Text Analysis using Python

Let's get started with the task of Text Analysis by importing the necessary Python libraries and the dataset:

```
import pandas as pd
import plotly.express as px
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from textblob import TextBlob
import spacy
from collections import defaultdict
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

nlp = spacy.load('en_core_web_sm')
data = pd.read_csv("/content/articles.csv",encoding='latin-1')
print(data.head())
```

```
                                             Article  \
0  Data analysis is the process of inspecting and...
1  The performance of a machine learning algorith...
2  You must have seen the news divided into categ...
3  When there are only two classes in a classific...
4  The Multinomial Naive Bayes is one of the vari...

                                               Title
0                  Best Books to Learn Data Analysis
1          Assumptions of Machine Learning Algorithms
2           News Classification with Machine Learning
3  Multiclass Classification Algorithms in Machin...
4         Multinomial Naive Bayes in Machine Learning
```

The problem we are working on requires us to:

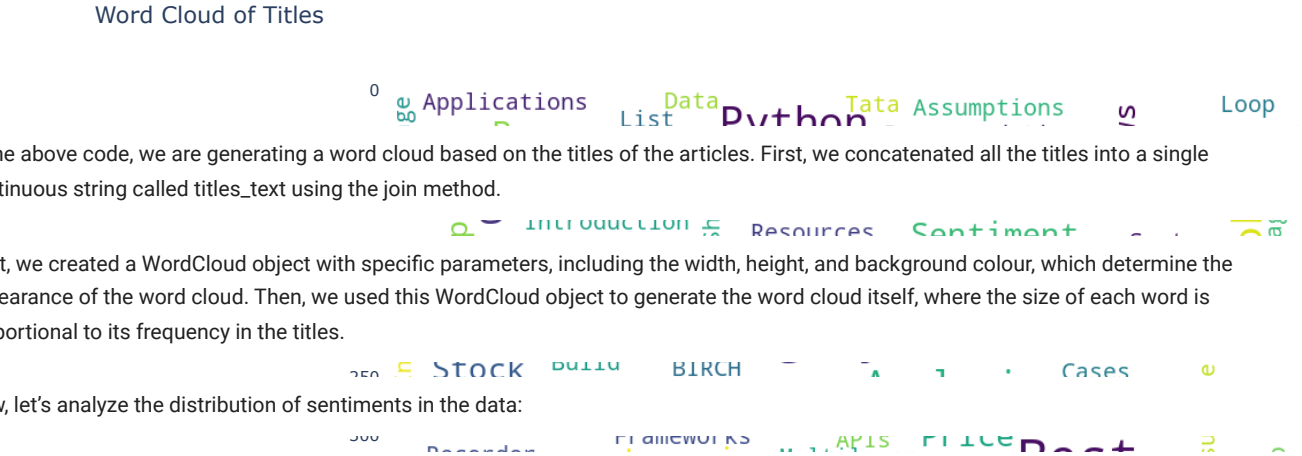
- Create word clouds to visualize the most frequent words in the titles.
- Analyze the sentiment expressed in the articles to understand the overall tone or sentiment of the content.
- Extract named entities such as organizations, locations, and other relevant information from the articles.
- Apply topic modelling techniques to uncover latent topics within the articles.

Now, let's move forward by visualizing the word cloud of the titles:

```
# Combine all titles into a single string
titles_text = ' '.join(data['Title'])

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(titles_text)

# Plot the Word Cloud
fig = px.imshow(wordcloud, title='Word Cloud of Titles')
fig.update_layout(showlegend=False)
fig.show()
```



