SESSION 3: [column calculation/manipulation]

For this training session we will go over the case_when(), rowSum() and best practice when manipulating columns.

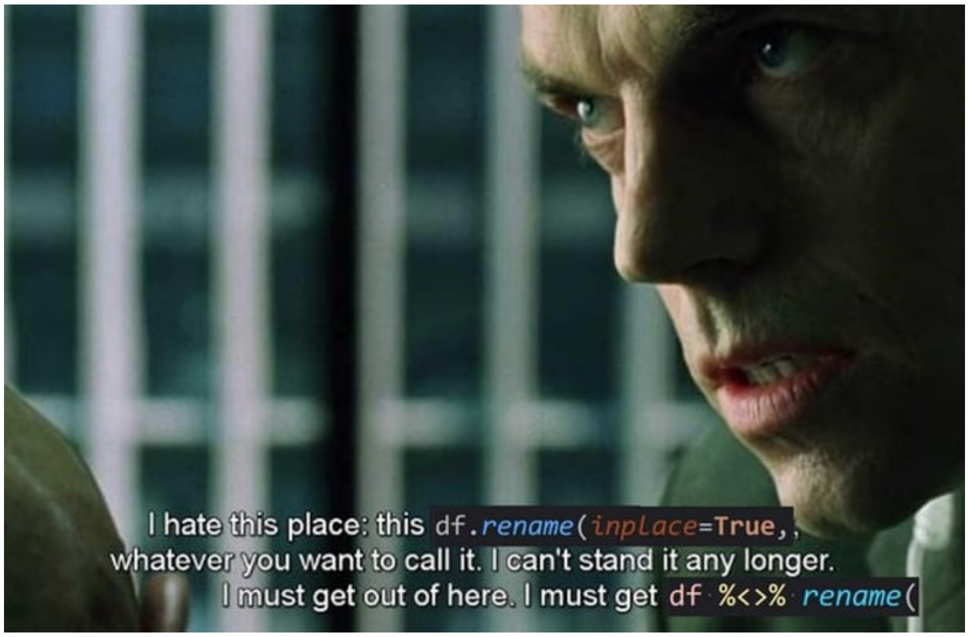
The code below is from session 1. We are standarizing the columns with janitor::clean_names() and renaming the columns.

session 1 code

```r
# import data
df_csv <- read.csv(paste0(getwd(), "/example_data.csv"))

# standarize col names
df_csv_clean_names <- df_csv %>% clean_names()

# rename col names
df_csv_new_column_names <- df_csv_clean_names %>%
  rename("ethnicity" = "hispanic", "exercise" = "excerise")

# reorder col names
df_csv_select <- df_csv_new_column_names %>%
  select(dob, ethnicity, race, sex, zip_code, insurance, exercise, everything())

# rename object for clarity
df_csv_formating <- df_csv_select
```

session 1 code altnerative w/ pipe operator

This is the same output as above, just a different way in writing it. Note how the pipe operator condenses the code so we do not have to initialized a new dataframe object after every manipulation to x dataframe. . . im looking at you python.


I hate this place: this df.rename(inplace=True,
whatever you want to call it. I can't stand it any longer.
I must get out of here. I must get df %<>% rename(

```r
df_csv <- read.csv(paste0(getwd(), "/example_data.csv"))

df_csv_formating <- df_csv %>%
  clean_names() %>%
  rename("ethnicity" = "hispanic", "exercise" = "excerise") %>%
  select(dob, ethnicity, race, sex, zip_code, insurance, exercise, max_squat, everything())
```

session 3 code

Now we will move to manipulating columns. For now lets clean the race column using the case_when(), remember to standardize the race column by converting into lower case so you don't have to write more conditions in the case_when(). Then calculate the average person lift throughout the sum of the 3 lifts (squat, deadlift and bench). We will also get the value for the average strength of the overall contestants in the power lifting meet.

3 outout in code below: - clean race column - create column where it calucate x contestant average weight of their combined 3 lifts - find the value of the overall average strength in this power lifting meet

note: there are many ways to standardize/clean a dataset so if you have an idea on how to manipulate but you do not know how.... search it up on google! Generally your question has been asked and answer previously. Asking the right question is 90% coding... then 10% reading documentation if youre cooked... jk documentation is great too.

```r
df_clean_race_strength <- df_csv_formating %>%
  mutate(
    lower_case_race = tolower(race), # standardize col so we dont have write more in case_when()
    edit_race = case_when(
      lower_case_race == "caucasian" ~ "white",
      lower_case_race == "am-af" ~ "black",
      TRUE ~ lower_case_race
    ),
    race = edit_race, # rename clean race column to race (overwrites old unclean race column)
    avg_strength = round(rowMeans(select(., max_squat, max_bench ,max_deadlift))) # find the average li
  ) %>%
  select(-c("lower_case_race", "edit_race"))

# check if this the intended race output
vec_race <- c("white","black", "asian", "mixed", "other") %>% sort()

if (identical(sort(unique(df_clean_race_strength$race)), vec_race) == TRUE){
  insight::print_color("PASS: vectors match", "green")
} else {
  insight::print_color("!!FAIL: vectors NO match!!", "red")
}

# average strengh of contestants in dataset
mean_strenth_overall <- sum(df_clean_race_strength[,8:10]) / nrow(df_clean_race_strength) # use nrow(),
```

Note in the line below, I use nrow() instead of the actual number of how many rows in the dataset. Now both method works but what if I have the formatted dataset but from a different lift meet? Reusing the code with the actual number of rows from this code will be incorrect because the new dataset may not have the same number of rows (contestant). That is why we use nrow(), that way the code can be reused/maintained properly. We want to write code that is easily reusable/maintainable in the future.

here here is an example below:

```
new_lift_meet <- head(df_clean_race_strength, 15) # new dataset with less than 5 constestant

# correct
correct_new_mean_strenth_overall <- sum(new_lift_meet[,8:10]) / nrow(new_lift_meet) # use nrow(), not t
correct_new_mean_strenth_overall
```

```
## [1] 532.3333
```

```
# incorrect
incorrect_new_mean_strenth_overall <- sum(new_lift_meet[,8:10]) / 20 # lets say i reused the code from
incorrect_new_mean_strenth_overall
```

```
## [1] 399.25
```

See how "correct_new_mean_strength_overall" object is 532.33 vs "incorrect_new_mean_strength_overall" object is 399.25. Sometimes we may overlook that just because the code ran with no issues therefore the output is correct. To avoid make sure you test your desired outputs and write code where reusable in the future. Its much easier to use a function that pull x value if the x value is not a constant when reusing code.