```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.metrics import silhouette_score
from textblob import TextBlob
from joblib import Parallel, delayed
import multiprocessing
#from wordcloud import WordCloud
import time


nltk.download('stopwords')
nltk.download('punkt')

start_time = time.time()
# 1. Datenquelle laden
data = pd.read_csv('justdoit_tweets_2018_09_07_2.csv',
usecols=['tweet_full_text'])

# 2. Daten vorverarbeiten
def clean_text(text):
    # Sonderzeichen entfernen bis auf #
    text = re.sub('[^a-zA-Z# ]', ' ', text)
    # Kleinbuchstaben umwandeln
    text = text.lower()
    # Wörter tokenisieren
    text = text.split()
    # Stoppwörter entfernen
    stop_words = set(stopwords.words('english'))
    # manuelle Stopwörter für bessere ergebnisse
    custom_stopwords = ["http", "https", "nike", "justdoit", "justdoit nike",
"#nike", "#justdoit", "realdonaldtrump"]
    stop_words.update(custom_stopwords)
    text = [word for word in text if not word in stop_words]
    # Token wieder zu Text zusammensetzen
    text = ' '.join(text)
    return text

# Token Pattern, um Hashtags als Tokens zu berücksichtigen
token_pattern = r"(?u)\b\w\w+\b|#\w+"

data['cleaned_text'] = data['tweet_full_text'].apply(lambda x: clean_text(x))

# 3. Text in numerische Vektoren umwandeln
cv = CountVectorizer(max_df=0.8, min_df=2, stop_words='english',
token_pattern=token_pattern)
doc_term_matrix = cv.fit_transform(data['cleaned_text'])

# 4. Themen extrahieren
# Latent Dirichlet Allocation (LDA)
```

```
#multiprocessing
def fit_lda(n_topics, doc_term_matrix):
        lda = LatentDirichletAllocation(n_components=n_topics, random_state=42)
        lda.fit(doc_term_matrix)
        transformed = lda.transform(doc_term_matrix)
        score = silhouette_score(doc_term_matrix, transformed.argmax(axis=1))
        return lda, score, n_topics

best_lda_score = -1
best_lda_model = None
best_lda_topics = 0

num_cores = multiprocessing.cpu_count()
results = Parallel(n_jobs=num_cores)(
    delayed(fit_lda)(n_topics, doc_term_matrix) for n_topics in range(5, 15)
)

for lda, score, n_topics in results:
    if score > best_lda_score:
        best_lda_score = score
        best_lda_model = lda
        best_lda_topics = n_topics

print(f'Bestes LDA model hat {best_lda_topics} Themen mit einer
Silhouettenpunktzahl von {best_lda_score}')

# Top 10 Wörter pro Thema ausgeben
vocab = cv.vocabulary_
for i, topic in enumerate(best_lda_model.components_):
    print(f'Top 10 Wörter nach Themen #{i}:')
    word_probs = [(word, topic[index]) for word, index in vocab.items()]
    sorted_words = sorted(word_probs, key=lambda x: x[1], reverse=True)[:10]
    print("LDA:", [word[0] for word in sorted_words])
    print()
# 5. Sentimentanalyse durchführen
def get_sentiment(text):

 #   Berechnet das Sentiment (Polarität und Subjektivität) eines Textes.

    blob = TextBlob(text)
    sentiment = blob.sentiment
    return sentiment.polarity, sentiment.subjectivity

data['sentiment'] = data['cleaned_text'].apply(lambda x: get_sentiment(x))

# Anzahl positiver und negativer Bewertungen zählen
pos_count = 0
neg_count = 0
for polarity, subjectivity in data['sentiment']:
    if polarity > 0:
        pos_count += 1
    else:
        neg_count += 1
```

```
print(f'Positive bewertungen: {pos_count}')
print(f'Negative bewertungen: {neg_count}')

# Erstellen einer Liste aller Wörter in den Tweets
all_words = ' '.join(data['cleaned_text'])

# Erstellen einer WordCloud
"""
wordcloud = WordCloud(width=800, height=500, random_state=42,
background_color='white',
stopwords=stopwords.words('english')).generate(all_words)

# Zeigen Sie die WordCloud an
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
"""

print("\n\n")

# 6. Themen extrahieren
# Bigramme

cv_bigram = CountVectorizer(ngram_range=(2, 2), stop_words='english',
token_pattern=token_pattern)
doc_term_matrix_bigram = cv_bigram.fit_transform(data['cleaned_text'])
vocab_bigram = cv_bigram.vocabulary_
# Themen extrahieren mit Bigrammen
lda_bigram = LatentDirichletAllocation(n_components=best_lda_topics,
random_state=42)
lda_bigram.fit(doc_term_matrix_bigram)
transformed_bigram = lda_bigram.transform(doc_term_matrix_bigram)
data['sentiment'] = data['cleaned_text'].apply(lambda x: get_sentiment(x))
# Gesamtsentiment und kombinierter Score für jedes Thema
combined_scores = {i: [] for i in range(best_lda_topics)}

for doc_index, topic_probs in enumerate(transformed_bigram):
    main_topic = np.argmax(topic_probs)
    polarity, subjectivity = data['sentiment'].iloc[doc_index]
    combined_score = polarity * subjectivity
    combined_scores[main_topic].append(combined_score)

# Bigramme, Durchschnittssentiment und kombinierter Score für jedes Thema
ausgeben
for i, topic in enumerate(lda_bigram.components_):
    print(f'Top 10 nach Themen #{i + 1}:')
    bigram_probs = [(bigram, topic[index]) for bigram, index in
vocab_bigram.items()]
    sorted_bigrams = sorted(bigram_probs, key=lambda x: x[1], reverse=True)[:10]
    print("Bigramme:", ", ".join([bigram[0] for bigram in sorted_bigrams]))
    avg_polarity = np.mean(combined_scores[i])
    sentiment = "positiv" if avg_polarity > 0 else "negativ" if avg_polarity < 0
```

```
else "neutral"
    print(f'Sentiment: {sentiment} (Durchschnittspolarität:
{avg_polarity:.2f})')
    print("\n")

# Gesamtsentiment und kombinierter Score aller Dokumente ausgeben
all_polarities = [polarity for polarities in combined_scores.values() for
polarity in polarities]
avg_overall_polarity = np.mean(all_polarities)
overall_sentiment = "positiv" if avg_overall_polarity > 0 else "negativ" if
avg_overall_polarity < 0 else "neutral"


print(f'Gesamtsentiment aller Dokumente: {overall_sentiment}
(Durchschnittspolarität: {avg_overall_polarity:.2f})')


end_time = time.time()
print('Laufzeit:', end_time - start_time)
```