Pivots and Levels Case Assignment

Welcome to the pivots and levels case! .

All of the activities for this assignment are to be completed in a separate R script in this repository. I've provided some starter code in the `pivots_and_levels_case.R` file. You can write all of your code in that file from within RStudio, then save your changes, commit, and push those changes to GitHub before the deadline to submit.

Have fun!

Assignment instructions

Intro / Setup

Follow the instructions in the beginning section ("Intro / Setup") in the `pivots_and_levels_case.R` file. You'll note that we aren't using a single dataset for this assignment, but will instead be reading in a series of datasets and practice our pivoting and data assembly in a variety of scenarios. Those read commands are already provided for you throughout the assignment.

ATTENTION: Because pivoting data can be somewhat tough to grasp, I hereby promise that **there are no intentional "catches" in the data** you'll be using, nothing that is intended to give you "real-world" experience with messiness. Pivoting *can be* an effective cleaning tool to help solve some specific issues, but we won't be doing much of that in any of these problems. The one exception is the soccer player dataset (last problem), where I'll carefully outline what the data issue is and then walk you through each step of the cleaning process. (And don't worry - we'll return to messy data with a few surprises in a future assignment. Because it's good for you.)

This first scenario uses a dataset that summarizes various driving statistics by US state. Use the provided command to read the data into the `bad_drivers_long` object as a tibble. You'll see that the data is organized in long format by state.

Your task is to display the dataset in a wider format such that each row represents a state with its associated statistics. Once the data is in the correct format, you can convert each of the columns that start with "perc_" (which are all percentages) from their integer form to a more useful decimal form.

> *Partial answer:*

```
1 Alabama  18.8  0.39  0.3  0.96  0.8  785.  145.
```

1. Save the resulting tibble (51 rows, 8 columns) to `bad_drivers_wide` .

This next scenario uses a sales contacts dataset. Use the provided command to read the data into the `addr` object as a tibble. This data contains (fake) contact information that a salesperson has gathered while acting as the sole salesperson at a small company. You'll see that the data is stored efficiently in a long format.

The salesperson is hiring a new junior salesperson who will be taking over half of the accounts. Their plan is to split the contacts between the two of them according to region. (Use the provided command to read the region data into `regions` as a tibble.)

Your job is to figure out a reasonable way to split up the contacts so that about half are assigned to each salesperson. To do this, you'll have to join the region data to the address data and then play with a few possible groupings to come up with something that is a reasonable split. The final data should have one row per sales contact, and should additionally contain a `region` column indicating the region where that contact is located and a `sales_assignment` column indicating to whom that sales contact is assigned. (You can assume the salespeople are named "Jack" and "Jill".)

> *Hint 1: Think about the final desired format and where in your pipe chain would be the best place for you to join in the region data. Note also that some of the contact information might not be complete, but all contacts should be assigned to one of the two salespeople (i.e., there should not be any `NA` values in the `sales_assignment` column.)*

> *Partial answer:*

```
3 Art  Venere  Chemel, James L Cpa  NA  NA  NA  856-264-4130 art@venere.org  NA  Jill
```

2. Save the resulting tibble (28 rows, 10 columns) to `sales_regions`.

Next is a song chart dataset representing various songs and their reign on the list of top 100 songs. Use the provided command to read the data into the `songs` object as a tibble. The data consists of songs, their associated artists, and then other information about where each song was ranked in the top 100 songs from week to week, with each song's `start_date` representing the date that each song first entered the top 100. (The `start_date` is always a Saturday because the top 100 songs are apparently calculated each week on Saturdays. Who knew? Not I.)

Your first task is to calculate the number of weeks that each song was rated anywhere in the top 100 songs. (The way to interpret this question is to count the number of non-empty values across all of the columns for each row in the dataset, since any non-empty number represents a ranking.)

