Abstract

In recent times, Natural Language Processing as a field has grown tremendously in a variety of fields including but not limited to chat bots, autocorrect, autocomplete, and language translation. This project aims to demonstrate the capability of basic text analytics on two toy datasets to deliver unique insights. The data was extracted from two separate kaggle datasets mined from the movie adaptations of *Harry Potter* and *Lord of The Rings* respectively*.*

The *Harry Potter* dataset contained 7 csv files of size ranging from 5KB to 64KB. The *Lord of The Rings* dataset contained just 2 csv files with the core file being of size 229KB. In order to maintain conformity, only 'Harry Potter x.csv' files were utilized where x is a whole number between 1 and 3 inclusive representing the first three *Harry Potter* movies. From the *Lord of The Rings* dataset, only 'lotr_scripts.csv' was utilized in the analysis.

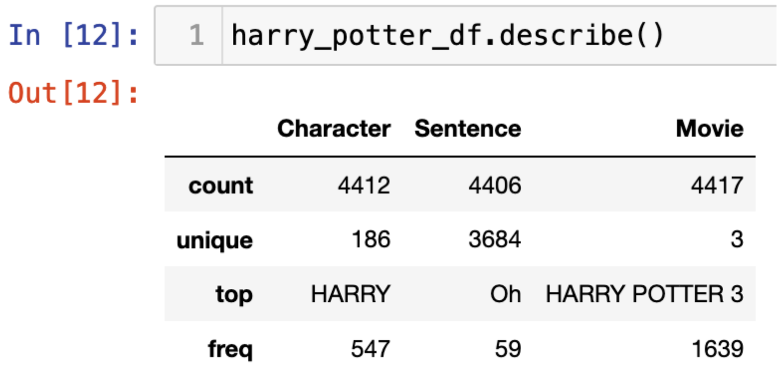
The final code submission contains two .ipynb files for the respective movie series. In each file, The project was split into multiple phases: Data Wrangling, creating word frequency dictionaries, and Data Analysis/visualization. Ultimately, for each of the top 10 characters in the respective series, a histogram was generated containing the frequency distribution of their top 5 words.

Table of Contents

I.Data Cleaning

Harry Potter

Let us begin with the *Harry Potter* dataset. The creators deliberately partitioned each of the three movies into separate csv files with the expectation that separate analysis be performed on each movie or a side-by-side comparison. However, for the scope of this project, the three movie's data frames were concatenated into one dataframe. **Figure 1a** shows the result of the pandas dataframe describe() command result



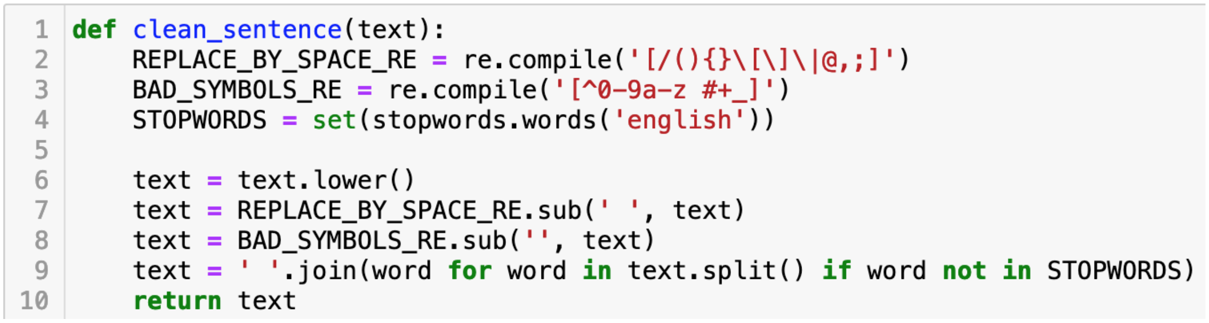
```
In [12]: harry_potter_df.describe()
Out[12]:
```

	Character	Sentence	Movie
count	4412	4406	4417
unique	186	3684	3
top	HARRY	Oh	HARRY POTTER 3
freq	547	59	1639

Figure 1a. Describe() command result for Harry Potter

Based on the output, Harry Potter has the most lines and the third Harry Potter movie has the most character dialog. Other than that, these results do not provide much useful information. In order to set the stage for more advanced analytics one needs to decrease the number of characters. As such, we introduce the first utility function used to accomplish this task.

