Case-Study

The case-study will provide and illustrate the following:

1. A pro-tip for setting up a pre-processing data pipepline.
2. The `function` I use often: `clean_names()`.
3. Newly discovered `functions` from `{janitor}`.

Let's dive in…

Imagine being tasked with doing an analysis on Starbucks coffee locations. Your manager has provided you with `raw-data` from coffee chains and requested that you:

1. QA the data for duplicates (by store and by location).
2. Tabulate the various types of Starbucks Ownership:
   - Worldwide &
   - US (lower 48)
3. Deliver a US map that identifies patterns in ownership types.

To streamline your efforts and get swiftly to making that map, you decide to leverage the `{janitor:package}`.

Load our Libraries

```
library(tidyverse)      # Work-Horse Package
library(janitor)        # Data cleaning (+tabulating data)
library(janitor)        # Business Ready Plots
library(ggthemes)       # Clean ggplot theme for Maps
library(USAboundaries) # Get state name/code mapping
```

Let's Get Some Data

For our case-study we are using data from the Tidy Tuesday Project archive.

```
# Import Data ----
# tuesdata <- tidytuesdayR::tt_load("2018-05-07")
```

Pro-Tip: Pre-Processing Pipeline

When working with new data, I'll typically setup up a pre-processing step at the beginning of the script. It typically starts out with no steps and then they get added as I move through my analysis.

The idea is that as you conduct your Exploratory Data Analysis `(EDA)`, you will discover pre-processing steps that need to be added to your pipeline.

In this post, I'll illustrate this technique by adding to our pipeline as we go; however, this data pipeline would live near the top of the script and would not move.

Step 1

Save `raw` data to a variable.

```
# coffee_chains_raw <- tuesdata$week6_coffee_chains
```

Step 2

Immediately save the `raw` data to new variable labeled with the suffix, `processed`.

```
# Beginning of Pre-Processing Pipeline
```

```
coffee_chains_processed <- coffee_chains_raw
```

This obviously has ZERO pre-processing done to the data at this point. The point though is that as you discover areas of your data that require attention, you then can circle back to this pipeline and add those steps.

This may seem odd, but the beauty comes in not having to get further along in your analysis before realizing that you need to do data cleaning steps; if you approach it that way, then you have to go back and rename your variables created along the way – this method allows you to keep working with your `processed` data as you move swiftly through your analysis.

I picked up this `pro-tip` while watching `David Robinson` in his `Tidy Tuesday Screencasts` – check those out here: Tidy Tuesday R Screencasts

```
# Hat-Tip to D-Rob
```

Step 3

Begin `Exploring` your `Data` and conducting your analysis.

At this point, I'll do a bit of `EDA` to familiarize myself with the data I'm working with; this process is always to get a high-level understanding of the data so that I can pick up on nuances along with data integrity issues that need attention (dealt with in the pre-processing pipeline).

Initial Exploration

Let's look at these `raw` data using the `tibble::glimpse()` function.

The `glimpse()` function allows us to quickly assess column names, data-types, and also view a sample of the values contained in each column – you can read more about the `glimpse()` function in my archived post, Examining Data with glimpse().

```
coffee_chains_processed %>%
  tibble::glimpse()
```

```
## Rows: 25,600
## Columns: 13
## $ Brand              "Starbucks", "Starbucks", "Starbucks", "Starbucks", …
## $ `Store Number`     "47370-257954", "22331-212325", "47089-256771", "221…
## $ `Store Name`       "Meritxell, 96", "Ajman Drive Thru", "Dana Mall", "T…
## $ `Ownership Type`   "Licensed", "Licensed", "Licensed", "Licensed", "Lic…
## $ `Street Address`   "Av. Meritxell, 96", "1 Street 69, Al Jarf", "Sheikh…
## $ City               "Andorra la Vella", "Ajman", "Ajman", "Abu Dhabi", "…
## $ `State/Province`   "7", "AJ", "AJ", "AZ", "AZ", "AZ", "AZ", "AZ", "AZ",…
## $ Country            "AD", "AE", "AE", "AE", "AE", "AE", "AE", "AE", "AE"…
## $ Postcode           "AD500", NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, "31…
## $ `Phone Number`     "376818720", NA, NA, NA, NA, NA, NA, NA, "26670052",…
## $ Timezone           "GMT+1:00 Europe/Andorra", "GMT+04:00 Asia/Dubai", "…
## $ Longitude          1.53, 55.47, 55.47, 54.38, 54.54, 54.49, 54.49, 54.6…
## $ Latitude           42.51, 25.42, 25.39, 24.48, 24.51, 24.40, 24.40, 24.…
```