> *Hint 1: You could find a row-based function to count up all of the non-empty values in each row, but that would be a really inefficient way to go about it. Think about reshaping the data to make this logic easier.*

> *Hint 2: As there are 317 songs represented in the dataset, your final answer should consist of exactly 317 rows, and probably just 2 or 3 columns (one for the song title and one for the number of weeks the song spent in the top 100, and [optionally] one for the artist who performed the song). The names of those columns aren't important, but they should contain the data I just described.*

> *Partial answer of one way this could be displayed:*

```
# A tibble: 317 × 3
   artist         track                      n
   <chr>          <chr>                  <int>
 1 2 Pac          Baby Don't Cry (Keep...     7
 2 2Ge+her        The Hardest Part Of ...     3
 3 3 Doors Down   Kryptonite                53
```

3. Save the resulting tibble to `songs_time_in_100`.

> ***Bonus Challenge**: After you've saved the result above, see if you can figure out how to count the number of "ties" there are in terms of artists (not songs) who had spent the same number of weeks in the top 100 for one or more songs they performed. What is the week count for which there were the most "ties" in the data? How many artists were tied for that number? Answers: 20 weeks, 39 artists.*

> *Surprising how large that group is, relative to the next highest (6 weeks, with 8 artists). (Just make sure you don't save over your answer to the problem.)*

Starting with the same `songs` tibble (not the one you saved in the previous problem), find the artist who has enjoyed the #1 spot for the most weeks (not necessarily consecutive) for a song (or songs) that he/she has performed. In finding the answer, include in your result tibble all artists who have ever had a #1 ranking, but sort the result such that the artist with the most #1 weeks is at the top.

> *Hint 1: If you can pivot the data correctly, then a week with a #1 spot is really just any row where the "rank" on the top 100 list is equal to 1. Then the answer is simply a matter of finding the artist with the **most** weeks (rows) matching those criteria.*

> *Answer (just the top row shown):*

```
# A tibble: 15 × 2
   artist                n
   <chr>             <int>
 1 Destiny's Child      14
```

4. Save the resulting tibble to `artist_most_at_1`.

Starting with the same `songs` tibble (not the ones you saved in the previous two problems), filter the pivoted data to the week when each artist's song "peaked," meaning the week number that it hit its maximum ranking. (And remember that "maximum ranking" is the ranking closest to 1!) Ensure that the week in question is found within a column called `week_number` and that the column is, in fact, formatted as a number rather than a character (string).

> *Hint 1: Remember that, even when you've pivoted a tibble, the result is still just a tibble that can be **grouped** and **sliced** in order to solve problems like these. Just think about what you have after pivoting and how it can be used to get to the right answer.*

> *Hint 2: Also note that some artists stayed at the same "peak" ranking for more than one week. In that case, you should list the **first** week that the artist reached that peak ranking. (In other words, you should not allow ties when computing the highest ranking.)*

> *Answer (top 3 rows shown):*

```
# A tibble: 317 × 5
# Groups:   artist, track [317]
   artist         track                   start_date week_number  rank
   <chr>          <chr>                   <chr>            <dbl> <dbl>
 1 2 Pac          Baby Don't Cry (Keep... 2/26/2000            3    72
 2 2Ge+her        The Hardest Part Of ... 9/2/2000             2    87
 3 3 Doors Down   Kryptonite              4/8/2000            32     3
```

5. Save the resulting tibble to `peak_week`.

> ***Bonus Challenge**: After completing and saving the tibble above, see if you can determine the date on which the "peak week" began. Partial answer: 2 Pac's Baby Don't Cry peaked during the week starting March 18th, 2000. (Again, make sure you don't save over your answer to the problem.)*

The last (required) problem uses a dataset containing soccer players and a few of their statistics. Use the provided command to read the data into the `soccer` object as a tibble. If you look at the data, you'll see

that, aside from a few ids and the player's team affiliation, there is an `events` column that contains a series of 3 (and only 3 - no gimmicks!) events in typical "soccer statistic" format. Each event is denoted with a letter code (e.g., `I` = *injury* `G` = *goal*, `Y` = *yellow card*, etc.) and then an associated minute marker (e.g., `54'` or `12'`). For example, the events of the first row of data (for player Roger Milla), indicate that Roger was injured in the 54th minute, and (apparently unphased by his probably-fake soccer injury) later scored in the 106th and 108th minutes. Professional soccer players are wimps.

