Data Cleaning Case Assignment

Welcome to the data cleaning case! This case will walk you through a bunch of REALLY FUN methods for cleaning data and performing logic to clean up and then extract meaning from character strings stored in columns.

As a reminder, you'll start this code assignment (and all code assignments) by cloning the repository to your local machine.

All of the activities for this assignment are to be completed in a separate R script in this repository. I've provided quite some starter code in the `data_cleaning_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 `data_cleaning_case.R` file. This will guide you through reading in the data we'll use for this assignment.

NOTE: For the last assignment, we had you use the `setwd()` command and then comment that command out before submitting your assignment. We also had you create a local `data/` directory and read the assignment data from there. For this assignment (and actually the next several), neither of those will be required. You can just read the file straight from the web using the code provided in the case script. Easy peasy!

If you send the data to the console or `glimpse()` it, you'll notice that the columns contain lots of useful information, but the information is hiding among various strings. (Notice all of the columns' data types as `<chr>`.) In this assignment, you'll walk through a handful of cleaning steps to extract the information from those columns.

Let's start with the easy one. The `sq_meters` column contains the area of each rental. Add a new column called `sq_meters_c` that converts those area numbers to actual numbers. For example, you should convert the `"127_m2"` string to a `127`, with a numeric (i.e., either interger or double) datatype.

> *Partial answer:*

```
1 listingType:apartment |Brasil|São Paulo|São Paulo|Morumbi|            26/11/2015
1918.2 BRL    127_m2    5th floor, 1 room           127
```

1. Save the resulting tibble (360 rows, at least 7 columns in any column order) to `sq_meters_fixed`.

Looks like the `property_type` column has some extra label data included. First, go ahead and make sure that the "listingType:" is the only label embedded in the column. (No need to save that quick peek; it was just a QA check. Remember: **BE DEFENSIVE**!) Let's clean out the "listingType:" labels and add a nice, clean column with the actual property type information. This new column should be saved as `property_type_c`.

> *Hint: The `property_type_c` column should contain 3 separate types: "apartment" (339), "house" (9), and "store" (12). If something is missing, you'll need to add it in as you clean that column.*

> *Partial answer:*

```
 1 listingType:apartment |Brasil|São Paulo|São Paulo|Morumbi|             26/11/2015
1918.2 BRL    127_m2    5th floor, 1 room   apartment
```

2. Save the resulting tibble (360 rows, at least 7 columns in any column order) to `property_type_fixed`.

Have a look at the `listing_price` column. Let's convert that to a numeric value that represents the listing price (in US dollars) for each rental. (Assume that the conversion rate is 1 dollar to 5.2 Brazilian real.) Name that new column `listing_price_usd` and make sure it has a numeric (i.e., either integer or double) datatype.

> *Hint: Depending on how you go about this, you may end up adding a few columns, which is totally fine. The result below should have at least 7 columns (the original 6 plus the new `listing_price_usd` column), but it can have more without affecting your score for this problem.*

> *Partial answer:*

```
1 listingType:apartment |Brasil|São Paulo|São Paulo|Morumbi|             26/11/2015
1918.2 BRL    127_m2    5th floor, 1…      0   1918.    369.
```

3. Save the resulting tibble (360 rows, at least 7 columns in any column order) to `listing_price_fixed`.

The `floor_numRooms` column contains two different pieces of information: the floor (as in, the level of a many-story building like in a large apartment flat) and the number of rooms the rental has (as in, 1 room studio vs. 2 room apartment, etc.). The problem is that *someone* combined them into a single column with some varying formats. Your job is to pull out the floor and room information into `floor_num` and `room_count` columns, respectively, both of which should have a numeric (i.e., integer or double) datatype.

