

Data Cleaning Presentation

Jonathan Neimann



2024-10-02

We start by loading in the data set

```
strawberry <- read_csv("strawberries25_v3.csv", col_names = TRUE)

## Rows: 12669 Columns: 21
## — Column specification


---


## Delimiter: ","
## chr (15): Program, Period, Geo Level, State, State ANSI, Ag District,
County...
## dbl (2): Year, Ag District Code
## lgl (4): Week Ending, Zip Code, Region, Watershed
##
##  Use `spec()` to retrieve the full column specification for this data.
##  Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

Now we check to make sure that every row is associated with a state.

```
## Is every line associated with a state?

state_all <- strawberry |> distinct(State)

state_all1 <- strawberry |> group_by(State) |> count()

## every row is associated with a state

if(sum(state_all1$n) == dim(strawberry)[1]){print("Yes every row in the data
is associated with a state.")}

## [1] "Yes every row in the data is associated with a state."

## rm(state_all, state_all1)
```

Since the data set has some columns that are all NA, we can create a function that removes columns with only a singular item. We call this function `drop_one_value_col`

```
##/label: function def - drop 1-item columns

drop_one_value_col <- function(df){  ## takes whole dataframe
drop <- NULL

## test each column for a single value
```

```

for(i in 1:dim(df)[2]){
  if((df |> distinct(df[,i]) |> count()) == 1){
    drop = c(drop, i)
  } }

## report the result -- names of columns dropped
## consider using the column content for labels
## or headers

if(is.null(drop)){return("none")}else{

  print("Columns dropped:")
  print(colnames(df)[drop])
  strawberry <- df[, -1*drop]
}
}

## use the function

strawberry <- drop_one_value_col(strawberry)

## [1] "Columns dropped:"
## [1] "Week Ending"      "Zip Code"          "Region"            "watershed_code"
## [5] "Watershed"        "Commodity"

drop_one_value_col(strawberry)

## [1] "none"

```

We can look at the data and see that the county level in the 'Geo Level' column doesn't really tell us much about the data, so we can drop it.

```

unique(strawberry$`Geo Level`)

## [1] "COUNTY" "NATIONAL" "STATE"

strawberry <- strawberry |>
  filter(`Geo Level` == "NATIONAL" | `Geo Level` == "STATE")

```

now examine the rest of the columns

We can see that the data is split into two main categories under the 'Program Column'. There is CENSUS data which deals with strawberry sales, and SURVEY data that deals with the chemicals used for the strawberries. We can split these into two different data rames called straw_cem for CENSUS data and straw_sur for SURVEY data.

```

##/label: split strawberry into census and survey pieces

```

```

straw_cen <- strawberry |> filter(Program=="CENSUS")
straw_cen <- straw_cen |> drop_one_value_col()

## [1] "Columns dropped:"
## [1] "Program"          "Period"          "Ag District"     "Ag District
Code"
## [5] "County"          "County ANSI"

straw_sur <- strawberry |> filter(Program == "SURVEY")
straw_sur <- straw_sur %>% drop_one_value_col()

## [1] "Columns dropped:"
## [1] "Program"          "Ag District"     "Ag District Code" "County"
## [5] "County ANSI"     "CV (%)"

nrow(strawberry) == (nrow(straw_sur) + nrow(straw_cen))

## [1] TRUE

## Move marketing-related rows in strw_b_chem
## to strw_b_sales

```


Now let's look at the survey data first and examine it. We notice that the column 'Domain Category' contains a lot of information. Namely the fact there's a chemical involved, the chemical's use, it's name and it's associated number. We can start by splitting the chemical name and number into two columns named chemical_name and chemical_num.

```

straw_sur <- straw_sur %>%
  mutate(`Domain Category` = gsub(".*: \\((([^=]+) = ([0-9]+)\\)", "\\1,\\2",
`Domain Category`)) %>%
  separate(`Domain Category`, into = c("chemical_name", "chemical_num"), sep
= ",") %>%
  mutate(chemical_name = trimws(chemical_name), # Remove Leading/trailing
whitespace
        chemical_num = as.numeric(trimws(chemical_num))) # Convert
chemical_num to numeric

## Warning: Expected 2 pieces. Additional pieces discarded in 35 rows [126,
219, 310, 401,
## 494, 631, 703, 772, 841, 911, 1139, 1249, 1357, 1465, 1575, 2100, 2222,
2342,
## 2462, 2584, ...].

## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 606 rows
[1, 2, 3, 4, 5,
## 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].

## Warning: There was 1 warning in `mutate()`.
##  In argument: `chemical_num = as.numeric(trimws(chemical_num))`.
## Caused by warning:
## ! NAs introduced by coercion

```

That looks better but we still have a column “domain” that contains multiple information pieces. Namely whether it’s a chemical and its use. We can split that up to a domain column and a use column. We will keep the domain column because there are some rows that contain “total” data marked in this column as well.

