Group GitHub Collaboration Assignment

Welcome to your first group assignment! This assignment will get you familiar with how to use GitHub to collaborate as a group on a single script. Some will have more experience with this than others, so this simple assignment is designed to just get everyone on the same page.

Assignment Instructions

The assignment is divided into 4 sections, with each section completed by a different group member. You can do this as a group in one sitting or, if you prefer, you can do it remotely over Zoom or even asynchronously over Slack. But each section should be completed and committed to GitHub by a different person. (And if your group has just 3 people, one person will need to do two sections.)

Just like the rest of the assignments, you'll make all of your edits in the included `github_collaboration.R` script and push your changes to GitHub as your submission. The key difference, as you'll see, is that you'll be creating a branch and then take turns accomplishing each section. After each person takes their turn completing their section, they will commit their changes, push those changes to GitHub, and then the next person can pull those changes locally and complete their portion. The final task, after the last code changes are committed and pushed within the branch, is for that last person to open a pull request, which is reviewed and approved by at least two people, and then the branch will be merged into the main branch. That final "merge" operation will constitute your group's submission of the assignment.

Make sense? Hopefully. Let's do it.

Part 1: Create a Branch, Read and Clean the Data

The first person will create a new branch, based on the `main` branch. This branch should be called `initial-data-prep`, and can be created either on GitHub.com or locally via the GitHub desktop application. (There are about 5 million different tutorials out there, in either written or video form, that will show you how to create a branch if you need some help there.) I have also created a little walkthrough screencast for the branch creation and the pull request process (which will be accomplished at the end). You can watch that short video here.

Within this new branch, your task is to ensure that the data being read in appropriately as a tibble. The data should be read into the `data_raw` object as shown in the starter code.

You should then use the `janitor::clean_names()` function **without loading the library** to ensure that the rest of your team members have reliable, consistent column names that they can access for their downstream cleaning functions. The result of this cleaning operation will be saved to a new object called `data_clean_names`. It should look like the data summarized below:

> *Hint: Don't use `library(janitor)` because that will load the library unecessarily. If you just need one function from a library, it's good practice to just call the function with its "fully qualified" name, which is the format I provided above with the `::` after the package name.*

```
# A tibble: 21,987 × 21
   borough neighborhood  buildi…¹ tax_c…² block   lot easem…³ build…⁴
     <dbl> <chr>         <chr>    <chr>   <dbl> <dbl> <lgl>   <chr>
 1       1 ALPHABET CITY categor… 1         374    46 NA      A4
```

```
 2       1 ALPHABET CITY categor… 1          377     1 NA      S2
 3       1 ALPHABET CITY categor… 1          377    70 NA      B1
```

Once you have completed your edits, you should save the script (not the data itself) and commit your changes with an applicable commit message. All of your changes should be committed and pushed to the same `initial-data-prep` branch that you created at the beginning of this section.

Part 2: Checkout the Branch, Add and Clean some Columns

The next team member will work from the same `initial-data-prep` branch that was created in the previous step. You'll "check out" or clone the branch, pull those changes to your local disk, then operate on the same script to add a few more cleaning steps. Note that your script will never actually write any data out to the disk. Each time it runs, it should pull the raw data from the source URL and then apply the cleaning operations.

Your job will be to clean up a few of the columns so that the data is ready for the next person to use. There are three things you'll need to do:

1. Convert the `sale_date` column to a usable date format. You should probably use the `mdy()` function from the lubridate package. And since you will be doing several date operations (see the next item), you'll probably want to just load that whole package with a `library(lubridate)`. Your conversion should happen *in place*, meaning that you won't be adding a new column but instead you'll save the result of the date conversion back into the same `sale_date` column.
2. You should now derive three new columns from that nicely formatted `sale_date` column, saving each to the tibble so they can be used downstream: `year`, `quarter`, and `month`. (All of these can be accomplished using simple, obviously named functions in the lubridate package.)
3. Have a look at the `building_class_cat_num` column. You'll see that there is some text preceding the building category numbers that should be cleaned out before you send the data on to the next person. Apply some masterful string cleaning operation(s) to remove the stray text and leave just the "number" portion of the contents of that column. (Please **do not** convert the column to a number. The joining operation in the next step assumes that the colum is formated as a character column.)

The result of these cleaning operations should be saved to a new object called `data_clean_2`. It should look like the data summarized below:

```
# A tibble: 21,987 × 24
   borough neighborhood  buildi…¹ tax_c…² block   lot easem…³ build…⁴
     <dbl> <chr>         <chr>    <chr>   <dbl> <dbl> <lgl>   <chr>
 1       1 ALPHABET CITY 01       1         374    46 NA      A4
 2       1 ALPHABET CITY 02       1         377     1 NA      S2
 3       1 ALPHABET CITY 02       1         377    70 NA      B1
```

Once you have completed your edits, you should save the script (not the data itself) and commit your changes with an applicable commit message. All of your changes should be committed and pushed to the same `initial-data-prep` branch that you cloned at the beginning of this section.

Part 3: Checkout the Branch, Add some Columns

The next team member will work from the same `initial-data-prep` branch that was used in the previous step. You'll "check out" or clone the branch, pull those changes to your local disk, then operate on the same

script to add a few more operations. Note that your script will never actually write any data out to the disk. Each time it runs, it should pull the raw data from the source URL and then apply the cleaning operations.