The function *get_char_stats()* is a simple utility function that displays all the unique values in the 'Characters' column. We notice several obvious issues right in the beginning and correct them by simply using the strip() command to remove all space/hidden characters to the left and right of a string and then using upper() to standardize the text. Applying the function lowers the number of classes from 187 to 109. Next, applying *fix_spelling()* changes typos, lowering the number of classes by 1. Now that we have lowered the number of classes, it is time to clean the character dialog. The *clean_sentence()* method illustrated in **Figure 2** first converts each sentence to lower-case for the sake of standardization. Next it converts various types of punctuation to a space and all remaining bad symbols to the empty string. Finally, using the set of stopwords from the nltk Python library, the function first tokenizes and then joins the words in a line of character dialog that are not stop words.



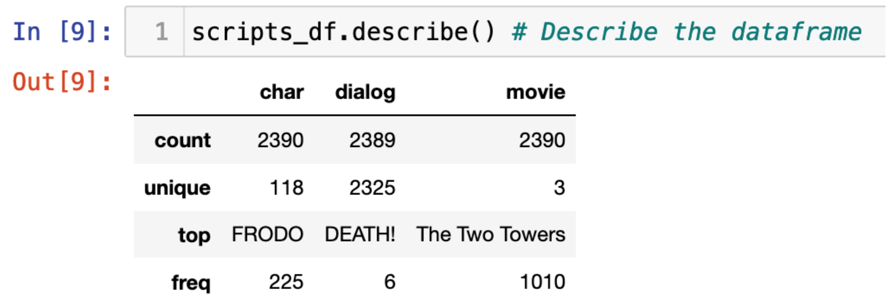
```python
def clean_sentence(text):
    REPLACE_BY_SPACE_RE = re.compile('[/(){}\[\]\|@,;]')
    BAD_SYMBOLS_RE = re.compile('[^0-9a-z #+_]')
    STOPWORDS = set(stopwords.words('english'))

    text = text.lower()
    text = REPLACE_BY_SPACE_RE.sub(' ', text)
    text = BAD_SYMBOLS_RE.sub('', text)
    text = ' '.join(word for word in text.split() if word not in STOPWORDS)
    return text
```

Figure 2. clean_sentence() function for cleaning character dialog also called clean_dialog() in Lord of The Rings analysis

Lord of the Rings

Unlike in the *Harry Potter* dataset, the *Lord of the Rings* dataset contains data from all three movies in one unified csv file. Thus, there is no merge step necessary like with *Harry Potter.*



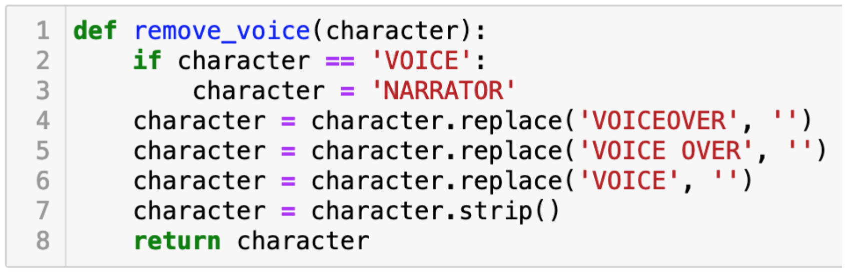
```
In [9]:  1 scripts_df.describe() # Describe the dataframe
Out[9]:
```

	char	dialog	movie
count	2390	2389	2390
unique	118	2325	3
top	FRODO	DEATH!	The Two Towers
freq	225	6	1010

Figure 1b. Describe() output for Lord of the Rings

Figure 1b shows that FRODO has the most lines and that the second movie in the trilogy, The Two Towers contains the most speaking lines for characters. Other than that, there are few useful statistics to be gleaned.