Immediately, we can see that our column names are not optimal for analysis. Personally, I'm VERY biased towards `snake_case` and therefore always like to get column names into that format.

janitor::clean_names()

In comes `{janitor::clean_names}` to the rescue ⛑️

By default, `clean_names()` outputs column naming with the `snake_case` format – maybe this is one of the reasons that it's in my top 10 for favorite functions in `R`.

Let's test it out on our coffee data.

```
# clean_names() with default naming
coffee_chains_processed %>%
  janitor::clean_names() %>%
  base::names()
```

```
##  [1] "brand"          "store_number"   "store_name"     "ownership_type"
##  [5] "street_address" "city"           "state_province" "country"
##  [9] "postcode"       "phone_number"   "timezone"       "longitude"
## [13] "latitude"
```

Awesome!

You'll notice the `function` took care of the `/` in `State/Province` and replaced it with an underscore – simply amazing 😎

Naming Convention Options

If you prefer a different naming convention – I'm not sure why you would 🙄 – then you can use the `case` argument.

```
# clean_names() with diff. naming convention
coffee_chains_processed %>%
  clean_names(case = "small_camel") %>%
  names()
```

```
##  [1] "brand"         "storeNumber"   "storeName"     "ownershipType"
##  [5] "streetAddress" "city"          "stateProvince" "country"
##  [9] "postcode"      "phoneNumber"   "timezone"      "longitude"
## [13] "latitude"
```

Pre-Processing Addition

Now let's add this step to our pre-processing pipeline.

```
# Adding to our Pre-Processing Pipeline
coffee_chains_processed <- coffee_chains_raw %>%

  # clean up column names
  janitor::clean_names()
```

janitor::get_dupes()

`get_dupes()` is at the top of the list for newly discovered `functionality` within the `{janitor}` package.

This is one of those things you need to do often (check for duplicates) and `{janitor}` makes it simple.

Going back to our `case-study`, our manager asked us to check for duplicated records (a common `data-cleaning` and `EDA` step).

Let's subset our data and investigate.

```
coffee_chains_processed %>%

  # subset data by store and by location
  dplyr::select(brand, store_number,
                city, state_province, country) %>%

  # identify duplicated records
```

```
  janitor::get_dupes()
```

```
## # A tibble: 2 x 6
##   brand     store_number city  state_province country dupe_count
##
## 1 Starbucks 19773-160973 Seoul 11             KR               2
## 2 Starbucks 19773-160973 Seoul 11             KR               2
```

Using `janitor::get_dupes()` we've quickly identified a potential issue: store number `19773-160973` has duplicated records.

