All your data this week is dealing with text, particularly that of Aesop's Fables, as published here. Initially it will all import as a single block of text, so you will have some parsing and cleaning to do before you can use it for analysis, but that will be described in the following questions. The data in question resides in the file `fables.txt`, which has already been prepped so that it can be read in as a single column, single row CSV file.

Follow the link here to accept the assignment and get access to the repository:

Assignment link: https://classroom.github.com/a/Is_8ihvG

1. (4 points) You initially need to get the data into Postgres. Create a table named `fables_raw` which has just a single column named `contents` and which has the `TEXT` type, then import the contents of `fables.txt` into that file. (*Despite being a .txt file, the file is prepped with quotes to be read as a headerless .csv file, so choose your import format appropriately.*) If you look at the contents of `fables.txt`, you'll notice that it consists of many fables, each of which has a title and body, and some of which have a little moral at the end. Eventually, we are going to want a table with each fable on its own row and with title, story, and moral as the columns. But initially, let's just focus on getting a table of one column with *all* the text corresponding to each fable on each row. Looking at the contents of `fable.txt` might give you some ideas of how you could split all the fables apart onto their own rows. At the end of this problem, you should have a table (call it `fables_split`) which has a single column and 284 rows of text. In addition to including the queries you issue to make this happen, export a headerless copy of this table as a CSV named `fables_split.csv` and make sure to upload it back to your repository as part of your submission.

2. (8 points) Now for some trickier splitting. From each row (and thus each fable) you want to extract the title, story, and moral from that fable, saving them to a new table called just `fables`. One aspect that complicates this is the fact that some fables do not have a moral, and thus the pattern matching can be tricky to match both fables with morals and fables without at the same time. One approach might be to just match one (or the other) initially, filling out all those rows in the table, and then going back later to fill in the other rows with the necessary information (or tweaking them to contain *just* the desired information). Fables without a moral should have `NULL` values in the `moral` column, and fables *with* a moral should *only* have that moral in the `moral` column (it should not still also appear in the `story` column). At the end of this part, you should have a table called `fables` with three columns: `title`, `story` and `moral`, and which still has 284 rows of text. Every fable will have both a title and a story, but not all will have a moral. Again, in addition to including the queries you issue to make this happen, export a copy of this table as a headerless CSV named `fables.csv` and make sure to upload it back to your repository as part of your submission.

3. (4 points) Now to actually do some analysis! What percentage of these Aesop's Fables have a moral attached to them?

4. For this last question, we are going to do some full text searching.

   (a) (4 points) Add a new column to `fables` called `story_vec` which will be a `tsvector` of the story contents. Update `fables` to populate this column, add a `GIN` index to it, and then use it to answer the following two parts.

   (b) (4 points) What are the titles of the Aesop's Fables that have the word "sea" as part of their story, but *not* the words "sailor" or "perform"?

   (c) (4 points) Included in the repository is a file called `make_animals.sql` which when run will create a very simple table called `animals` with a single column of animal names. Use this table and your existing `fables` table to determine how many fables have each animal making an appearance in the story. Order the table by decreasing counts, so that the most common animals appear at the top. As a sanity check, a crow should show up in 12 different fables. (*The query to achieve this really is not complicated, so if you find yourself doing anything super complicated, perhaps think of a simple application of a subquery that could help.*) Upload a CSV of this table (with header!) called `animal_counts.csv` back to GitHub!