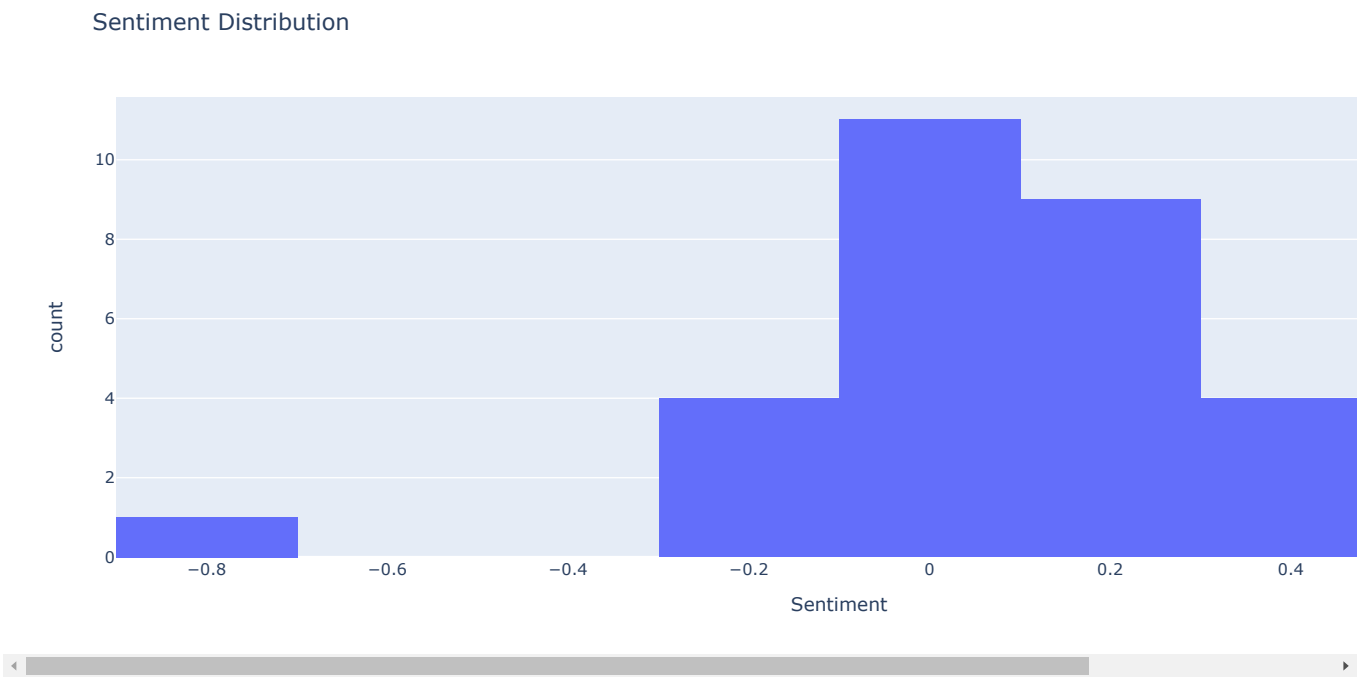
In the above code, we are generating a word cloud based on the titles of the articles. First, we concatenated all the titles into a single continuous string called titles_text using the join method.

Next, we created a WordCloud object with specific parameters, including the width, height, and background colour, which determine the appearance of the word cloud. Then, we used this WordCloud object to generate the word cloud itself, where the size of each word is proportional to its frequency in the titles.

Now, let's analyze the distribution of sentiments in the data:

```
# Sentiment Analysis
data['Sentiment'] = data['Article'].apply(lambda x: TextBlob(x).sentiment.polarity)

# Sentiment Distribution
fig = px.histogram(data, x='Sentiment', title='Sentiment Distribution')
fig.show()
```



In the above code, sentiment analysis is performed on the articles in the dataset to assess the overall sentiment or emotional tone of the articles. The TextBlob library is used here to analyze the sentiment polarity, which quantifies whether the text expresses positive, negative, or neutral sentiment.

The sentiment.polarity method of TextBlob calculates a sentiment polarity score for each article, where positive values indicate positive sentiment, negative values indicate negative sentiment and values close to zero suggest a more neutral tone. After calculating the sentiment polarities, a histogram is created to visualize the distribution of sentiment scores across the articles.