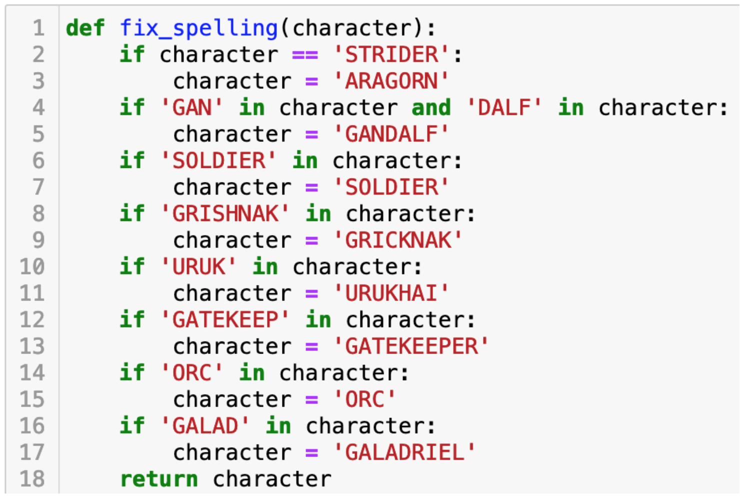
Given that this notebook was created before the *Harry Potter* analysis notebook, the code is more prototype as opposed to modularized. However, the data cleaning done is very similar. Looking at the character column, we see 118 unique character classes. Uppercasing all the characters removes one class. Replacing brackets and punctuation removes an additional 3 classes. Applying the python strip() function puts the class count at 109. A closer scrutiny shows that the *Lord of the Rings* movies contain a lot of voice over lines meaning that the characters speak but are not present on the screen. It was a conscious choice to unify these character lines with those of characters present on screen. **Figure 3** shows the nuances of the *remove_voice()* function which successfully lowers the character count from 109 to 96.



```python
def remove_voice(character):
    if character == 'VOICE':
        character = 'NARRATOR'
    character = character.replace('VOICEOVER', '')
    character = character.replace('VOICE OVER', '')
    character = character.replace('VOICE', '')
    character = character.strip()
    return character
```

Figure 3. Code snippet of remove_voice() function used for preprocessing

While the *fix_spelling()* function for the *Harry Potter* analysis was simple, the one for *Lord of The Rings* is both more technical and requires some knowledge of the movies. **Figure 4** shows for example that Strider is a nickname for the character that we find out later is Aragorn, the future King of men and one of the main heroes in *Return of the King.* Even though the movie script distinguishes between the two, a conscious decision was made to merge the lines of the two. Next, several characters are spelled with a space delimiter and without. Other than through a manual inspection, it is impossible to create a generic method to deal with all cases. Applying the *fix_spelling()* method lowers the character class count from 96 to 84.



```python
def fix_spelling(character):
    if character == 'STRIDER':
        character = 'ARAGORN'
    if 'GAN' in character and 'DALF' in character:
        character = 'GANDALF'
    if 'SOLDIER' in character:
        character = 'SOLDIER'
    if 'GRISHNAK' in character:
        character = 'GRICKNAK'
    if 'URUK' in character:
        character = 'URUKHAI'
    if 'GATEKEEP' in character:
        character = 'GATEKEEPER'
    if 'ORC' in character:
        character = 'ORC'
    if 'GALAD' in character:
        character = 'GALADRIEL'
    return character
```