Let's investigate further.

```
# filter to store with dupes
coffee_chains_processed %>%

  # filter to store and glimpse data
  dplyr::filter(store_number == "19773-160973") %>%
  glimpse()
```

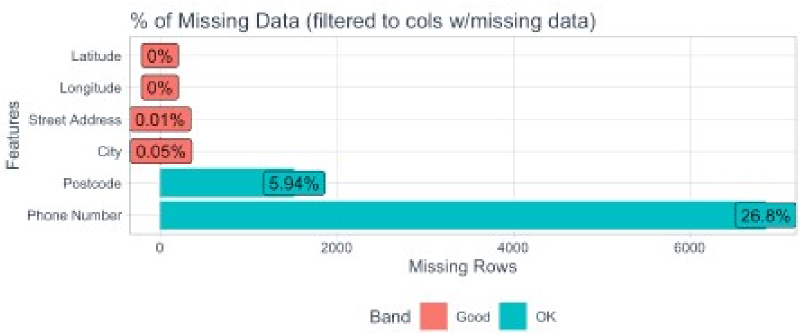
```
## Rows: 2
## Columns: 13
## $ brand          "Starbucks", "Starbucks"
## $ store_number   "19773-160973", "19773-160973"
## $ store_name     "Yoido IFC Mall - 1F", "Yoido IFC Mall - 1F"
## $ ownership_type "Joint Venture", "Joint Venture"
## $ street_address "23 & 23-1, Yoido-Dong, Yongdongpo-Gu, 1F, #101", "23 …
## $ city           "Seoul", "Seoul"
## $ state_province "11", "11"
## $ country        "KR", "KR"
## $ postcode       "153-023", "153-023"
## $ phone_number   NA, NA
## $ timezone       "GMT+09:00 Asia/Seoul", "GMT+09:00 Asia/Seoul"
## $ longitude      NA, 126.92
## $ latitude       NA, 37.53
```

Look carefully and you'll notice that the latitude/longitude are missing for one of these records.

We need lat/long for mapping and so we will want to prioritize the records with those data. Also, we don't want duplicated records to interfere with our tabulations later on in this analysis.

Let's quickly look and see how much data is missing from the lat/long columns.

```
# plot missing data (using raw data)
DataExplorer::plot_missing(
  title    = "% of Missing Data (filtered to cols w/missing data)",
  data     = coffee_chains_raw,
  ggtheme = tidyquant::theme_tq(),
  missing_only = TRUE)
```



The plot shows that `0%` of data are missing for lat/long leading me to believe that the store identified earlier is the only record with missing data (insignificant amount when plotted).

We will filter that record out in our `data-cleaning` step.

Pre-Processing Addition

Now let's add this step to our pre-processing pipeline

```
# Adding to our Pre-Processing Pipeline
coffee_chains_processed <- coffee_chains_raw %>%

  # clean up column names
  janitor::clean_names() %>%

  # filter out records missing lat/long values
  dplyr::filter(!is.na(latitude), !is.na(longitude))
```

Let's use `get_dupes()` to confirm the problem is solved

```
coffee_chains_processed %>%

  # subset data
  dplyr::select(brand, store_number, city, state_province, country) %>%

  # identify duplicated records
  janitor::get_dupes()
```

```
## # A tibble: 0 x 6
## # … with 6 variables: brand , store_number , city ,
## #   state_province , country , dupe_count
```

Starbucks Analysis

Now that we've done our due diligence in being sure we've dealt with data issues, let's knock out this analysis by
tabulating these data and compiling a map, or two 🤓

Before doing so, let's add one final step to our pre-processing data pipeline.

Pre-Processing Addition

The final step is to subset the columns needed to complete the analysis.

```
# Adding to our Pre-Processing Pipeline
coffee_chains_processed <- coffee_chains_raw %>%

  # clean up column names
  janitor::clean_names() %>%

  # filter out records missing lat/long values
  dplyr::filter(!is.na(latitude), !is.na(longitude)) %>%

  # subset columns for analysis
  dplyr::select(brand, ownership_type, country,
                state_province, latitude, longitude)
```

View Data

```
# view first 5 rows
```

```
coffee_chains_processed %>% head(5)
```

```
## # A tibble: 5 x 6
##   brand     ownership_type country state_province latitude longitude
##
## 1 Starbucks Licensed       AD      7                  42.5      1.53
## 2 Starbucks Licensed       AE      AJ                 25.4     55.5
## 3 Starbucks Licensed       AE      AJ                 25.4     55.5
## 4 Starbucks Licensed       AE      AZ                 24.5     54.4
## 5 Starbucks Licensed       AE      AZ                 24.5     54.5
```

Tabulate Data (worldwide)

Let's start with looking at Ownership Types worldwide.