```
straw_sur <- straw_sur %>%
  separate(Domain, into = c("Domain", "use"), sep = ",", extra = "merge")

## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 606 rows
## [1, 2, 3, 4, 5,
## 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].
```

Now that Domain Category is taken care of, there is another column that contains lots of information called “Data Item”. This column tells us mostly what we are measuring in the “value column”. I would like to make these measurement units their own column called measurement. So we can use the mutate function to split the columns at the word MEASURED which appears on all the rows that contain measurements.

This column also contains a category that they are processed. Such as processing, fresh market or production. We are also going to make this its own column by separating at the - sign.

```
straw_sur <- straw_sur %>%
  # First, extract the measurement portion and remove it from Data Item
  mutate(measurement = str_extract(`Data Item`, "(?<=MEASURED\\s).*")) %>%
  mutate(`Data Item` = str_remove(`Data Item`, "MEASURED.*")) %>%

  # Now, separate the Data Item into Data Item and category based on the
  # first comma or hyphen
  separate(`Data Item`, into = c("Data Item", "category"), sep = "[,-]",
    extra = "merge", fill = "right")

#seperate_wider_delimiter
```

We notice our measurement column got moved to the end of the data frame with the mutate function, so let’s move that to be placed after ‘Category’ so it makes sense.

```
straw_sur <- straw_sur %>%
  select(1:7, 13, 8:12)
```

We also see the measurement column contains some excess words (IN). So let’s get rid of that as well so the column just displays units.

```
straw_sur <- straw_sur %>%
  mutate(measurement = gsub("\\bIN\\b", "", measurement)) %>% # Remove "IN"
  # as a whole word
  mutate(measurement = trimws(measurement)) # Remove Leading/trailing
  whitespace
```

We also see that our value column is at the end, it makes more sense to me to put it after measurement so will move it there.

```
straw_sur <- straw_sur %>%  
  select(1:8, 13, 9:12)
```

Now our data item column only contains one value (strawberries). However by using the unique function we see some of these have a space after and some of them don't. So we can't use our drop_one_value function because R is registering it as two values. So we can just drop the whole column altogether with a pipe.

```
straw_sur <- straw_sur %>%  
  select(-`Data Item`)
```

Checking through the data. We realize the "Category" Column has some unnecessary commas at the end of the values. We can get rid of those.

```
straw_sur$category <- gsub(",", "", straw_sur$category)
```

The last thing we want to do is split this into two tables at the domain column. One for total and one for chemical. We can call these tables sur_total and sur_chem.

```
sur_total <- straw_sur %>% filter(Domain == "TOTAL")  
sur_chem <- straw_sur %>% filter(Domain == "CHEMICAL")  
sur_total = drop_one_value_col(sur_total)  
  
## [1] "Columns dropped:"  
## [1] "Domain"          "use"              "chemical_name" "chemical_num"  
  
sur_chem = drop_one_value_col(sur_chem)  
  
## [1] "Columns dropped:"  
## [1] "Period"         "Geo Level" "Domain"
```

Our survey is now split into two data tables, sur_total and sur_chemical and looks clean.

Now we can move on to the Census data

We again see that Data Item column in this data frame contains multiple information.

```
straw_cen <- straw_cen |>  
  separate_wider_delim( cols = `Data Item`,  
                        delim = " - ",  
                        names = c("strawberries",  
                                  "Category"),  
                        too_many = "error",  
                        too_few = "align_start"  
                      )
```

We now want to create a table of just the "organic" strawberry sales, so we can isolate this from the main table and create a table called organic_cen. We do this by creating an organic column in our straw_cen data frame and then filter by the word "ORGANIC" within

that column. We can then see that that column just contains two categories, NA and Organic. So once we separate the organic rows out, the only rows left are NA, which we will be able to drop using our function.

```
straw_cen <- straw_cen |>
  separate_wider_delim( cols = strawberries,
                        delim = ",",
                        names = c("strawberries",
                                   "ORGANIC",
                                   "organic_detail"),

                        too_many = "error",
                        too_few = "align_start"
                      )
```

```
straw_cen <- straw_cen |> drop_one_value_col()
```

```
## [1] "Columns dropped:"
```

```
## [1] "strawberries"
```

```
## how many organic rows?
```

```
organic_cen <- straw_cen |> filter(ORGANIC == "ORGANIC")
```

```
sum(is.na(straw_cen$ORGANIC))
```

```
## [1] 662
```

```
straw_cen <- straw_cen[(is.na(straw_cen$ORGANIC)),]
```

```
straw_cen <- straw_cen |> drop_one_value_col()
```

```
## [1] "Columns dropped:"
```

```
## [1] "Year"          "ORGANIC"        "organic_detail"
```

Now we can split the category column in straw_cen into two columns. Measure and bearing to organize the bearing type.

```
straw_cen <- straw_cen |>
  separate_wider_delim( cols = `Category`,
                        delim = " ",
                        names = c("COL1",
                                   "COL2"),
                        too_many = "merge",
                        too_few = "align_start"
                      )
```

```
straw_cen$COL2 <- str_replace(straw_cen$COL2, "WITH ", "")
```

```
straw_cen <- straw_cen |> rename(Measure = COL1, Bearing_type= COL2)
```

Next, we see the column “Domain Category” again contains lots of information (as it did in the straw_sur data frame). We can separate this into two columns called domain and area grown.

