```
# libraries for parsing and sentiment analysis
import nltk
# nltk corupus downloads
nltk.download('vader_lexicon')
nltk.download('stopwords')
nltk.download('punkt')
from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
sia = SIA()

from textblob import TextBlob
from nltk.corpus import stopwords
from newspaper import Article

# utility
from multiprocessing.dummy import Pool as ThreadPool
import re


def parse(story, text_only=False):
    try:
        article = Article(story['url'])
        article.download()
        article.parse()
        parsed = article.text
        if text_only:
            return parsed
        story['text'] = parsed
        return story
    except:
```

```
        return ""



def parse_async(stories):
    pool = ThreadPool(10)
    results = pool.map(parse, stories)
    pool.close()
    pool.join()
    return results



def polarize(headline, parsed=False):
    # function returns an object of two sepearate polarity scores; one based off the text of the article and the other
    # from just the headline alone. Each of these are represented in their own respective objects. Currently, headline_res
    # isn't being used publically.
    pol_obj = {}
    headline_res = {}
    article_res = {}


    """Extracting polarity from article text"""
    try:
        """Story might be a dictionary that's already been parsed
        to be used in order_pol() by async_parse (in the case of multiple articles). Alternatively, story could be passed from
        our SA api still needing to be parsed either as a sqlalchemy instance in a user's saved stories or a dictionary in the
        session result's list. Regardless, calling this function through the SA directly sends result back to form (button) that
```

```
        initiated the call, so there is no need to keep track of id and other information. Only text."""

        if parsed:
            parsed = headline # parses story text
            sentenced = nltk.tokenize.sent_tokenize(parsed['text']) # tokenizes story text by sentence
            headline = sia.polarity_scores(parsed['headline']) # nltk analysis is performed (this is where the magic happens)
        else: # same logic as above
            parsed = parse(headline, text_only=True)
            sentenced = nltk.tokenize.sent_tokenize(parsed)
            headline = sia.polarity_scores(headline['headline'])
    except:
        return None

    coms = []
    pos = []
    negs = []
    neus = []

    for sentence in sentenced:
        res = sia.polarity_scores(sentence) # nltk analysis is performed
        pos.append(res["pos"])
        negs.append(res["neg"])
        neus.append(res["neu"])
        if res['compound']:
            # sometimes the composite will be zero for certain sentences. We don't want to include that data.
            coms.append(res['compound'])
    if len(coms) == 0:
```

```
        return None

    avg_com = round((sum(coms) / len(coms)), 2)
    avg_pos = round((sum(pos) / len(pos)), 2)
    avg_neu = round((sum(neus) / len(neus)), 2)
    avg_neg = round((sum(negs) / len(negs)), 2)

    article_res["avg_com"] = round(avg_com, 2)
    article_res["avg_pos"] = round(avg_pos, 2)
    article_res["avg_neg"] = round(avg_neg, 2)
    article_res["avg_neu"] = round(avg_neu, 2)

    if avg_com >= 0.4:
        article_res['result'] = f"{avg_com} (Very Positive)"
    elif avg_com >= 0.2:
        article_res['result'] = f"{avg_com} (Positive)"
    elif avg_com <= - 0.2:
        article_res['result'] = f"{avg_com} (Negative)"
    elif avg_com <= - 0.2:
        article_res['result'] = f"{avg_com} (Very Negative)"
    else:
        article_res['result'] = f"{avg_com} (Neutral)"
    article_res["message"] = f"{article_res['result']}. {avg_neg *100}% Negative, {avg_neu *100}% Neutral,
and {avg_pos *100}% Positive"

    """Extracting polarity from headline text"""

    headline_res["com"] = headline['compound']
    headline_res["pos"] = headline['pos']
```

```
    headline_res["neg"] = headline['neg']
    headline_res["neu"] = headline['neu']
    if headline_res['com'] >= 0.2:
        headline_res['result'] = "Positive"
    elif headline_res['com'] >= 0.4:
        headline_res['result'] = "Very Positive"
    elif headline_res['com'] <= - 0.2:
        headline_res['result'] = "Negative"
    elif headline_res['com'] <= - 0.4:
        headline_res['result'] = "Very Negative"
    else:
        headline_res['result'] = "Neutral"
    pol_obj['headline_res'] = headline_res
    pol_obj['article_res'] = article_res
    return pol_obj



def subjectize(headline, parsed=False):
    try:
        if parsed:
            parsed = headline['text']
        else:
            parsed = parse(headline, text_only=True)
    except:
        return None
    tblobbed = TextBlob(parsed)
    subjectivity = round(tblobbed.sentiment.subjectivity, 2)
    subjectivity = str(subjectivity)
    subjectivity = subjectivity[-2:]
```

```
if subjectivity == ".":
    subjectivity = f"{subjectivity}0"
if subjectivity == ".0":
    return None
subjectivity = round(float(subjectivity))

sub_obj = {}
if subjectivity > 80:
    sub_obj['measure'] = f"{subjectivity}% (Very Objective)"
elif subjectivity > 60:
    sub_obj['measure'] = f"{subjectivity}% (Objective)"
elif subjectivity > 40:
    sub_obj['measure'] = f"{subjectivity}% (Neutral)"
elif subjectivity > 20:
    sub_obj['measure'] = f"{subjectivity}% (Subjective)"
else:
    sub_obj['measure'] = f"{subjectivity}% (Very Subjective)"
sub_obj['score'] = subjectivity

if sub_obj['score'] == 0.0:
    return None
return sub_obj
```