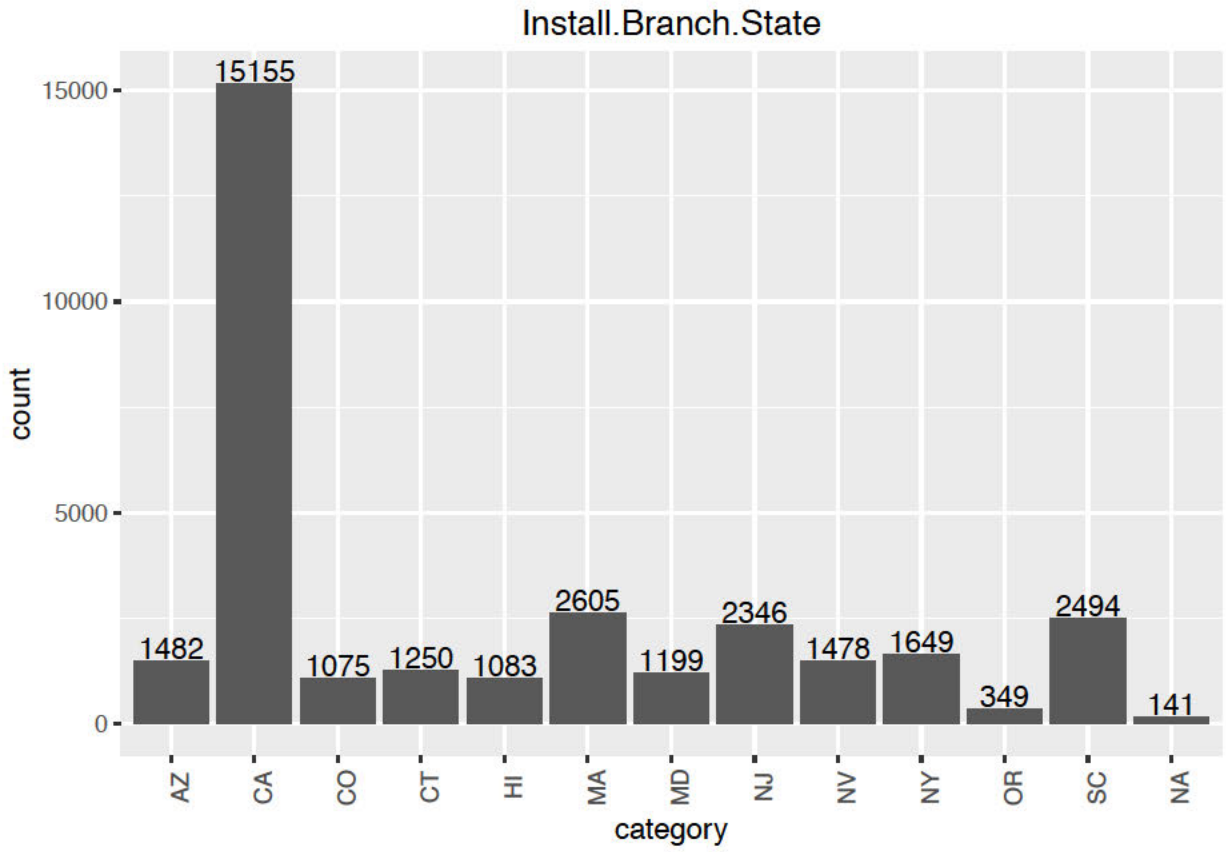

Figure 7: Histogram of entries in categories of Install.Branch.State

```
print(barplot_fun(df, "Utility.Company"))
```