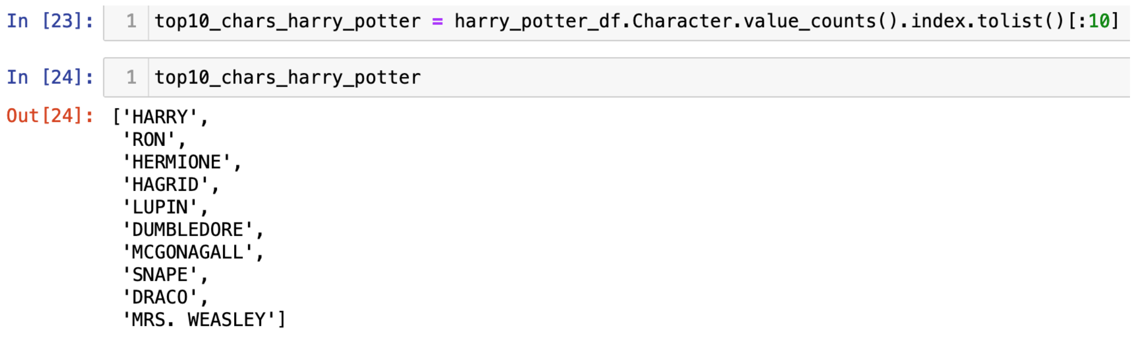
Figure 4. fix_spelling() function in Lord of the Rings Analysis

Finally, the *clean_dialog()* function is used to process the text in the exact same manner as in the *Harry Potter clean_sentence()* function. This marks the next

II. Building Character Vocabulary

Harry Potter

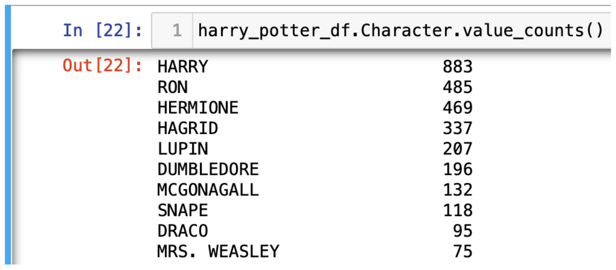
Figure 5a illustrates the powerful code snippet used to extract the top 10 characters from the first three *Harry Potter* movies. It should be noted that the one-line code snippet is identical for *Lord of the Rings.*



```
In [23]:  top10_chars_harry_potter = harry_potter_df.Character.value_counts().index.tolist()[:10]

In [24]:  top10_chars_harry_potter

Out[24]: ['HARRY',
          'RON',
          'HERMIONE',
          'HAGRID',
          'LUPIN',
          'DUMBLEDORE',
          'MCGONAGALL',
          'SNAPE',
          'DRACO',
          'MRS. WEASLEY']
```

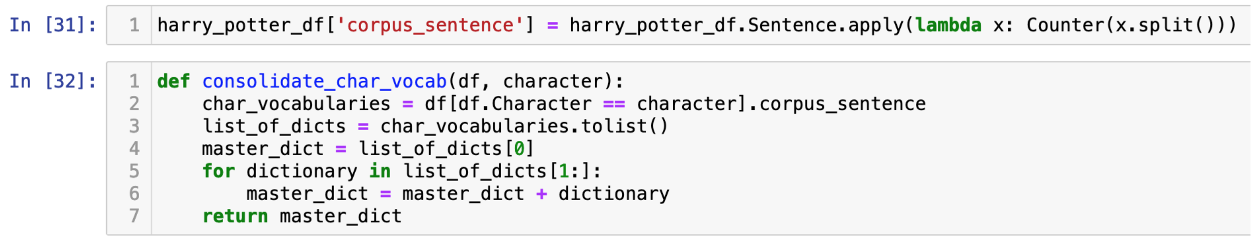


```
In [22]:  harry_potter_df.Character.value_counts()

Out[22]:  HARRY                          883
          RON                            485
          HERMIONE                       469
          HAGRID                         337
          LUPIN                          207
          DUMBLEDORE                     196
          MCGONAGALL                     132
          SNAPE                          118
          DRACO                           95
          MRS. WEASLEY                    75
```

Figure 5a. Code snippet to get top 10 characters in Harry Potter

Displaying the output of the *value_counts()* function tells a drastically different story. Harry Potter has the most speaking lines set at 883. Ron Weasley is in distant second place at 485 lines which is approximately a 45% decrease. Hermione has a similar count. The last 5 characters to round out the top 10 have less than 200 speaking lines. In fact, Draco Malfoy and Mrs. Weasley sitting at 9th and 10th place each have fewer than 100 lines. A person unfamiliar with the plot could easily ascertain that Harry, Ron, and Hermione are at the pinnacle of the story from the value_counts().
Now that we have determined the top 10 characters, we can build their character dictionaries. The reason we had to undergo this process was because looking at more than the top 10 characters would be both unfeasible and yield results that are statistically insignificant. **Figure 6** illustrates the method of creating character vocabulary. Cell 31 is used to create a dictionary from each line of dialog. The *consolidate_char_vocab()* function in cell 32 is applied in a loop to the top 10 characters discovered earlier to create a "mega" dictionary with a key representing the character name and the value being a dictionary of their vocabulary.