`janitor::tabyl` stuck out to me because the ease with which to generate frequency tables.

Check it out.

```
# generate frequency table
coffee_chains_processed %>%

  # filter data
  dplyr::filter(brand == "Starbucks") %>%

  # tabulate and arrange data
  janitor::tabyl(ownership_type) %>%
  arrange(desc(percent)) %>%

  # formatting
  janitor::adorn_totals() %>%
  janitor::adorn_pct_formatting() %>%
  rmarkdown::paged_table()
```

Using just the `tabyl` function we were able to generate frequencies along with the percent of total.

However, `{janitor}` is packed full of other goodies – the creator(s) have crafted a number of `adorn` options for formatting our outputs. I used the `adorn_totals` and `adorn_pct_formatting` to tidy up and make our table ready for presentation.

Simply Amazing 😎

Tabulate Data (US, lower 48)

```
# generate frequency table
coffee_chains_processed %>%

  # filter data
  dplyr::filter(brand   == "Starbucks",
                country == "US",
                state_province != "AK",
                state_province != "HI") %>%

  # tabulate and arrange data
  janitor::tabyl(ownership_type) %>%
  arrange(desc(percent)) %>%

  # formatting
  janitor::adorn_totals() %>%
  janitor::adorn_pct_formatting() %>%
```

```
rmarkdown::paged_table()
```

> All Starbucks are either company owned, which is almost all of them, or else they're "licensed" locations, which are the Starbucks in airports, supermarkets, etc. – Charles Partrick

Map Starbucks Locations

Now lets make those maps and get this analysis wrapped up.

Lets start by getting a general sense of where in the US these Starbucks are located.

Data Manipulation

```
# Data Manipulation
starbucks_lower_48 <- coffee_chains_processed %>%

  # filter data
  dplyr::filter(brand   == "Starbucks",
                country == "US",
                state_province != "AK",
                state_province != "HI")
```

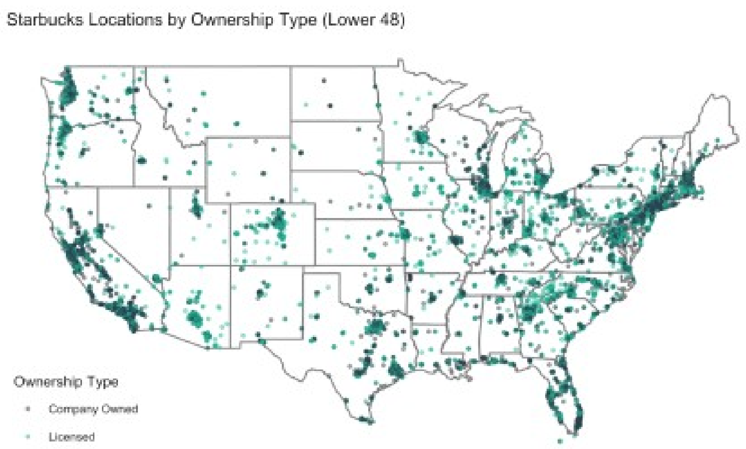
Data Visualization

```
# Data Visualization
starbucks_lower_48 %>%

  # setup ggplot canvas + US borders
  ggplot(aes(longitude, latitude, color = ownership_type)) +

  # add geometries
  borders("state") +
  geom_point(size = .75, alpha = 0.5) +

  # formatting
  ggthemes::theme_map() +    # remove x/y for tidy map
  coord_map() +              # scales map (simple approach)
  scale_color_manual(values = c("#2c3e50", "#18BC9C")) +
  labs(title = "Starbucks Locations by Ownership Type (Lower 48)",
       color = "Ownership Type")
```



That's a solid map but I think we can do better to identify patterns in ownership types.