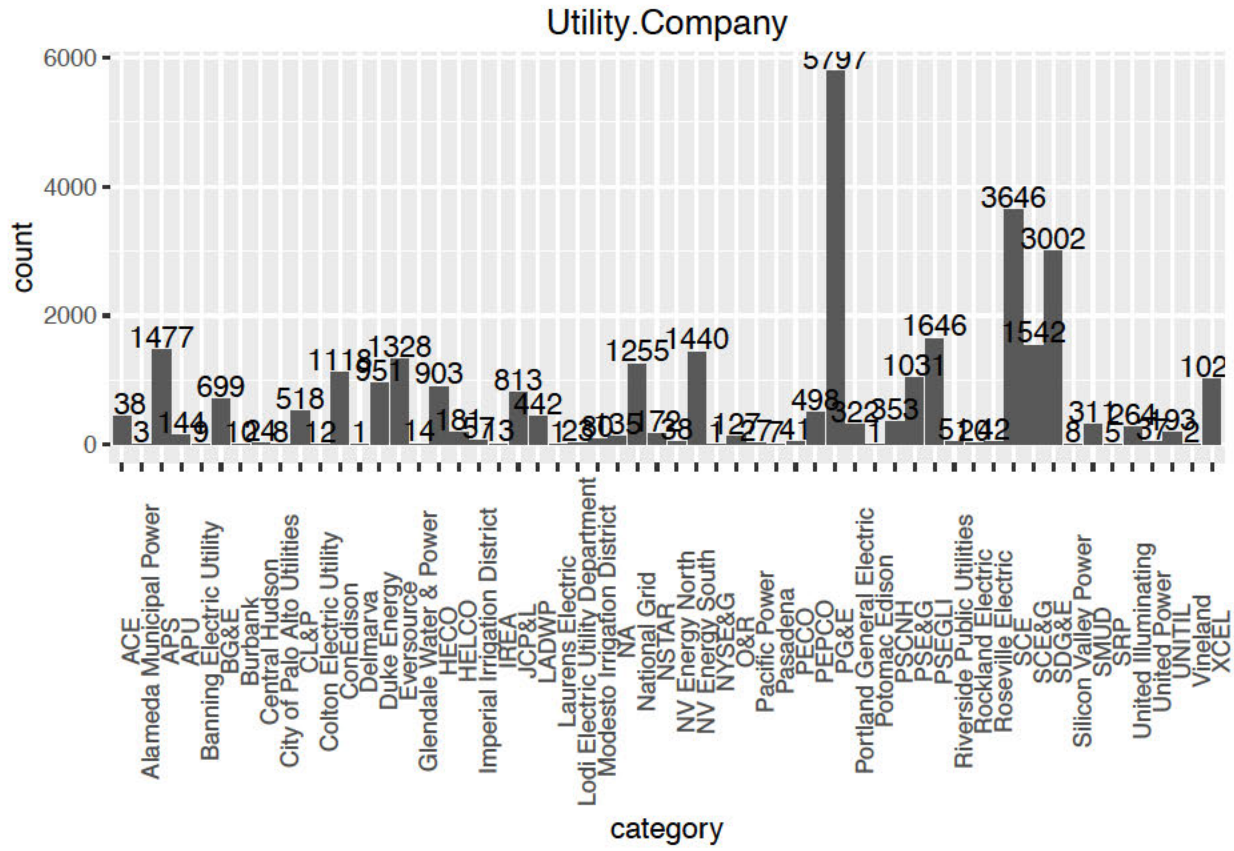

Figure 8: Histogram of entries in categories of Utility.Company

All of the categories and frequencies in Figure 8 look reasonable, so no cleaning is needed other than changing **Utility.Company** to a factor.

```
df <- df %>% mutate(Utility.Company = factor(Utility.Company))
```

Section 2.8: Jurisdiction.Name Per Table 1, there are too many unique entries in **Jurisdiction.Name** for a histogram to be useful. We inspect the data directly instead.

It looks like most of the data follows the standard format *AB-CITYNAME*. We use regex to look for any entries that do not follow the standard format and print the result.

```
# First use regex to check whether the data all follows
# AB-CITYNAME
jurisdiction.filter <- df %>% select(Jurisdiction.Name) %>% na.omit %>%
    filter(!grepl("^[A-Z][A-Z][A-Z]?-.+", Jurisdiction.Name))
print(jurisdiction.filter)
```

```
## # A tibble: 1 x 1
##   Jurisdiction.Name
##               <chr>
## 1            Summit
```

One unusual row found. We make the decision to replace this value with NA.

```
df <- df %>% mutate(Jurisdiction.Name = replace(Jurisdiction.Name,
    Jurisdiction.Name == "Summit", NA))
```

Next we check that all the state abbreviations in **Jurisdiction.Name** are valid. Non-compliant abbreviations are printed.

```
# Create regex pattern to match
state.abb.gsearch2 <- paste(state.abb, collapse = "|")
state.abb.gsearch2 <- paste(c("^(", state.abb.gsearch2, ")$"),
    collapse = "")

jurisdiction.states <- df %>% select(Jurisdiction.Name) %>% na.omit() %>%
    # pull out state abbreviations
mutate(j.states = gsub("^([A-Z][A-Z][A-Z]?)-.+", "\\1", Jurisdiction.Name)) %>%
    distinct() %>% # search for non-matches
mutate(j.is.state = !grepl(state.abb.gsearch2, j.states))  # look for not states
print(jurisdiction.states$j.states[jurisdiction.states$j.is.state])
```

```
## [1] "NYC"
```

The only invalid abbreviation found is *NYC*. That is a meaningful abbreviation, so we leave it in.

