```python
# modules.text_features.py
# https://github.com/QuantCS109/TrumpTweets/blob/master/modules/text_features.py
# This is a .py file used in the notebooks

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.decomposition import TruncatedSVD

import numpy as np
from scipy import sparse
import os

import pandas as pd


class TextFeaturesGenerator:

    def __init__(self,text_series=None,score_series=None):
        """
        :param text_series: A pandas series with the text
        """
        self.text_series = text_series
        self.count_vectorizer = None
        self.tfidf_vectorizer = None

        self.bow_mat = None
        self.tfidf_mat = None

        self.bow_mat_scored = None
        self.tfidf_mat_scored = None

        self.svd_bow_mat = None
        self.svd_tfidf_mat = None
        self.svd_bow_mat_scored = None
        self.svd_tfidf_mat_scored = None

        self.score_series = score_series

    def get_bow_matrix(self):
        """
        Returns:
            bow_matrix: A CSR (Compressed Sparse Row Matrix) of bag-of-words
            representation
            of the matrix
        """
        if self.bow_mat is None:
            self.count_vectorizer = CountVectorizer()
            self.count_vectorizer = self.count_vectorizer.fit(self.text_series)
            self.bow_mat = self.count_vectorizer.transform(self.text_series)
        if self.score_series is not None:
            self.bow_mat_scored = self.count_vectorizer.transform(self.score_series)
            return self.bow_mat, self.bow_mat_scored
        return self.bow_mat

    def get_tfidf_matrix(self):
        """
        Returns:
            bow_matrix: A CSR (Compressed Sparse Row Matrix) of tf-idf representation
            of the matrix
        """
        if self.tfidf_mat is None:
            if self.bow_mat is None:
                _ = self.get_bow_matrix()
            self.tfidf_vectorizer = TfidfTransformer(use_idf=True).fit(self.bow_mat)
            self.tfidf_mat = self.tfidf_vectorizer.transform(self.bow_mat)
        if self.score_series is not None:
            self.tfidf_mat_scored = self.tfidf_vectorizer.transform(self.bow_mat_scored)
            return self.tfidf_mat, self.tfidf_mat_scored
        return self.tfidf_mat

    def get_svd_bow_mat(self,n_components=2,
                         algorithm='randomized',
                         n_iter=5,
                         random_state=None,
                         tol=0.0):
        if self.bow_mat is None:
            _ = self.get_bow_matrix()
        svd_transformer = TruncatedSVD(n_components,algorithm,n_iter,
                                       random_state,tol).fit(self.bow_mat)
        self.svd_bow_mat = svd_transformer.transform(self.bow_mat)
        if self.score_series is not None:
            self.svd_bow_mat_scored = svd_transformer.transform(self.bow_mat_scored)
            return self.svd_bow_mat, self.svd_bow_mat_scored
        return self.svd_bow_mat

    def get_svd_tfidf_mat(self,n_components=2,
                         algorithm='randomized',
                         n_iter=5,
                         random_state=None,
                         tol=0.0):

        if self.tfidf_mat is None:
            _ = self.get_tfidf_matrix()
        svd_transformer = TruncatedSVD(n_components,algorithm,n_iter,
                                       random_state,tol).fit(self.tfidf_mat)
        self.svd_tfidf_mat= svd_transformer.transform(self.tfidf_mat)
        if self.score_series is not None:
            self.svd_tfidf_mat_scored = svd_transformer.transform(self.tfidf_mat_scored)
            return self.svd_tfidf_mat, self.svd_tfidf_mat_scored
        return self.svd_tfidf_mat


    def save_matrices(self,folder='../data/intermediate_data/',suffix=""):
        """
        Arguments:
        :param folder: Folder / directory in which to save the matrices
                        Will save in current working folder if not specified
        """
        if self.bow_mat is None:
            _ = self.get_bow_matrix()
        if self.tfidf_mat is None:
            _ = self.get_tfidf_matrix()
        if folder:
            if not os.path.exists(folder):
                os.makedirs(folder)
        bow_file = "tfidf_mat"+suffix+".npz"
        tfidif_file = "bow_mat" + suffix + ".npz"
        svd_bow_file = "svd_tfidf_mat"+suffix+".npy"
        svd_tfidif_file = "svd_bow_mat" + suffix + ".npy"

        bow_location = os.path.join(folder,bow_file) if folder else bow_file
        tfidf_location = os.path.join(folder,tfidif_file) if folder else tfidif_file
        svd_bow_location = os.path.join(folder, svd_bow_file) if folder else svd_bow_file
        svd_tfidf_location = os.path.join(folder, svd_tfidif_file) if folder else
        svd_tfidif_file

        sparse.save_npz(bow_location, self.bow_mat)
        sparse.save_npz(tfidf_location,self.tfidf_mat)
        np.save(svd_bow_location, self.svd_bow_mat) if \
            self.svd_bow_mat is not None else None
        np.save(svd_tfidf_location,self.svd_tfidf_mat) if \
            self.svd_tfidf_mat is not None else None


```

```python

from nltk.sentiment.vader import SentimentIntensityAnalyzer

class SentimentFeaturesGenerator:
    def __init__(self,tweet_df,aggregate=False):
        self.text = tweet_df.tweets
        self.tweets_df = tweet_df
        self.sid = SentimentIntensityAnalyzer()
        self.sentiment_df = pd.DataFrame()
        self.sentiment_series = None
        self.sentiment_df_aggregate = pd.DataFrame()
        self.aggregate = aggregate

    def get_sentiments(self):
        self.sentiment_series = self.text.map(self.sid.polarity_scores)
        self.sentiment_df['negative_proportion'] = self.sentiment_series.map(lambda x: x.get('neg'))
        self.sentiment_df['positive_proportion'] = self.sentiment_series.map(lambda x: x.get('pos'))
        self.sentiment_df['neutral_proportion'] = self.sentiment_series.map(lambda x: x.get('neu'))
        self.sentiment_df['combined_score'] = self.sentiment_series.map(lambda x: x.get('compound'))
        self.sentiment_df['date'] = self.tweets_df.after4_date
        self.sentiment_df.index = self.sentiment_series.index
        self.sentiment_df.index = pd.to_datetime(self.sentiment_df.index)

    def aggregate_sentiments(self):
        self.sentiment_df_aggregate = self.sentiment_df.groupby('date').agg(['min','max','mean'])
        self.sentiment_df_aggregate.columns = ["_".join([x[0], x[1]]) for x in\
                                               self.sentiment_df_aggregate.columns]
        self.sentiment_df_aggregate.index = pd.to_datetime(self.sentiment_df_aggregate.index)

    def run(self):
        self.get_sentiments()
        if self.aggregate:
            self.aggregate_sentiments()
```