26-06-2023

**Implementation of Sentiment Analysis Using VADER**

1. The code begins by importing the necessary libraries: `pandas` for data manipulation and analysis, and `nltk` (Natural Language Toolkit) for natural language processing tasks. Specifically, it imports the `SentimentIntensityAnalyzer` class from `nltk.sentiment.vader`, which is a pre-trained sentiment analysis tool.
2. **df\_avatar = pd.read\_csv('avatar.csv', engine='python')**

This line reads a CSV file named 'avatar.csv' into a pandas DataFrame called `df\_avatar`. The file is assumed to contain data related to the movie or show 'Avatar'.

1. **df\_avatar\_lines = df\_avatar.groupby('character').count()**

**df\_avatar\_lines = df\_avatar\_lines.sort\_values(by=['character\_words'], ascending=False)[:10]**

These two lines perform some data wrangling on `df\_avatar`. First, the DataFrame is grouped by the 'character' column, and then the count of occurrences for each character is calculated. The resulting DataFrame, `df\_avatar\_lines`, contains the number of lines spoken by each character. It is then sorted in descending order based on the 'character\_words' column, and only the top 10 characters (those with the highest line counts) are retained.

1. **top\_character\_names = df\_avatar\_lines.index.values**

This line extracts the character names from the index of the `df\_avatar\_lines` DataFrame and stores them in the `top\_character\_names` variable. These names represent the top 10 characters with the most lines.

1. **df\_character\_sentiment = df\_avatar[df\_avatar['character'].isin(top\_character\_names)]**

**df\_character\_sentiment = df\_character\_sentiment[['character', 'character\_words']]**

These lines filter the original `df\_avatar` DataFrame to keep only the rows where the 'character' column value is present in the `top\_character\_names` list. The resulting DataFrame, `df\_character\_sentiment`, retains only the 'character' and 'character\_words' columns.

1. **sid = SentimentIntensityAnalyzer()**

This line creates an instance of the `SentimentIntensityAnalyzer` class from the NLTK library, which will be used to calculate sentiment scores.

1. **df\_character\_sentiment.reset\_index(inplace=True, drop=True)**

**df\_character\_sentiment[['neg', 'neu', 'pos', 'compound']] = df\_character\_sentiment['character\_words'].apply(sid.polarity\_scores).apply(pd.Series)**

The first line resets the index of the `df\_character\_sentiment` DataFrame, dropping the old index and updating it to a default numeric index.

The second line applies the `polarity\_scores` method of the `SentimentIntensityAnalyzer` to each value in the 'character\_words' column. This method calculates sentiment scores, returning a dictionary of scores containing 'neg', 'neu', 'pos', and 'compound' values. The resulting Series of dictionaries is then expanded into separate columns in the DataFrame using the `apply(pd.Series)` method.

1. **df\_character\_sentiment**

This line displays the resulting DataFrame, `df\_character\_sentiment`, which now contains the sentiment scores ('neg', 'neu', 'pos', 'compound') for each character's lines.

**sample output:**