Section 2.9: System.Size Per Table 1, **System.Size** is a continuous numeric variable. The distribution, shown in Figure 9, indicates that there is a threshold minimum size for systems at approximately size of 2. It is suspicious that there appear to be a few values at or near zero. Figure 10 takes a closer looks at the distribution near 2.

```
Plot <- ggplot(df, aes(x = System.Size)) + geom_histogram(binwidth = 1) +
    scale_x_continuous(breaks = 0:30)
print(Plot)
```

```
Plot <- ggplot(df, aes(x = System.Size)) + geom_histogram(binwidth = 0.2) +
    scale_x_continuous(breaks = seq(0, 2.5, 0.2), limits = c(-1,
        2.5))
print(Plot)
```

In Figure 10 we can see there there are one or two data points at zero. We replace all zero values with *NA*.

```
df <- df %>% mutate(System.Size = replace(System.Size, System.Size ==
    0, NA))
```

Section 2.10: Service.Contract A histogram of **Service.Contract** is shown in Figure 11.

```
print(barplot_fun(df, "Service.Contract"))
```

All data is either *NA*, 0, or 1. We therefore change the column type to logical.

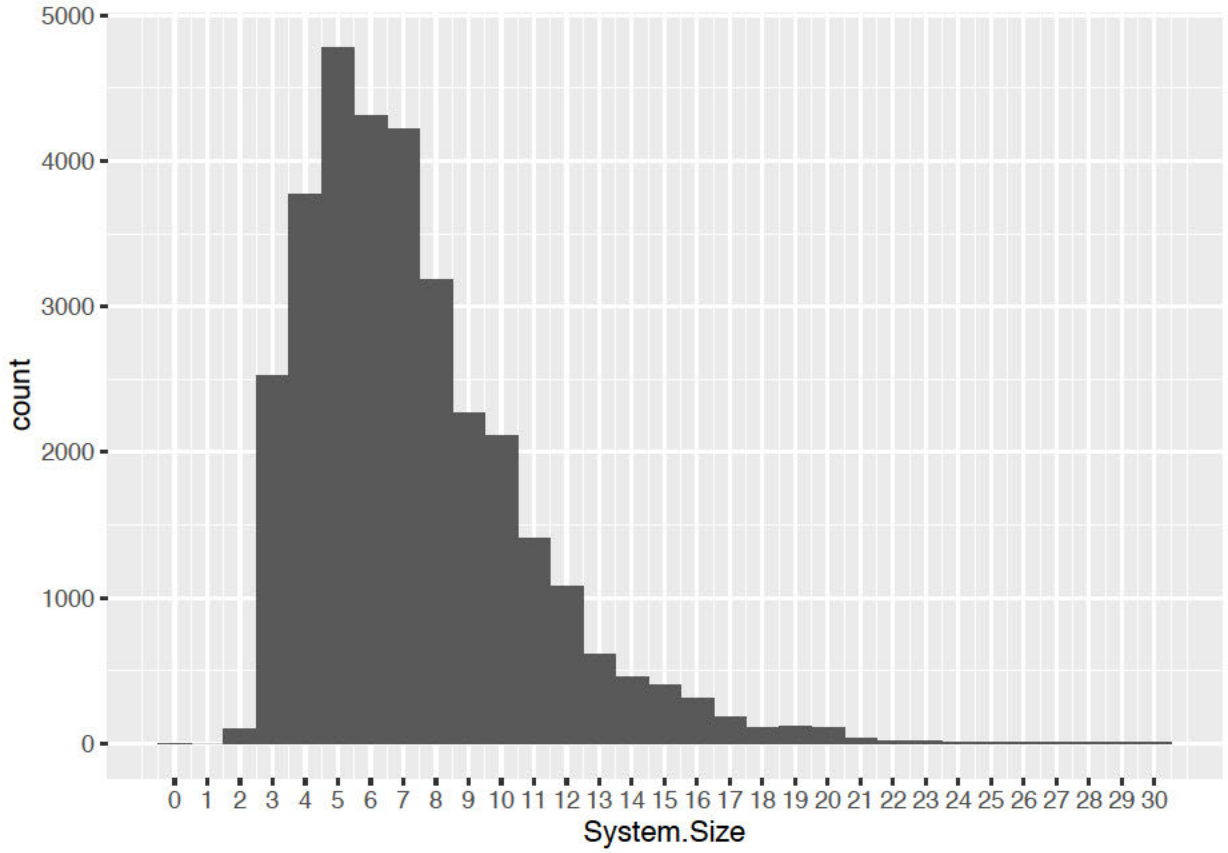

Figure 9: Histogram of System.Size

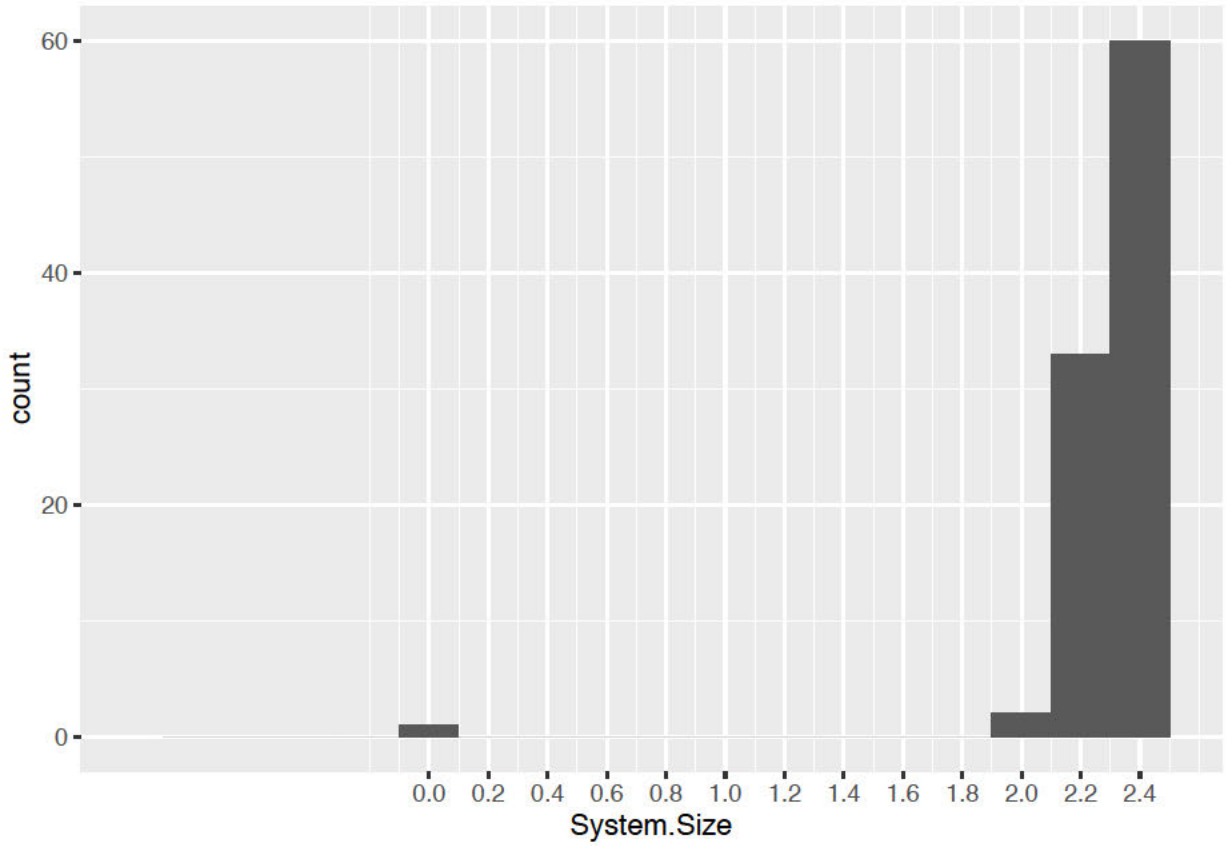

Figure 10: Zoomed-in histogram of System.Size

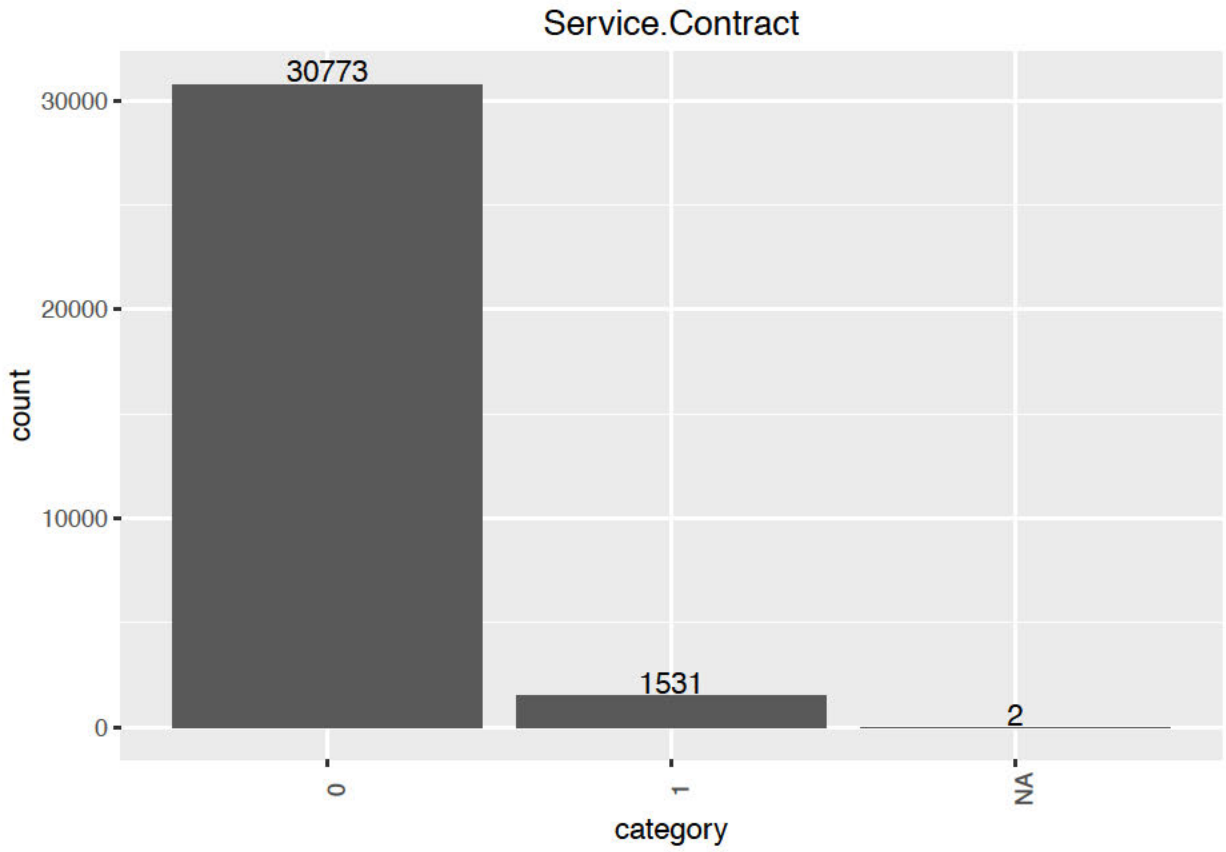

Figure 11: Histogram of entries in categories of Service.Contract

```
df <- df %>% mutate(Service.Contract = as.logical(Service.Contract))
```

Section 2.11: Opportunity.Upgrade A histogram of **Opportunity.Upgrade** is shown in Figure 12.

```
print(barplot_fun(df, "Opportunity.Upgrade"))
```