There is already a `read_csv()` command in your section that will read in some data to the `categories_raw` object. After reading the data in, you should then use the `janitor::clean_names()` function **without loading the library** to make the data easier to work with. Save the result to a new tibble called `categories_clean`.

Next, you should join the `categories_clean` tibble to the `data_clean_2` object that was prepared for you in the previous step. You'll need to specify a join key, but other than that the join should be pretty smooth (i.e., no trickery here!). After joining, you should sort the data by the `sale_date` column (in ascending order).

The result of these cleaning operations should be saved to a new object called `clean_with_categories`. It should look like the data summarized below:

```
# A tibble: 21,987 × 25
   borough neighborhood  buildi…¹ tax_c…² block   lot easem…³ build…⁴
     <dbl> <chr>         <chr>    <chr>   <dbl> <dbl> <lgl>   <chr>
 1       1 ALPHABET CITY 13       2         406  1311 NA      R4
 2       1 ALPHABET CITY 13       2         406  1317 NA      R4
 3       1 CHELSEA       04       1C        769  1006 NA      R6
```

Once you have completed your edits, you should save the script (not the data itself) and commit your changes with an applicable commit message. All of your changes should be committed and pushed to the same `initial-data-prep` branch that you cloned at the beginning of this section.

Part 4: Finish Data Prep, Commit, Open Pull Request

The final team member will work from the same `initial-data-prep` branch that was used in the previous steps. You'll "check out" or clone the branch, pull those changes to your local disk, then operate on the same script to add a few more operations. Note that your script will never actually write any data out to the disk. Each time it runs, it should pull the raw data from the source URL and then apply the cleaning operations.

The final code addition task for person 4 is to provide a quarterly summary of sales for each neighorhood. Specifically, for each neighborhood, show the `count_of_sales` and `mean_sales_price` for each year and quarter number. (Use the column names in the previous sentence.)

> *Hint: The next step will ask you to pivot the data to a longer format, but I'll show you what the pre-pivoted data looks like below. Note that this pre-pivoted data* ***does not need to be saved****; we will only grade the output after it's been pivoted.*

```
# A tibble: 195 × 5
# Groups:   neighborhood, year [78]
   neighborhood   year quarter count_of_sales mean_sale_price
   <chr>         <dbl>   <int>          <int>           <dbl>
 1 ALPHABET CITY  2021       4             23        2832087.
 2 ALPHABET CITY  2022       1             40        1306569.
 3 ALPHABET CITY  2022       2             75        1292543.
```

After these data have been summarized, pivot the data longer such that the `count_of_sales` and `mean_sale_price` columns are pulled down into long format. Use the column names `measure` and `value` to house those column names and values, respectively.

The result of this quarterly summary and pivoting should be saved to a new object called `quarterly_summary`. It should look like the data summarized below:

```
# A tibble: 390 × 5
# Groups:   neighborhood, year [78]
   neighborhood   year quarter measure            value
   <chr>         <dbl>   <int> <chr>              <dbl>
 1 ALPHABET CITY  2021       4 count_of_sales       23
 2 ALPHABET CITY  2021       4 mean_sale_price 2832087.
 3 ALPHABET CITY  2022       1 count_of_sales       40
 4 ALPHABET CITY  2022       1 mean_sale_price 1306569.
```

Once you have completed your edits, you should save the script (not the data itself) and commit your changes with an applicable commit message. All of your changes should be committed and pushed to the same `initial-data-prep` branch that you cloned at the beginning of this section.

After committing your changes and pushing them, your final task is to open a "pull request" on GitHub. This pull request is the standard mechanism used by one or more people to have others on the team review and approve the work they did within a branch before that work is then merged into the `main` branch. This is a great way to catch errors or otherwise just make sure that the code is readable and understandable to everyone on the team.

Once you have pushed your changes to GitHub, you can open a pull request. This is usually accomplished right on GitHub's website. You'll create a pull request that compares the branch you've all been working in (`initial-data-prep`) to the `main` branch. The process for creating a pull request was summarized in the second half of the little screencast I made, accessible here. You can also find more info on what pull requests are or how to create one using GitHub's documentation.

After creating your pull request, let your team members know so they can review the code before you merge the changes into the `main` branch.

Part 5: Approve Pull Request (x2) and Merge

Once the final team member has finished their code edits, they will create a pull request and request some reviews from the others on the team. At least two of you can perform a review of the pull request and, if everything looks good, provide an "approval". You should be carefully double-checking your code for errors, making sure that all added columns are correct and that the objects saved within the script are named correctly, etc.

Once your team has provided two approvals, you can go ahead and merge the pull request. (You can delete the branch or keep it after the merge, it won't matter much.)

It might be a good idea for one team member to take care of the final cleanup and housekeeping items after you complete the merge, just to make sure your team's submission will be gradable without errors. This can be accomplished (if you choose) by pulling the merged changes (in the `main` branch), running the code from an empty environment as described below, and just verifying that everything looks good.

Either way, after you've merged your changes, the assignment has already been submitted because that main branch will be what we grade, and the pull request has merged the code into the main branch on

GitHub.

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 categories_clean
2 categories_raw
3 clean_with_categories
4 data_clean
5 data_clean_2
6 data_raw
7 quarterly_summary
```

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!