Handin 6

Due 25 Oct by 18:00 **Points** 100 **Submitting** an external tool
Available after 11 Oct at 18:00

This week will focus on the pandas external library.

Part 1

1. Create a file called `handin6.py`. Inside this file, write a function called `find_palindromes` that takes two arguments: 1) `word_dataframe`, which should be a pandas dataframe with a single column called `"word"`, and 2) `minimum_length`, which is an integer specifying the minimum length of the matching words (this should default to 4). The function should return a new dataframe, containing only those words that 1) are palindromes (i.e. spelled the same in the forward and reverse direction - e.g. "madam"), and 2) should be at least `minimum_length` characters long, 3) should not contain any apostrophes. Note that the function should be case insensitive - i.e. it should not differentiate between lower and upper case letters (e.g. "Otto" is a palindrome). The returned dataframe should just keep the indices from the original dataframe (no need to reset the index).
   Hint: one way to solve this exercise is to create boolean masks for each of the requirements - if you have two boolean Series (mask1 and mask2), you can create a combined mask matching all cases where both mask1 and mask2 are true (element-wise AND operation), by writing (`mask1 & mask2`). Also, remember that you can access string functionality in a pandas array through the .`str` accessor (see the slides for details).

   Create a file called `handin6_test.py`. Inside this file, read the `british-english` file into a pandas dataframe called `df_british` as we have done in class (setting `keep_default_na=False` and making sure that pandas doesn't read the first line as a header). Remember to change the columns name to what is expected by `find_palindromes`. Now call the `find_palindromes` function on this dataframe. Save the result in a variable called `palindromes`.

2. Inside `handin6.py`, write a function called `find_words_starting_with`. This function takes two arguments: 1) `word_dataframe` (similar to before), and 2) `prefix`, a string specifying what the first characters of the words should be. The function should find all words in `word_dataframe` that start with `prefix` (case-insensitive), and do not contain apostrophes. It should return these matches, but grouped by the length of the words. The return values should be a dictionary, where the key specifies the length of the words, and the values are lists of all matching words of that length. For instance, if I search for "DATA", I should get back `{4:`

`['data'], 8: ['database', 'datatype'], 9: ['databases']}`. Hint: remember that you can access string functionality in a pandas array through the `.str` accessor (see the slides for details).

In `handin6_test.py`, call the `find_words_starting_with` function on the `df_british` dataframe, using the search word "congra". Save the result in a variable called `matching_words`.

Part 2: Project

Now that we've learned pandas, we want to use it to process our `Land_and_Ocean_summary.txt` file. To simulate a typical Data Science workflow, we will do this exercise in a jupyter notebook. You should therefore start by creating a notebook called `handin6_project.ipynb` (see the Monday's slides for different ways to set this up).

1. The difference between pandas and numpy is that pandas associates labels with rows and columns. We thus need to extract this information from the file. However, when looking at the file, we see that this is not trivial to do: there are a bunch of comment lines at the top of the file, and only the last ones of these is the header information. We could manually count how many comment lines there are, and then use the `skiprows` argument to `read_table` to skip that number of lines, but that is not a very robust solution (it would break if someone added an extra line of comments to the file). Instead, we will write a small function called `extract_header` that will extract the header in a more robust way. The function should take four arguments, 1) `filename`, 2) `comment_prefix`, 3) `header_row_index`, 4) `header_row_delimiter`. The function should open `filename`, iterate over the lines, find all lines starting with `comment_prefix` and save them in a list. From this list of comment lines, it should then extract the single line corresponding to `header_row_index` (e.g., if I choose `header_row_index=-1` it should give me the last of the comment lines) . It should then 1) remove the `comment_prefix` character from the line, 2) split the line based on `header_row_delimiter`, 3) remove any white space from these header labels, and finally return the header as a list of strings. The function should be written inside a cell in your jupyter notebook.

   In a new, subsequent cell, call your function using the following code:

   ```
   extract_header('Land_and_Ocean_summary.txt', comment_prefix='%', header_row_index=-1, header_row_de
   limiter=',')
   ```

   and save the result in a variable called `header`.

2. Now, we are ready to read in the data. In a new cell, create a function called `read_anomaly_data_into_dataframe`, which takes four arguments: 1) `filename`, 2) `header` (list of strings), 3) `max_cols`, and 4) `comment_prefix`. The function should use pandas' `read_table` to

read in the data, using the provided `header`, but reading only the left-most `max_cols` columns and ignoring lines starting with the character specified by `comment_prefix`. The function should return a dataframe, where the first column (i.e. the year) is the index. Hint: use the `delimiter=r"\s+"` option for `read_table`.

In a new, subsequent cell, call the function as `read_anomaly_data_into_dataframe('Land_and_Ocean_summary.txt', header, max_cols=5, comment_prefix='%')` and save the result in a variable called `anomaly_df`.

3. Now, let's use pandas functionality to do some calculations. Create a new cell, and write a function called `anomaly_avg_per_decade` that takes a single argument: 1) `anomaly_df`, which is a dataframe like the one we produced in the previous question. This function should group the rows in the dataframe by decade such that `1970`, `1971`, ..., `1979` belong to the group `1970`, and `1980`, `1981`, ..., `1989` belong to the group `1980`, etc. For each of these groups it should then calculate the average anomaly for the `Annual Anomaly` column. The return value should be a pandas Series containing these values.

   In a new, subsequent cell, call the function on the `anomaly_df` obtained in the previous exercise. Save the result in a variable called `anomalies_per_decade`.

4. Finally, let's plot the data within our notebook. Create a new cell where you plot the `anomalies_per_decade` dataframe as an inline matplotlib plot. Hint: remember that dataframes have associated plotting functionality.

This tool needs to be loaded in a new browser window

Load Handin 6 in a new window