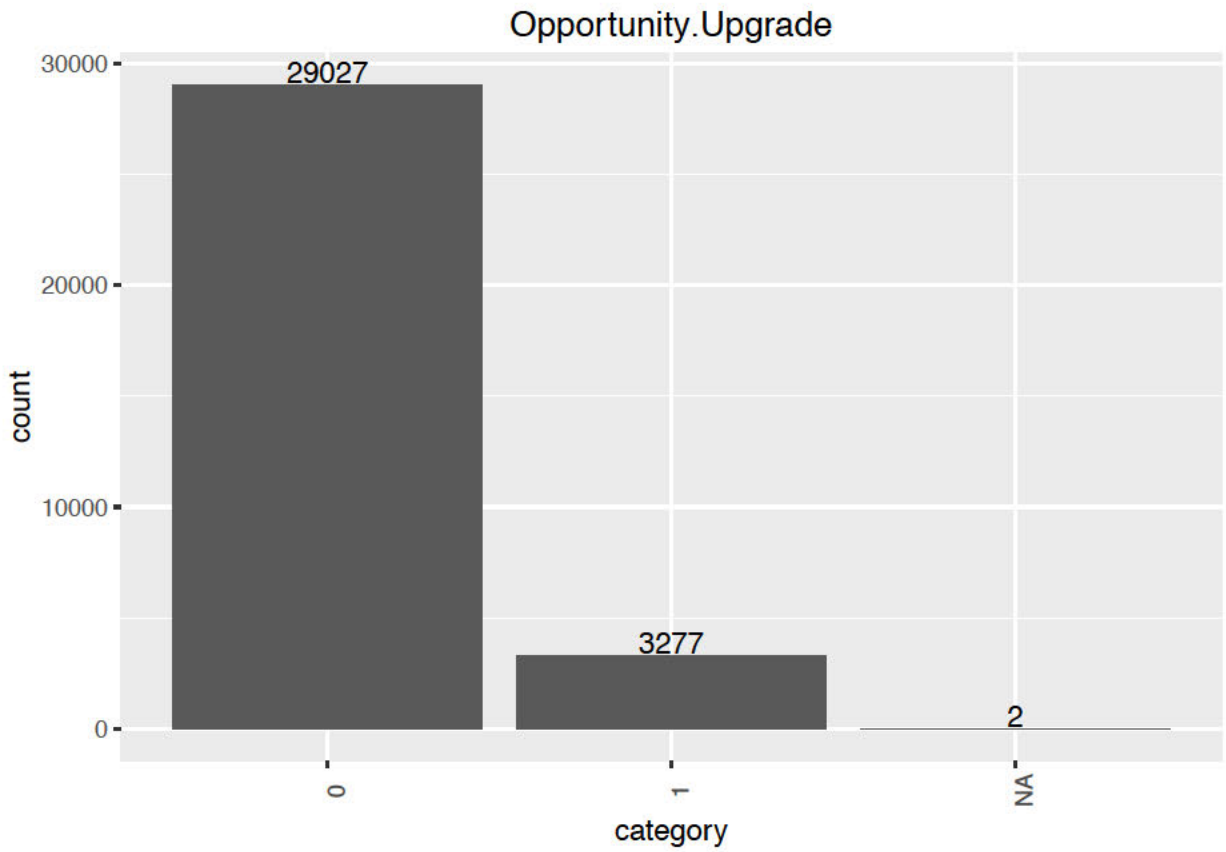

Figure 12: Histogram of entries in categories of Opportunity.Upgrade

All data is either *NA*, 0, or 1. We therefore change the column type to logical.

```
df <- df %>% mutate(Opportunity.Upgrade = as.logical(Opportunity.Upgrade))
```

Section 2.12: Opportunity.Reroof A histogram of **Opportunity.Reroof** is shown in Figure 13.

```
print(barplot_fun(df, "Opportunity.Reroof"))
