Lab 5: Data cleaning

> In this lab, you will be guided through the process of cleaning a real-world dataset.

What is data cleaning?

Data cleaning is the process of detecting and fixing mistakes and inaccurate records within a given dataset and stripping away extraneous formatting from raw input data. Mistakes and extraneous formatting appear for a variety of reasons, including data entry errors (typos), inconsistencies during format conversion, unexpected input, coding errors, and the list goes on. Data cleaning and organizing takes up a large portion of a data scientist's time, as each dataset is unique and will require its own custom cleaning workflow. Without data cleaning, these spurious entries in the dataset would affect the data trends and could lead to incorrect conclusions.

About this week's dataset

The City of Chicago has a data portal where you can access information about the city, which includes a list of vehicles that have been towed and impounded within the last 90 days. Illegally parked vehicles, abandoned vehicles and vehicles used for illegal activities may be towed by the Chicago Police Department, the Department of Streets and Sanitation, the Department of Revenue, Aviation and the office of the City Clerk. After a tow request is issued, an inventory number is assigned by the Department of Streets and Sanitation and a truck is dispatched to tow the requested vehicle to a City auto pound.

The dataset saved in the file `towed_vehicles.rds` was retrieved on May 30, 2018.

Variables

Variable	Description
Tow Date	The data the vehicle was towed
Make	Vehicle manufacturer
Style	Two-letter abbreviation of the vehicle’s body type
Model	Vehicle product name
Color	Vehicle’s color
Plate	Vehicle’s license plate number
State	License plate’s state
Towed to Address	Address where car is impounded
Tow Facility Phone	Phone number of auto pound
Inventory Number	Identification number for towed vehicle

Renaming the variables

Although no general recipe exists for cleaning datasets, there are a few standard checks and procedures you can follow as part of the cleanup process. First among those is renaming the variables (columns) within a dataset. Many datasets will come with variable names that are either hard to read due to abbreviations or are hard to use in R programs due to spaces or unusual symbols. For this reason, it is good practice to standardize the variable names. One such standard is called “snake case”, which is where you make all letters lowercase and you represent the spaces with underscores `_`. So, for example, the variable `Tow Date` in our dataset would become `tow_date`.

When we load the `tiydverse` package, it provides us with the `rename` function. We would use `rename` as follows in order to change `Tow Date` to `tow_date`:

```
towed %>%
  rename(tow_date = `Tow Date`)
```

The new name goes to the left of the equals sign, and the old name goes to the right. You can rename several columns at once in the same `rename` command using commas. The general format for renaming several columns at once is:

```
dataset %>%
  rename(
    new_name1 = old_name1,
    new_name2 = old_name2,
    ...
  )
```

1. Use `rename` to rename **all** 10 variables in the dataset so that they all conform to the "snake case" format. Assign the renamed dataset to a variable named `towed_renamed`.

Fixing data types

Sometimes the data importer does not assign an appropriate data type to one or more columns. In this dataset, the `tow_date` column is the character data type, but it should be the `date` data type. Let's fix that.

2. The `lubridate` package has been loaded at the top of your R Markdown document, which provides you with helpful functions for working with dates. The `mdy()` function from `lubridate` can be used to fix the `tow_date` data type. You need to pair `mdy()` with `mutate()` in order to fix the data type:

```r
towed_renamed %>%
  mutate(tow_date = mdy(tow_date))
```

Assign the result to a variable name `towed_renamed_2`.

Checking for discrepancies and fixing them

Now that we've renamed our variables into the convenient snake case format and fixed the data types, let's turn to looking for discrepancies in the dataset. Discrepancies refer to mistakes in data entry, which can be mispellings, inconsistent labeling, and other kinds of typos. For relatively small data sets like this one, we can look for discrepancies by asking R to count the number of occurances of unique values within each categorical variable. The general idea is that entries that have a low count are more likely to be a typo that entries with a high count, especially if the category closely resembles another category with a higher count. We can do this using the `count()` function.

3. Use `count` to count the unique values of the `make` column in `towed_renamed_2` and assign the results to `tow_make_count`. How many unique values are there in the `make` column? Are there any categories that only have a count of 1?