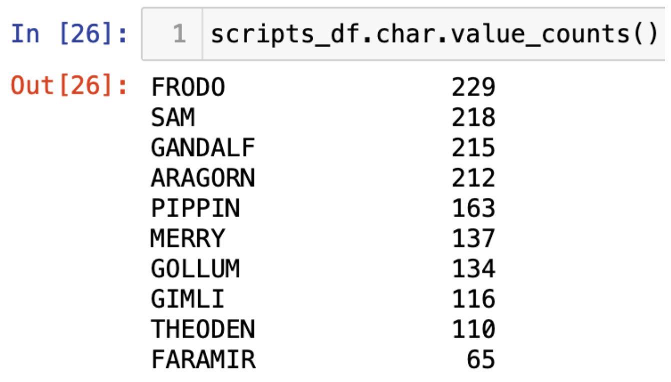
```
In [31]:  harry_potter_df['corpus_sentence'] = harry_potter_df.Sentence.apply(lambda x: Counter(x.split()))

In [32]:  def consolidate_char_vocab(df, character):
              char_vocabularies = df[df.Character == character].corpus_sentence
              list_of_dicts = char_vocabularies.tolist()
              master_dict = list_of_dicts[0]
              for dictionary in list_of_dicts[1:]:
                  master_dict = master_dict + dictionary
              return master_dict
```

Figure 6. Method for creating character vocabulary for the top 10 characters

Lord of the Rings

When viewed in conjunction with the results for *Harry Potter,* **Figure 5b** shows that even the top 10 characters in *Lord of The Rings* have relatively few lines. Harry Potter has almost 4 times as many lines as Frodo Baggins. The range of values for *Harry Potter* is over 800, while for *Lord of the Rings,* it is only 164. A casual observer would see that there is a much greater emphasis on body language and visuals/effects in *Lord of the Rings.* The process of creating a “mega” dictionary for *Lord of the Rings* is identical to the one depicted in **Figure 6.**



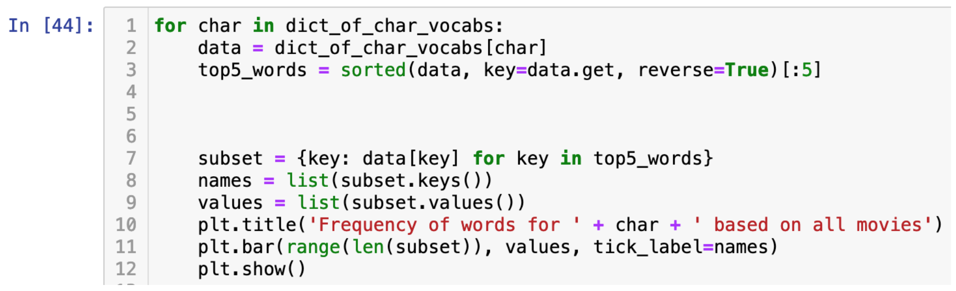
```
In [26]:  scripts_df.char.value_counts()

Out[26]: FRODO                 229
         SAM                   218
         GANDALF               215
         ARAGORN               212
         PIPPIN                163
         MERRY                 137
         GOLLUM                134
         GIMLI                 116
         THEODEN               110
         FARAMIR                65
```