```

All data is either *NA*, 0, or 1. We therefore change the column type to logical.

```
df <- df %>% mutate(Opportunity.Reroof = as.logical(Opportunity.Reroof))
```

Section 2.13: Opportunity.HOA A histogram of **Opportunity.HOA** is shown in Figure 14.

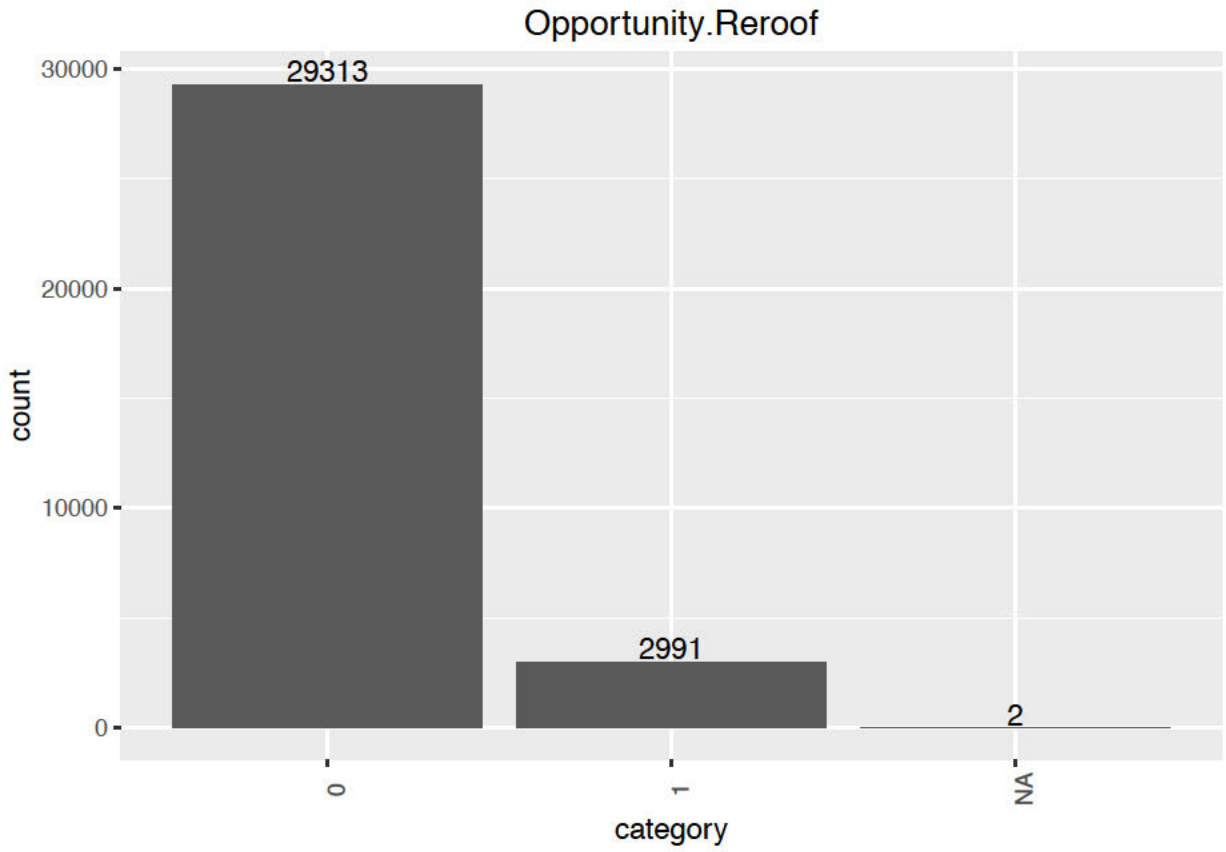

Figure 13: Histogram of entries in categories of Opportunity.Reroof

```
print(barplot_fun(df, "Opportunity.HOA"))
```