> *Hint: It may be helpful to think of the problem as a two-stage problem, where you first try to isolate each piece of information into its own column, and then convert them to numbers from there. As you go through that process, there are many ways to manipulate the strings, some of them more tedious than others. You can save any columns that you'd like to save along the way, as long as the two new columns are correctly found in the tibble you save below.*

> ***BONUS CHALLENGE**: For fun (and **no** extra credit), see if you can figure out how to end up with exactly 8 columns from just two piped commands (for a total of three lines of code) and no extra "intermediary" columns that have to be dropped later. (And no nested pipes or comma-separated mutates or the like.) It's possible, but you'll probably have to find some functions that we haven't talked about in class. You may want to finish the rest of the assignment and then return to this if you have time. Slack me if you figure it out! :)*

> *Partial answer:*

```
1 listingType:apartment |Brasil|São Paulo|São Paulo|Morumbi|             26/11/2015
1918.2 BRL    127_m2    5th floor, 1 room          5       1
```

4. Save the resulting tibble (360 rows, at least 8 columns in any column order) to `floors_rooms_fixed`.

The `place_with_parent_names` column contains a set of pipe-separated (`|`) places for each listing. The format of the column goes from "general" to "specific" from left to right. On the first row, for example, the places go, from left to right, Brasil (the country), São Paulo (the state), São Paulo (the city), and Morumbi (which is a city district/neighborhood). Let's assume that a real estate investor wants to use this data to understand how rental listings differ by district within a city. Let's extract the district names from the 10 most common districts in the dataset (which are provided for you below...please don't calculate that yourself). Save these names to a new column called `ten_districts_name` and ensure that it *only* contains one of the 10 district names you see in the list below (no pipes like `|` or any other extra characters, and it should be missing [`NA`] if the listing is in a district other than the 10 in the list).

Once you have extracted those district names, add a binary integer column for each of the top three most common districts (in terms of their frequency in the dataset). The columns should be named `common_district_1`, `common_district_2`, and `common_district_3` and should contain a `1` if the listing is located in the corresponding (top-3) neighborhood and a `0` otherwise. (For example, if the overall most common district was the "Panamby" district, you would add a column called `common_district_1` and it would have a value of `1` for any listing that was located in the "Panamby" district, with every other listing containing a value of `0`.) Unlike the `ten_districts_name` column, these three "common_district" columns should *only* contain values of `0` or `1`, which means that there should be no missing [`NA`] values in these three columns.

The ten most common districts in this dataset are (in **alphabetical** order): Barra da Tijuca, Jardim Goiás, Jardim Paulista, Moema, Morumbi, Panamby, Perdizes, Pinheiros, Recreio dos Bandeirantes, Vila Suzana".

> *Hint 1: There are lots of different ways to go about this, some of them very complex and some quite simple. Think of the functions that we learned **in class** and whether any of those will be helpful here. You might be able to find other ways of going about solving this, but the methods we talked about in class will likely be the easiest way to accomplish the majority of task. (And I know I threw a LOT of words at you up there, but that was just to make the task abundandtly clear. It's not as complicated as the count of three paragraphs above might imply.)*

> *Hint 2: You will **definitely** get partial credit here if you at least have the columns added and if they contain the values they should. The really specific instructions about the `NA`s in the `ten_districts_name` column and the lack of `NA`s in the "common_district" columns might be a bit annoying, and you are welcome to just ignore those requirements and move on if you don't want to spend the time to figure them out. You'll still get almost all of the available credit.*

> *Hint 3: Depending on how you go about this, you may end up adding a few columns, which is totally fine. The result below should have at least 10 columns (the original 6 plus the four new ones: `ten_districts_name`, `common_district_1`, `common_district_2`, and `common_district_3`), but it can have more without affecting your score for this problem.*

> *Partial answer:*

```
1 listingType:apartment |Brasil|São Paulo|São Paulo|Morumbi|             26/11/2015
1918.2 BRL   127_m2  5th fl… Morumbi       0       0       1
```

5. Save the resulting tibble (360 rows, at least 10 columns in any column order) to `districts_extracted`.

Now let's fix that date column. Use the `lubridate` package to convert the `date_listed` column from a character string to an actual date column called `date_listed_c`.

