Exercise 5

In this exercise you are allowed to import NumPy, SciPy, Matplotlib and Pandas.

Part A – getting familiar with pandas:

1. Write a function named *filter_numeric_column.*
   The function receives a *DataFrame* object, column name and two numeric parameters.
   The function returns a *DataFrame* which contains only rows in which the value in the given column name is between the numeric values given (including the lower value and not including the higher value).
   df = pd.DataFrame(data = {'col1':[1, 2, 3, 4, 5], 'col2': [3, 4, 5, 6, 7]})
   df_filtered = *filter_numeric_column*(df, 'col1', 2, 4)
   print(df_filtered)

```
Out[6]:
   col1  col2
1     2     4
2     3     5
```

2. Write a function named *switch_col_names*.
   The function receives a *DataFrame* and two strings that are names of two columns in the dataframe.
   The function returns a *DataFrame* with the names of the two columns switched.
   df = pd.DataFrame(data = {'col1':[1, 2, 3, 4, 5], 'col2': [3, 4, 5, 6, 7]})
   switch_col_names(df, 'col1', 'col2')

```
   col1  col2
0     3     1
1     4     2
2     5     3
3     6     4
4     7     5
```

3. Write a function named *double_special_letters.*
   The function receives a *DataFrame* and 3 columns named: *col_name_letter, col_name_value, col_name_results*.
   The function creates a new column named *col_name_results* that contains similar values to the values in the *col_name_value* column. The only difference is that in the new column the values are doubled if the letter in the *col_name_letter* is one of the "huji" letters (h\u\j\i, only lowercase).
   df = pd.DataFrame(data = {'letter':['h', 'a', 'b', 'c', 'i'], 'value': [3, 4, 5, 6, 7]})
   double_special_letters(df, 'letter','value','col_res')

```
  letter  value  col_res
0      h      3        6
1      a      4        4
2      b      5        5
3      c      6        6
4      i      7       14
```

4. Write a function named *put_row_items_together*.
   The function receives a *DataFrame df* and a name of a new column named *col_name_results*.
   The function creates a new column (named *col_name_results)*. Each value in this column contains a list of all the items in the row.

   ```
   df = pd.DataFrame(data = {'col1':[1, 2, 3, 4, 5], 'col2': [3, 4, 5, 6, 7]})
   put_row_items_together(df, 'res1')
   ```

	col1	col2	res1
0	1	3	[1, 3]
1	2	4	[2, 4]
2	3	5	[3, 5]
3	4	6	[4, 6]
4	5	7	[5, 7]

Part B – Getting to know the TV show "Friends"

In this part you will use *pandas* to answer certain questions regarding the tv show "Friends".

Data regarding the show was uploaded to the model and together we going to analyze it.

You should return *DataFrame* exactly as defined in the exercise – same number of columns and same column names.

1. Understanding our data:
   Download the RAR file and unzip it.
   In the RAR file you will find 12 CSV files. The files contain our data.
   Friends_imdb_episode_season_X.csv – contains 3 columns: "tconst" - a unique IMDB identifier for each episode, "seasonNumber" – the season number and "episodeNumber" – the episode number.
   rating_all_imdb.csv – contains all ratings in the IMDB website. This file contains 3 columns: "tconst" - a unique identifier for each episode (same as in the episode files), "averageRating" – the average rating score for the specific item. "Number of votes" – the number of people rating this item.
   friends_quotes.csv – contains 6 columns: "author" – the character who said the quote, "episode_number" – the episode number, "episode_title" – the title of the episode, "quote" – the lines said by the characters, "quote_order" – an index representing the order of the quotes within each episode and "season" – season of the episode.

2. Write a function named *create_season_rating* that identifies which is the best Friend's season.
   The function receives the folder path containing the CSV files.
   The function returns a *DataFrame* which contains three columns in addition to the index's column: *seasonNumber, averageRating and numVotes*.

   The function should contain the following stages:
   a. Create a *DataFrame* which contains IMDB identifiers for all episodes.
   b. Merge the new *DataFrame* with the data from the rating dataset (rating_all_imdb.csv).
   c. Now that you have all the data you need in one *DataFrame*, create another *DataFrame* which contains the number of voters and the average rating across all episodes (for each season).

   Add the *DataFrame* you have created to the answer file.

3. Write a function named *most_lines_in_the_second_most_popular_episode* that identifies who is the character with the most lines in the second most popular episode, based on the imdb number of votes (the episode with the larger number of votes).
   The function receives the folder path containing CSV files.
   The function returns a *DataFrame* which contains 4 columns in addition to the index's column: *character name, number of lines, season number* and *episode number*. The *DataFrame* should include only one row per character and the *number of lines* column should include the number of quotes (quote= a row in the friends_quotes.csv file).

   Add the *DataFrame* you have created to the answer file.

Bonus Questions. You may choose only one of following questions (4 and 5):

4. (4 points) Write a function named *who_said_*it that computes how many times a specific character said a certain phrase.
   The function receives the folder path containing the CSV files, a name of a character (string) and a phrase (string).
   The function returns the number of times the character said this phrase.
   Make sure you convert all strings to lowercase letters before searching for the character name and phrase.
   Run you function on 3 examples and add the results to the answer file.

5. (6 points) Think of an additional question that can be answered based on the CSV files presented in this exercise. Answer your question and plot a supporting graph for your results. Add your question, answer, graph(s) and relevant code to the answer file.

* Question 4 is an example for a question that could appear in the test.