While we could manually inspect the table we just created in `tow_make_count`, it helps to make a visualization to aid us in our cleaning. We can create a bar chart that shows the counts for each category under `make`. Because there are a lot of unique entries, let's split `tow_make_count` into four subsets.

4. We can use `slice` to help us divide the dataset into four relatively equal pieces. To slice off the first 20 entries of the dataset, we would run:

```r
tow_make_count_1 <- tow_make_count %>%
  slice(1:20)
```

Slicing rows 21 through 40 would use `slice(21:40)` instead of `slice(1:20)`, and so on.

For this exercise, slice the dataset into four groups of 20 (or less) rows, which are assigned to variables `tow_make_count_1`, `tow_make_count_2`, `tow_make_count_3`, and `tow_make_count_4`.

Now that the dataset is split up, we can create a bar chart visualization for each one. For the first 20 rows, we would run:

```
ggplot(data = tow_make_count_1) +
  geom_col(mapping = aes(x = fct_rev(make), y = n)) +
  coord_flip() +
  xlab("Make") +
  ylab("count")
```

The `fct_rev()` that surrounds `make` is needed so that the `make` variable is sorted in alphabetical order from top to bottom.

5. Create bar chart visualizations for `tow_make_count_1`, `tow_make_count_2`, `tow_make_count_3` and `tow_make_count_4`, using the above code as a template for getting started.

If you look at the visualizations of the four groups of `tow_make_count`, you may see some obvious problems. For example, for *Dodge* automobiles, the code should be `DODG`, however, there is an erroneous `DODD` entry in the dataset that should be `DODG`. This is an example of what we need to fix, and we can use the `recode()` function in order to do it:

```
towed_renamed_2 %>%
  mutate(make = recode(make, DODD = "DODG"))
```

Just like the `rename` function, we can apply multiple recodes to fix the mispellings. The general format is:

```r
towed_renamed_2 %>%
  mutate(
    make = recode(
      make,
      MISPELLED1 = "CORRECTED1",
      MISPELLED2 = "CORRECTED2",
      MISPELLED3 = "CORRECTED3",
      ...
    )
  )
```

6. Correct two misspellings, `"DODD"` into `"DODG"` and `"BUCI"` into `"BUIC"`, and assign the result to `towed_partial_clean`.

While some of the mistakes may be obvious, not all of them may be. It would be helpful if we had a standardized list of automobile make codes to reference. This is an example of the need to bring in outside information during a data science project, for unless you are already familiar with these codes, you won't know when you have a standard abbreviation and and when you may have a mispelling. In your starter repository, you have been provided with the file `VTR-249.pdf`, which contains a list of standard abbreviations for vehicle makes and body styles put together by the Texas Department of Motor Vehicles.

7. Practice using the guide by opening the PDF file `VTR-249.pdf` and checking whether the *Jaguar* make of automobile should be encoded as `"JAG"` or `"JAGU"`. Write down whether we should use `"JAG"` or `"JAGU"`.

Now it's time to put all of these pieces together and to clean the rest of the `make` column in `towed_partial_clean`. You will need to reference the four bar charts you created earlier as well as the file `VTR-249.pdf` while doing this.

8. Clean the rest of the `make` column by using your bar chart visualizations and the file `VTR-249.pdf` to identify the mispellings. Use `recode` to fix the mispellings as you were shown in the previous exercises. Assign the final result to `towed_make_clean`.

From here, we would also want to check the remaining columns, including the `style` and `model` columns, for mistakes. However, you've done enough data cleaning for one lab, as you can see data cleaning is tedious (and necessary) work!

Additional questions

Now that your dataset has been cleaned, let's ask a couple of basic questions about it so that we can explore the data.

- Which `make` of automobile was impounded the most often in this dataset?
- Which day had the most cars impounded? Which day had the least cars impounded?

How to submit

You can submit the lab report as **.zip or .rar file on Blackboard**. To generate HTML document please select `Knit to HTML` option. You can download any file from RStudio server to your local computer in two steps:

- Click on the checkbox of the file which you want to download.
- Select `More` option and then click on `Export..` to download the file.

After downloading the file, please compress the HTML file and then upload it on Blackboard.

Credits