You'll notice, however, that the events are not listed in order of their minute markers. (They are, in fact, listed in random order, courtesy of me.) This means that, if we wanted to separate those events into three more useful columns that represented the first, second, and third event, we can't just use a `separate()` command and split apart the column on that nice space between each event.

We can, however, use some creative reshaping of the data to accomplish that goal. Because this is fairly complex, I'm just going to walk you through it. I'm going to do this without giving you the actual code because I don't want you to just copy and paste. Walk through the logic with me and hopefully you'll learn a few things.

First, let's start by separating the `events` column into three, we'll call them `a`, `b`, and `c`. (Note that, given the underlying order of the data in those three new columns, the column names are just names and don't imply any sort of order.)

Now let's pivot the data to a longer format, pulling those three columns into key-value pairs. You can name the two new columns whatever you'd like. After pivoting, you should now have three rows for each one row from the previous data. (And you should also see a repeating "a", "b", "c" in each of the three rows per player.)

Now we'll use our expert data cleaning skills to extract (or parse) a usable number from the event values and assign it to a new column, named whatever you'd like. This column should contain *only* the minute values and should be a number datatype. Using this new column, we can now sort the events for each player in the order that they actually happened during each match. (You'll have to do an `arrange()` with lots of columns in order to get this to work properly.)

Now that the events are sorted in the order they happened, you'll notice that the "a", "b", and "c" values are all out of order. This is as it should be. We now need a new variable that allows us to preserve this nice sorting when we pivot the data back to the original wide format. To do this, we'll `mutate()` a variable that counts from 1 to 3 repeatedly for each player's three rows. First, we group by the 4 "id-like" columns (`round_id`, `match_id`, `team_initials`, `player_name`), and then we can `mutate()` a new column called `order`, or whatever you'd like to call it, to which we assign the special function `row_number()`. You'll see as you test this that `row_number()` will sequentially count the rows in a dataframe, but when used within a grouped tibble, it will sequentially count the rows for each group of rows. (This is much the same as the way `n()` counts the total number of rows for a tibble but only counts the "group size" when used within a `group_by()`).

Once you get the above to work properly, you should have a nice `order` column that represents the order of each player's 3 events, which we can now use *instead* of the original "a" "b" "c" labels that were out of order. So our final operation is to now pivot the data back out wider, using the new `order` column as the `names_from =` parameter value. (You'll likely need to select down to eliminate the other individual columns we created along the way in order for the `pivot_wider()` to result in a single row per player, per match like we started with.) It also shouldn't matter if you ungroup or not before pivoting wider, but it's usually a good idea to get in the habit of ungrouping after you're done with a group-based task.

One last thing: when you supply numbers as the column names to the `pivot_wider()` function, the columns that are created are not very nice (you'll notice each is displayed wrapped with tick marks). This is because R doesn't like to have numbers at the beginning of any named objects (including column names) because it messes with its syntax interpretation. So the last thing you should do would be to either rename these columns with an "event" prepending each, or, if you want to be cool, figure out how to use the `names_prefix` parameter inside the `pivot_wider()` function to have the function do that for you.

There! Nicely ordered columns that represent the first, second, and third in-game events for each of these players' games. Fun, right? Hopefully you watched what you were doing along the way and learning a few things about how you can use these types of advanced pivoting tricks to get around problems that would otherwise be a nightmare to solve in a row-by-row fashion.