Figure 5b. Character value counts for Lord of the Rings characters

III. Data Analytics and Visualization

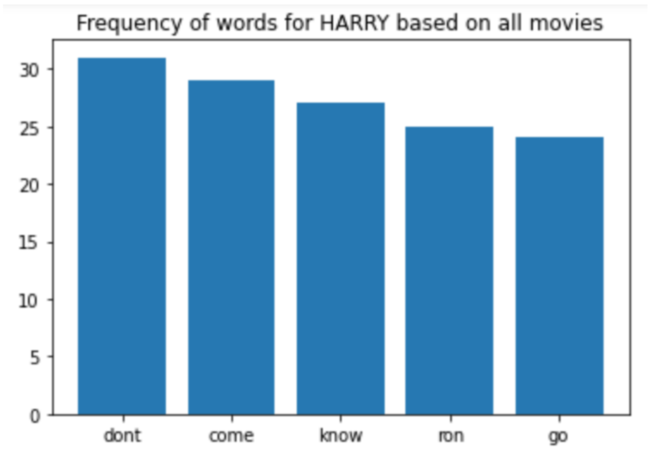
Now that the character vocabulary has been created for both elements one can stop talking about them separately. **Figure 7** shows the process by which the histogram of word frequencies is created for each of the top 10 characters. First the character’s vocabulary is selected from the “mega” dictionary based on the key. Next, the top 5 keys are extracted by sorting the keys in descending order and extracting the first 5 in the list. A subset dictionary is then created from these keys. Finally a matplotlib plot is created where the x axis is the word, and the y axis in the value count. For the sake of brevity, only the histograms for Harry Potter and Frodo Baggins are shown, extracted from two separate jupyter notebooks.



```
In [44]:
for char in dict_of_char_vocabs:
    data = dict_of_char_vocabs[char]
    top5_words = sorted(data, key=data.get, reverse=True)[:5]



    subset = {key: data[key] for key in top5_words}
    names = list(subset.keys())
    values = list(subset.values())
    plt.title('Frequency of words for ' + char + ' based on all movies')
    plt.bar(range(len(subset)), values, tick_label=names)
    plt.show()
```



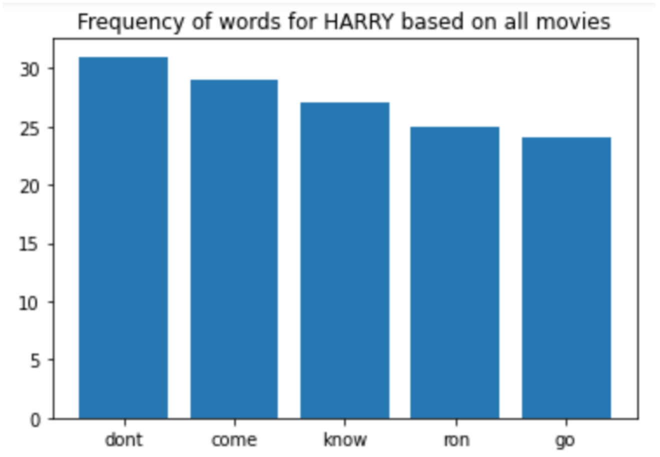


Figure 7. Creating the histogram of top 5 words for each character

IV. Conclusion

While the histograms in **Figure 7** provide basic information on characters, their relationships with other characters, and their temperament, they fail to show quantifiable analysis. For example, if a graph of the relationships between characters was included based on whether characters reference their friends a lot such as Frodo and Sam in *Lord of the Rings* and Harry and Ron in *Harry Potter,* a casual observer would get more useful information. The fact that I was able to build a character vocabulary is a proof of concept for far more exciting analysis. One could show character progression and growth by examining whether a character's top 5 words change and whether the rankings change or not. A change would indicate that a character has either grown or regressed. I hypothesize that such an analysis would indeed reveal that most of the top 3 characters experience drastic growth. Joining the character scripts with character data could help determine whether a character's features contribute to the way they speak in a nature versus nurture debate. Implementing Machine Learning models to test these hypotheses out would allow one to further create a quantifiable element.

Sources:

1. https://www.kaggle.com/gulsahdemiryurek/harry-potter-dataset
2. https://www.kaggle.com/paultimothymooney/lord-of-the-rings-data