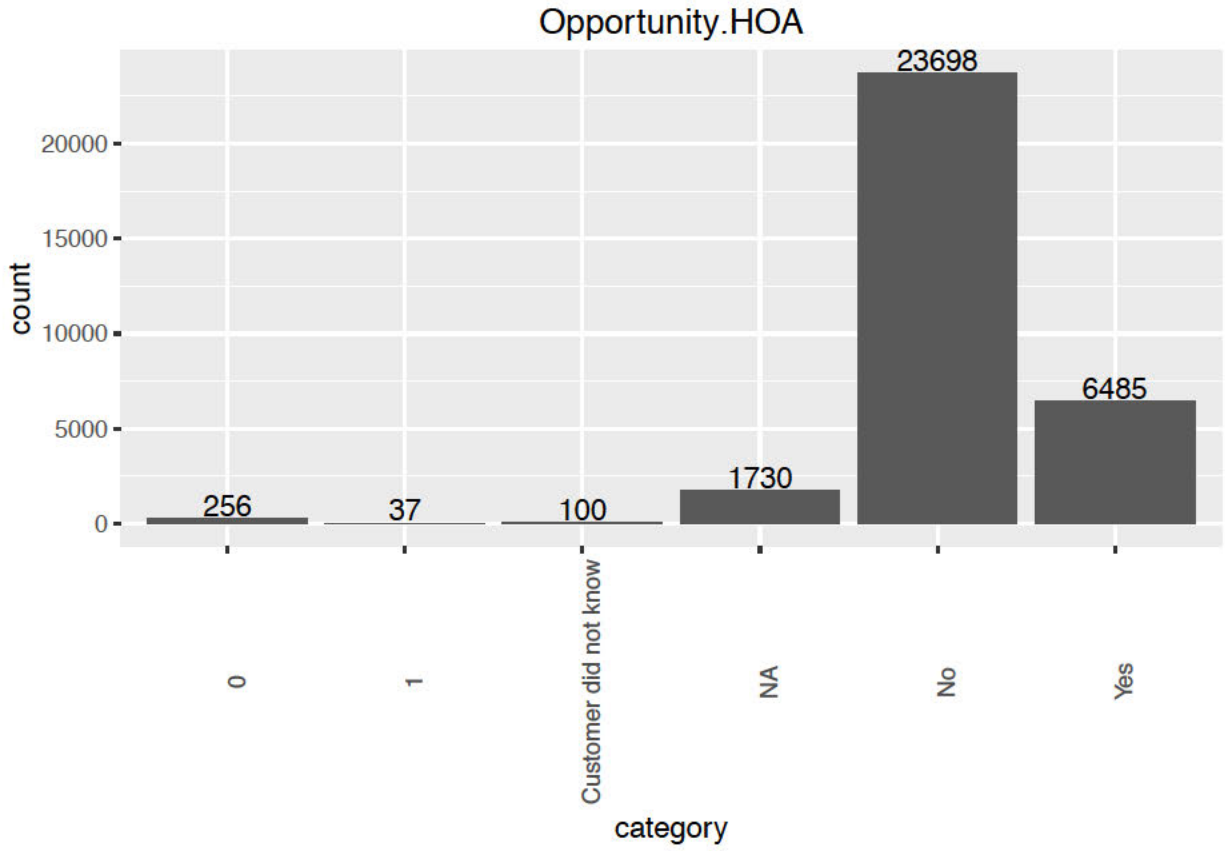

Figure 14: Histogram of entries in categories of Opportunity.HOA

In Figure 14 we see that there is a category called "Customer did not know". One option would be to merge "Customer did not know" with *NA*. However, the better option is likely to keep "Customer did not know as disintct" from *NA*, as it may be the case that *NA* means something akin to "Customer was not asked". We also remark that some data has been encoded with "yes" or "no" while others have been encoded with "0" or "1". Because we can not change this variable to boolean due to the "Customer did not know" category, we choose to recode "0" as "No" and "1" as "Yes". "No" and "Yes" also appear to be the preferred category labels based on the relative frequencies. Finally, we change **Opportunity.HOA** to a factor.

```
# Recode 0, 1 to 'No', 'Yes'.
df <- df %>% mutate(Opportunity.HOA = replace(Opportunity.HOA,
    Opportunity.HOA == "1", "Yes")) %>% mutate(Opportunity.HOA = replace(Opportunity.HOA,
    Opportunity.HOA == "0", "No")) %>% mutate(Opportunity.HOA = factor(Opportunity.HOA))
```

Section 2.14: PEStamp.Required A histogram of **PEStamp.Required** is shown in Figure 15.

```
print(barplot_fun(df, "PEStamp.Required"))
```

All data is either *NA*, 0, or 1. We therefore change the column type to logical.