> *Hint: The easiest way to do this is with one of the simple parser functions (e.g., `ymd()`, `mdy_hms()`, etc.) that we went over in class. The most obvious one given the date string format WILL give you **exactly** 5 parsing errors. You will get **almost** full credit on this problem if you want to just move on and ignore the errors. If you want to parse all dates correctly, you'll need to figure out a way to either clean up the problematic date formats before parsing (by either truncating the offending portion or extracting the month, day, year portion), or alternatively you can learn how to use the more flexible `parse_date_time()` function, which allows you to specify multiple date formats for the parser to try. Depending on which of those methods you end up using, you may end up adding some additional columns along the way, and you don't need to worry about removing them. Just make sure your final tibble that you save below has a column called `date_listed_c`.*

> *Partial answer:*

```
1 listingType:apartment |Brasil|São Paulo|São Paulo|Morumbi|            26/11/2015
1918.2 BRL    127_m2    5th floor, 1 ro… 2015-11-26 [and this date might also have
00:00:00]
```

6. Save the resulting tibble (360 rows, at least 7 columns in any column order) to `date_listed_converted`.

Now let's use that tibble you just created (`date_listed_converted`) to do one more data quality check. Use whatever means necessary to figure out the *range* of the `date_listed_c` column. (Don't worry - even if you didn't end up fixing the parsing issues in the previous problem, your answer to this problem won't be affected.) Does the range look right to you, given that this data represents *historical* listings? If anything looks off (like, for example, by about 100 years), then let's see if we can fix it by assuming that the rental listing actually happened during this century but someone (like your mean professor) made a data entry error. Add a new column called `date_listed_fixed` with corrections to any affected dates found in the `date_listed_c` column.

> *Hint 1: We talked in class about how dates that are truly formatted as dates are more useful in comparison and analysis (as compared to strings). This means that the `date_listed_c` column can be sorted, compared, or used with calculations like `min()` or `max()`. Identifying the issue is the easy part, however. Figuring out how to fix it will take some sleuthing. I would suggest looking into the `lubridate` package's documentation (or check the Cheat Sheet on the course website!) to see if you can find a way to subtract a certain number of a measure of time (e.g., month, hour, day) from another date.*

> *Hint 2: While learning more about the `lubridate` package might appeal to some of you, there are likely others of you who are tired of my antics. I don't blame you. If that's the case, you can get **almost** all the points for this problem if you instead do the following: Start with the `rentals_raw` tibble for this problem rather than the `date_listed_converted` tibble as instructed. If you take this route, you'll go about fixing those small handful of incorrect dates by doing a simpler `str_replace()` operation on the original `date_listed` column with some fairly straightforward regex logic. You can then use the same process you used in the previous problem to convert the string to a date datatype and verify that the weird future dates have indeed been taken care of, which you can then save to the same `date_listed_fixed` column as instructed above.*

> *Partial answer: (I've filtered the partial answer below to one of the affected dates so that you can check your work, but your answer tibble should be saved with all 360 rows.)*

```
1 listingType:apartment |Brasil|São Paulo|São Paulo|Morumbi|     09/12/2115  5047.96
BRL 290_m2  12th flo… 2115-12-09    1 2015-12-09 [again, you may not have these empty
time values: 00:00:00]
```

7. Save the resulting tibble (360 rows, at least 8 columns [if you're aiming for full credit following "Hint 1" above] or at least 7 columns [if you've opted for *almost* full credit following "Hint 2" above]) to `date_listed_converted2` .

Finalize, Commit, Push

BEFORE YOU COMMIT AND PUSH YOUR SUBMISSION

Please make sure you to do the following before you submit this assignment:

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. The list of objects below is what I have in memory after writing this case assignment. Any of those objects are fair game for the grading procedure to evaluate. You may want to clear your R session (Session -> Restart R) and then check the list against what you have in memory after running your script from the beginning.

Hint: you can see what's in your environment by executing the following code:

```
ls() %>% tibble() %>% print(n=30)
   1  date_listed_converted
   2  date_listed_converted2
   3  districts_extracted
   4  floors_rooms_fixed
   5  listing_price_fixed
   6  property_type_fixed
   7  rentals_raw
   8  sq_meters_fixed
```