```
## remove AREA GROWN and parens  
## change NOT SPECIFIED TO TOTAL
```

```
straw_cen <- straw_cen |> rename(size_bracket = `Domain Category`)
```

```
straw_cen$size_bracket <- str_replace(straw_cen$size_bracket, "NOT  
SPECIFIED", "TOTAL")
```

```
straw_cen$size_bracket <- str_replace(straw_cen$size_bracket, "AREA GROWN: ",  
")
```

Now onto the organic census data. First let's drop all the one value columns

```
organic_cen <- organic_cen |> drop_one_value_col()
```

```
## [1] "Columns dropped:"  
## [1] "ORGANIC"          "Domain"           "Domain Category"
```

Now we can clean “Category” column similar to how we cleaned it with the straw_sur data. By splitting the columns at the word measure.

```
organic_cen <- organic_cen %>%  
  separate(Category, into = c("Category", "measurement"), sep = " MEASURED ",  
  extra = "merge", fill = "right")
```

get rid of commas and 'IN' from the new column it is just units again.

```
organic_cen$Category <- gsub(",", "", organic_cen$Category)  
organic_cen$measurement <- gsub("IN", "", organic_cen$measurement,  
ignore.case = TRUE)
```

Get rid of (D) and replace with NA.

```
#Get rid of D and Z in the value column and replace them as NA  
straw_cen$Value[straw_cen$Value == "(D)"] <- NA  
straw_cen$Value[straw_cen$Value == "(Z)"] <- NA
```

We now have four cleaned tables. One called sur_chem with chemical data, one called sur_total with total survey data, one called straw_cen with non-organic sales data, and one called organic_cen with organic sales data.

We are going to try to fill in some of the missing ‘Value’ data. First we need to make sure the numbers in the Value column are numeric

```
straw_cen <- straw_cen %>%
  mutate(Value = as.numeric(gsub(",", "", Value)))
```

This took me a while and i used chat GPT for some of it, but we were able to create a function that gets ratios from every state using the total acres grown and the total acres bearing and applies that ratio to the value in total acres grown that matched the same size bracket as a value marked NA. It is not perfect but i checked some of the values and they are fairly accurate.

```
fill_na_bearing_values <- function(df) {
  # Create a copy of the data frame to avoid modifying the original
  df_filled <- df

  # Get the unique states
  states <- unique(df$State)

  for (state in states) {
    # Filter data for the current state
    state_data <- df %>% filter(State == state)

    # Extract the GROWN and BEARING values
    grown_value <- state_data %>%
      filter(Bearing_type == "GROWN", Domain == "TOTAL", size_bracket ==
"TOTAL") %>%
      pull(Value)

    bearing_value <- state_data %>%
      filter(Bearing_type == "BEARING", Domain == "TOTAL", size_bracket ==
"TOTAL") %>%
      pull(Value)

    # Calculate the percentage of bearing strawberries
    if (length(grown_value) > 0 && length(bearing_value) > 0 && grown_value >
0) {
      percentage <- bearing_value / grown_value
    } else {
      next # Skip if values are missing
    }

    # Fill in NA values for "BEARING" rows where size_bracket matches
    df_filled <- df_filled %>%
      mutate(Value = ifelse(is.na(Value) & Bearing_type == "BEARING" & State
== state,
                           percentage * Value[which(Bearing_type == "GROWN"
&
                           size_bracket ==
size_bracket &
                           State == state)],
              Value))
```



```

}

return(df_filled)
}

```

I did the same thing for the non-bearing acres here

```

fill_na_non_bearing_values <- function(df) {
  # Create a copy of the data frame to avoid modifying the original
  df_filled <- df

  # Get the unique states
  states <- unique(df$State)

  for (state in states) {
    # Filter data for the current state
    state_data <- df %>% filter(State == state)

    # Extract the GROWN and NON-BEARING values
    grown_value <- state_data %>%
      filter(Bearing_type == "GROWN", Domain == "TOTAL", size_bracket ==
"TOTAL") %>%
      pull(Value)

    non_bearing_value <- state_data %>%
      filter(Bearing_type == "NON-BEARING", Domain == "TOTAL", size_bracket
== "TOTAL") %>%
      pull(Value)

    # Calculate the percentage of non-bearing strawberries
    if (length(grown_value) > 0 && length(non_bearing_value) > 0 &&
grown_value > 0) {
      percentage <- non_bearing_value / grown_value
    } else {
      next # Skip if values are missing
    }

    # Fill in NA values for "NON-BEARING" rows where size_bracket matches
    df_filled <- df_filled %>%
      mutate(Value = ifelse(is.na(Value) & Bearing_type == "NON-BEARING" &
State == state,
                           percentage * Value[which(Bearing_type == "GROWN"
&
                                                    size_bracket ==
size_bracket &
                                                    State == state)],
                           Value))
  }
}

```

```
    return(df_filled)
}
```

last we apply both functions to the straw_cen data to fill in some of the NA's with the correct ratios based on the total numbers.

```
straw_cen = fill_na_bearing_values(straw_cen)
straw_cen = fill_na_non_bearing_values(straw_cen)
```