# What to Expect

I’m excited to share pro-tips that will expedite your process for cleaning and standardizing column names in your data; this is a critical yet sometimes overlooked step in the cleaning + tidying of data.

There are a couple of handy functions() available in R to help effectively execute these tasks.

By the end of this short article you’ll have a couple of new tricks up your sleeve for getting those column names just the way you want them 

# Data Wrangling Toolkit

The clean\_names() function from the janitor library. The set\_names() function from the purrr library.

# Load our Libraries

library(tidyverse) # Work-Horse Package library(tidytuesdayR) # Access Data from Tidy Tuesday library(janitor) # Data Cleaning Package library(purrr) # Functional Programming Toolkit

# Let’s Get Some Data

I’m grabbing a couple of data-sets from the Tidy Tuesday Project that will help us walk through a couple of examples together.

# Get Marine Mammal Data

cetacean\_week <- tidytuesdayR::tt\_load("2018-12-18") cetacean\_raw\_tbl <- cetacean\_week$allCetaceanData

# Get NFL Salary Data

nfl\_salary\_week <- tidytuesdayR::tt\_load("2018-04-09") nfl\_salary\_raw\_tbl <- nfl\_salary\_week$nfl\_salary

Each of these data-sets contain column naming useful for emphasizing the value in the aforementioned functions.

Let’s start with the janitor library and it’s nifty function called clean\_names().

# Janitor Makes Life Easy

My head exploded  when learning about the Janitor library – it’s one of my favorite’s and I use the

clean\_names() function ALL the time.

Standardizing our naming convention upfront in our data cleaning pipeline can save enormous amounts of time downstream. I’m a big fan of the  snake\_case naming convention and so I typically like the columns of my data to follow that pattern.

Fortunately, the janitor::clean\_names() function has built in functionality to programmatically clean up our column names – my favorite part is that by default it favors the snake\_case naming convention.

# Let’s Look at an Example

Pulling a few columns from our marine-mammal data we see that our columns are not in our preferred

snake\_case convention.

# Get subset of columns for example

cetacean\_subset\_tbl <- cetacean\_raw\_tbl %>%

# Select columns using helper\_functions() select(contains("origin"), contains("date"), COD)

# Transpose Data to view Column Names glimpse(cetacean\_subset\_tbl)

## Rows: 2,194

## Columns: 6

## $ originDate 1989-04-07, 1973-11-26, 1978-05-13, 1979-02-03, 1979-…

## $ originLocation "Marineland Florida", "Dolphin Research Center", "SeaW… ## $ statusDate NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N… ## $ transferDate NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N… ## $ entryDate 1989-04-07, 1973-11-26, 1978-05-13, 1979-02-03, 1979-… ## $ COD NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA…

# Mixed Naming, Let’s Standardize

As you can see we’ve got columns in lowerCamel and also in UPPERCASE. To standardize, let’s now use the clean\_names() function to tidy these up.

# Clean up Column Names + Glimpse Output cetacean\_subset\_tbl %>%

clean\_names() %>% glimpse()

## Rows: 2,194

## Columns: 6

## $ origin\_date 1989-04-07, 1973-11-26, 1978-05-13, 1979-02-03, 1979…

## $ origin\_location "Marineland Florida", "Dolphin Research Center", "Sea… ## $ status\_date NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, … ## $ transfer\_date NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, … ## $ entry\_date 1989-04-07, 1973-11-26, 1978-05-13, 1979-02-03, 1979… ## $ cod NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N…

**Those Column Names Look Great!**

Now imagine you have a picky partner/colleague who insists on a format like ALLCAPS – you’ve tried to convince them otherwise but they insist 

# Standardize Column Naming - ALLCAPS cetacean\_cols\_allcaps\_tbl <- cetacean\_subset\_tbl %>%

clean\_names(case = "all\_caps")