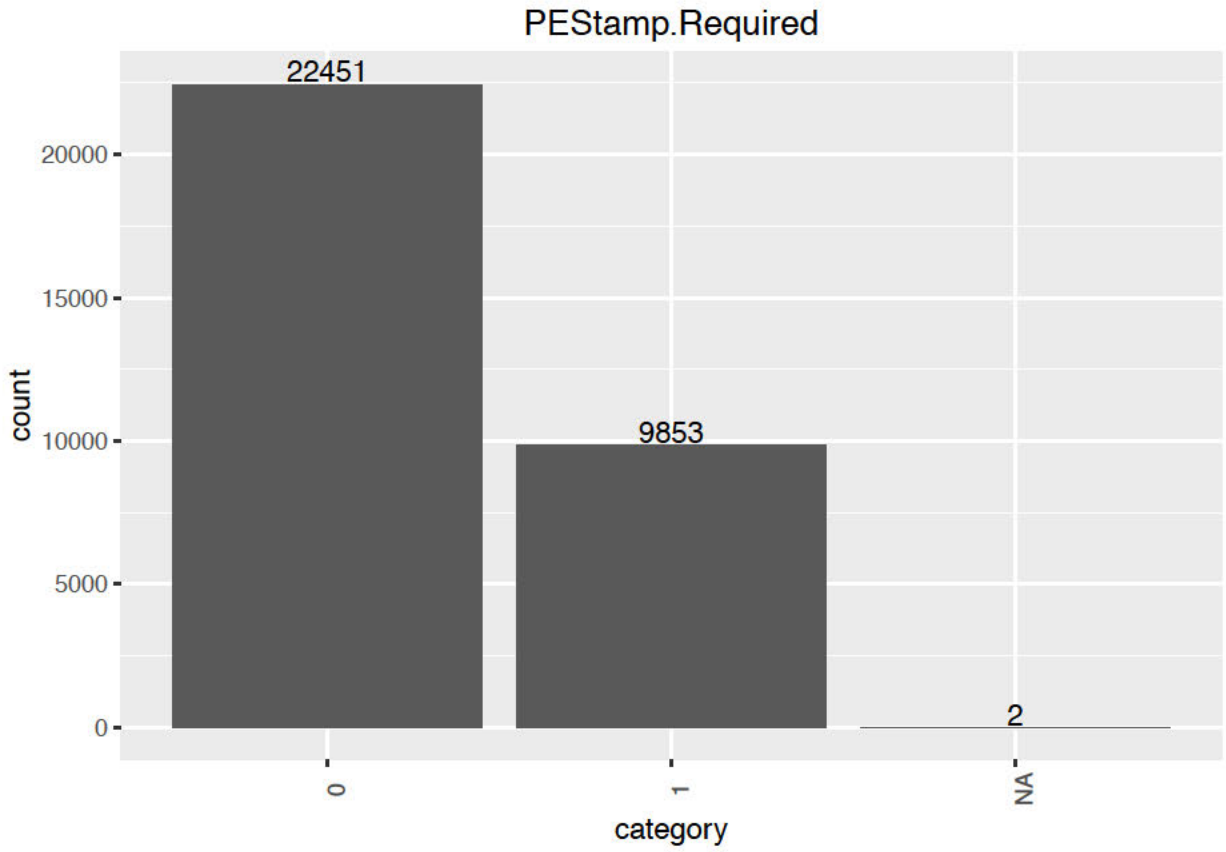

Figure 15: Histogram of entries in categories of PEStamp.Required

```r
df <- df %>% mutate(PEStamp.Required = as.logical(PEStamp.Required))
```

Section 3: Final Cleaning and Write Data to File

To complete the data cleaning task we run two final operations. Both are completed in the code block below.

1. Remove any duplicates
2. Remove any rows of all *NA* values

```r
# remove any duplicates
df <- distinct(df)
# remove row where all values are NA
df <- df[rowSums(is.na(df)) != ncol(df), ]
```

The cleaned data set is written to file. It is uploaded to Canvas for review.

```r
# write cleaned data to file
write.csv(df, file = "hw2_cleaned.csv", row.names = FALSE)
```