> *Answer (top 3 rows shown):*

```
# A tibble: 83 × 7
   round_id match_id team_initials player_name       event1 event2 event3
      <dbl>    <dbl> <chr>         <chr>             <chr>  <chr>  <chr>
 1      201     1086 ARG           Guillermo STABILE G8'    G17'   G80'
 2      201     1097 USA           Bert PATENAUDE    G10'   G15'   G50'
 3      202     1101 URU           Pedro CEA         G18'   G67'   G72'
```

6. Save the resulting tibble to `soccer_events_fixed`.

Bonus Problem

If you want to give the above idea a go on your own, with a bit more complexity, I've prepared a more difficult version of the problem. You'll receive no credit for accomplishing this one, but you will be the honored recipient of my congratulations.

You'll see the data import at the bottom of the script, reading the data into `soccer_2`. The problem is more difficult because the set of events is not constant for each player. (Meaning that some have 2, others 3, all the way up to 6 events.) This means that the `separate()` doesn't work quite as nicely. Because I just told you that there are up to 6 events, you can technically setup the `separate()` function to handle that, but if you're here doing a bonus problem just to learn, I'd encourage you to assume that the number of events is unknown and that the dataset is too big for you to reasonably figure it out.

In addition, rather than just reordering the events in the order of the game minutes, the end goal (no pun intended) is now to provide a count of each type of event (displayed as a word rather than the one-letter code) for each of the 898 player-match records. I've provided the answer below that you're aiming for.

> *Hints: Here are a few functions to think about exploring: `str_split()`, `unnest()`, `recode()`, as well as the `values_fill` parameter within the ending `pivot_wider()` operation. (I've also included in the script the set of event labels associated with each letter code.)*

> *Answer (top 3 rows shown, and sorry for the poor formatting if you're viewing this as a PDF):*

```
# A tibble: 898 × 9
   round_id match_id team_initials player_name       countGoal countPenalty
countRedCard countYellowCard countInjury
      <dbl>    <dbl> <chr>         <chr>                 <int>        <int>
<int>           <int>       <int>
 1      201     1084 ARG           Guillermo STABILE         2            0
0               0           0
```

```
 2      201     1086 ARG           Adolfo ZUMELZU            2            0
0               0           0
 3      201     1086 ARG           Guillermo STABILE         3            0
0               0           0
```

7. This one was optional, so no need to save it out to a tibble. Hope it was fun!

Final Cleanup and Submission

Final Run

Please do the following to make sure your code runs without errors and to make sure that you have the same objects in memory as the list below:

1. Restart your R session (Session >> Restart R).
2. Run your entire script from the beginning, watching for any errors along the way.
3. After your script is finished, use the following command to see what objects are in memory at the end of your script: `ls() %>% tibble() %>% print(n=30)`. You can compare your output to mine below and make sure that everything is named properly and exists:

```
 1 addr
 2 artist_most_at_1
 3 bad_drivers_long
 4 bad_drivers_wide
 5 peak_week
 6 regions
 7 sales_regions
 8 soccer
 9 soccer_events_fixed
10 songs
11 songs_time_in_100
```

Housekeeping

Once you have completed the steps in the prior section, please check all of the following and make adjustments as needed. (Failure to do the housekeeping steps below will likely cause an error in our grading process, and that will make it hard to give you the right credit for your hard work.)

1. If you used the `setwd()` command near the beginning of the script, please COMMENT OUT that line before committing and pushing your code.
2. Please also comment out any code where you are either using the `View()` or `glimpse()` functions. These cause issues with our grading procedures.
3. If you installed any new packages as you completed the assignment (using `install.packages()`), please comment those installation commands out as well.
4. Lastly, please ensure that you have **NO** absolute references in your code that are not commented out. (An absolute reference is something like: `/Users/YourName/Documents/GitHub/...` or `C:/GitHubProjects/...` .) These also throw errors and make it hard for us to grade your work.

Save, Commit, Push

You're now ready to do all three of the following:

1. Save your script.
2. Do a final commit with your git tool to stage your work for submission.
3. Push your changes up to your repository. And you're done!