# Glimpse Output cetacean\_cols\_allcaps\_tbl %>% glimpse()

## Rows: 2,194

## Columns: 6

## $ ORIGIN\_DATE 1989-04-07, 1973-11-26, 1978-05-13, 1979-02-03, 1979…

## $ ORIGIN\_LOCATION "Marineland Florida", "Dolphin Research Center", "Sea… ## $ STATUS\_DATE NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, … ## $ TRANSFER\_DATE NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, … ## $ ENTRY\_DATE 1989-04-07, 1973-11-26, 1978-05-13, 1979-02-03, 1979… ## $ COD NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N…

# Awesome!

Nice way to be a team-player 

# Next Example: set\_names()

With clean\_names() in our tool bag, we can now combine it with set\_names() to programmatically standardize ALL of our column names using advanced techniques.

Let’s take a quick peak at the columns from the NFL Salary data.

nfl\_salary\_raw\_tbl %>% names()

## [1] "year" "Cornerback" "Defensive Lineman" ## [4] "Linebacker" "Offensive Lineman" "Quarterback"

## [7] "Running Back" "Safety" "Special Teamer" ## [10] "Tight End" "Wide Receiver"

Now imagine instead of requiring snake\_case, the columns need to be lower-case with a dash instead of an

underscore in between words.

The set\_names() function allows us to Set the Names of a Vector programmatically.

Using the names() function above, we can pass a vector of our column names and manipulate each name in similar fashion.

Let’s look at an example.

nfl\_salary\_raw\_tbl %>% clean\_names() %>% names()

## [1] "year" "cornerback" "defensive\_lineman" ## [4] "linebacker" "offensive\_lineman" "quarterback"

## [7] "running\_back" "safety" "special\_teamer" ## [10] "tight\_end" "wide\_receiver"

We’ve effectively used clean\_names() to quickly clean up our column names. However, we still need to replace those underscores with dashes.

Check this out 

nfl\_salary\_raw\_tbl %>% clean\_names() %>%

set\_names(names(.) %>% str\_replace\_all("\_", "-")) %>% glimpse()

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ## | Rows: 800 | |  | | | | | | | | |
| ## | Columns: 11 | |
| ## | $ year | | 2011, 2011, 2011, 2011, 2011, | | | | | 2011, 2011, 2011, 2… | | | |
| ## | $ | cornerback | 11265916, | 11000000, | | 10000000, | | 10000000, | | 10000000,… | |
| ## | $ | `defensive-lineman` | 17818000, | 16200000, | | 12476000, | | 11904706, | | 11762782,… | |
| ## | $ | linebacker | 16420000, | 15623000, | | 11825000, | | 10083333, | | 10020000,… | |
| ## | $ | `offensive-lineman` | 15960000, | 12800000, | | 11767500, | | 10358200, | | 10000000,… | |
| ## | $ | quarterback | 17228125, | 16000000, | | 14400000, | | 14100000, | | 13510000,… | |
| ## | $ | `running-back` | 12955000, | 10873833, | | 9479000, 7700000, 7500000, 70… | | | | | |
| ## | $ | safety | 8871428, | 8787500, | 8282500, | | 8000000, | | 7804333, | | 7652… |
| ## | $ | `special-teamer` | 4300000, | 3725000, | 3556176, | | 3500000, | | 3250000, | | 3225… |
| ## | $ | `tight-end` | 8734375, | 8591000, | 8290000, | | 7723333, | | 6974666, | | 6133… |
| ## | $ | `wide-receiver` | 16250000, 14175000, 11424000, 11415000, 10800000,… | | | | | | | | |

At first, I was puzzled by the names(.) component and didn’t understand what the period was doing. In the course I learned that using the dot (.) enables passing the incoming tibble to multiple-spots in the

function.

set\_names() is a vectorized function and so the first argument is a vector. The dot functionality in R allows us to take the incoming tibble and pass it to the names(.) function. Once we have the names in a vector we use the str\_replace\_all() function to replace the underscore with a dash.

The str\_replace\_all() function uses regular expression pattern matching and so the options are endless for how creative you can get here.

# Wrap-Up

That’s it for today!

We used clean\_names() and set\_names() to effectively standardize our column naming conventions.