Let's calculate the ratio of Corporate (Company Owned) vs. Licensed ownership and map that at the state level.

Data Acquisition (state boundaries)

```
# Get state level lat/long table
states <- ggplot2::map_data("state") %>%
  tibble() %>%
  mutate(region = str_to_title(region))
```

Data Manipulation

```
# Data Manipulation
ownership_ratios_by_state <- starbucks_lower_48 %>%

  # count ownership types by state
  group_by(state_province, ownership_type) %>%
  summarize(n = n()) %>%
  ungroup() %>%

  # pivot data and calculate ratios
  pivot_wider(names_from = ownership_type,
              values_from = n) %>%
  clean_names() %>%
  mutate(corp_vs_lic = company_owned/licensed) %>%

  # join to get state names from codes
  left_join(USAboundaries::state_codes %>%
          select(state_name, state_abbr),
          by = c("state_province" = "state_abbr")) %>%

  # reorder columns
  select(state_name, everything())
```

View Data

```
ownership_ratios_by_state %>% head()
```

```
## # A tibble: 6 x 5
##   state_name  state_province company_owned licensed corp_vs_lic
##
## 1 Alabama     AL                        48       36       1.33
## 2 Arkansas    AR                        35       19       1.84
## 3 Arizona     AZ                       196      283       0.693
## 4 California  CA                      1943      839       2.32
## 5 Colorado    CO                       227      250       0.908
## 6 Connecticut CT                        83       35       2.37
```

Data Visualization

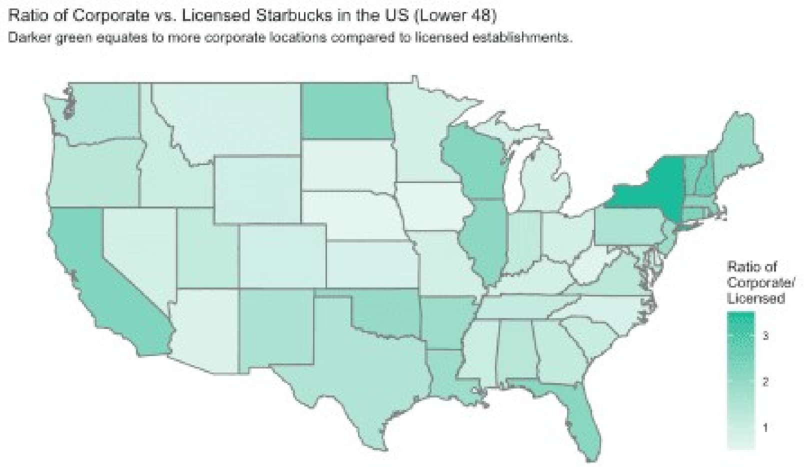
```
ownership_ratios_by_state %>%

  # join to get state boundaries (lat/long)
  left_join(states, by = c("state_name" = "region")) %>%

  # setup ggplot canvas + US borders
  ggplot(aes(long, lat, fill = corp_vs_lic, group = group)) +

  # add geometries
  geom_polygon() +
  ggplot2::borders("state") +
```

```
  # formatting
  ggthemes::theme_map() + # remove x/y for tidy map
  theme(legend.position = c(.9, .05)) +
  coord_map(projection = "mercator") + # scales map projection
  scale_fill_gradient2(low = "white", high = "#18BC9C", ) +
  labs(title = "Ratio of Corporate vs. Licensed Starbucks in the US (Lower 48)",
       subtitle = "Darker green equates to more corporate locations compared to
licensed establishments.",
       fill = "Ratio of\nCorporate/\nLicensed")
```



This represent the data in a way that helps us identify patterns – our manager will be pleased