Now, let's perform Named Entity Recognition:

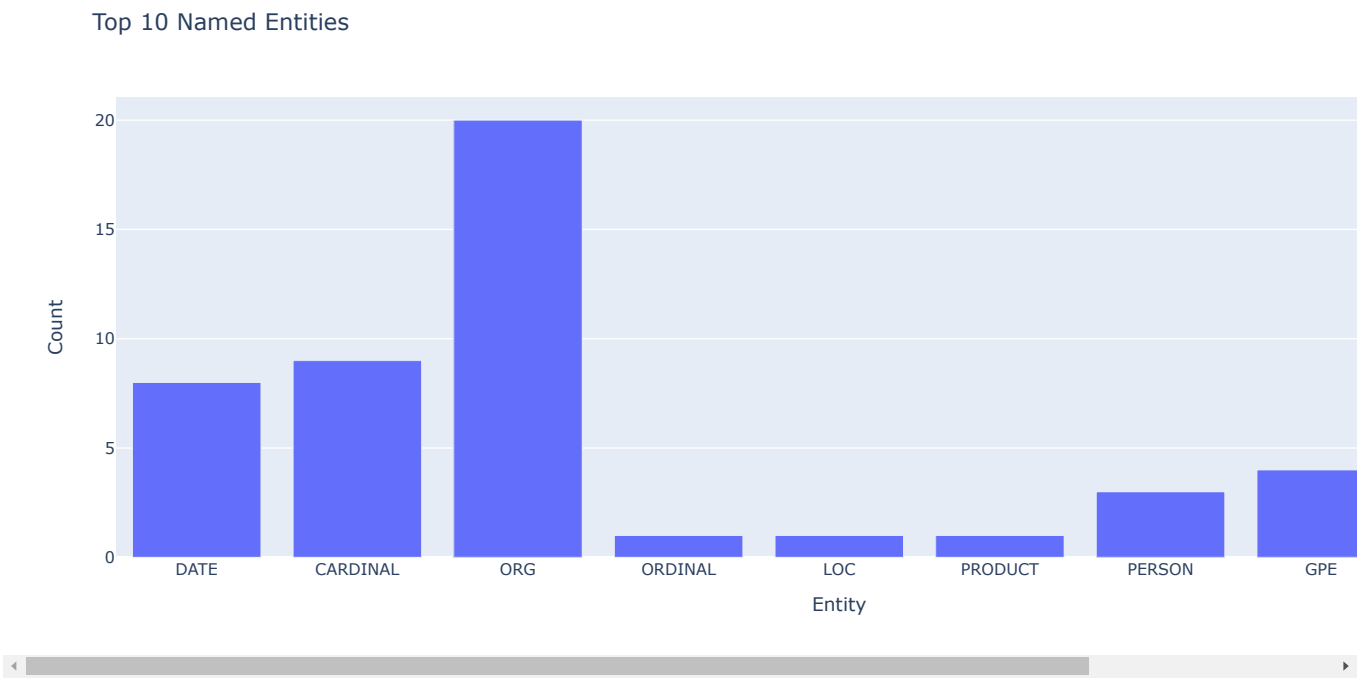
```
# NER
def extract_named_entities(text):
    doc = nlp(text)
    entities = defaultdict(list)
    for ent in doc.ents:
        entities[ent.label_].append(ent.text)
    return dict(entities)

data['Named_Entities'] = data['Article'].apply(extract_named_entities)

# Visualize NER
```

```
entity_counts = Counter(entity for entities in data['Named_Entities'] for entity in entities)
entity_df = pd.DataFrame.from_dict(entity_counts, orient='index').reset_index()
entity_df.columns = ['Entity', 'Count']

fig = px.bar(entity_df.head(10),x='Entity',y='Count',title='Top 10 Named Entities')
fig.show()
```



In the above code, we are performing Named Entity Recognition. NER is a natural language processing technique used to identify and extract specific entities such as organizations, locations, names, dates, and more from the text. The extract_named_entities function leverages the spaCy library to analyze each article, identify entities, and categorize them by their respective labels (e.g., “ORG” for organizations, “LOC” for locations).

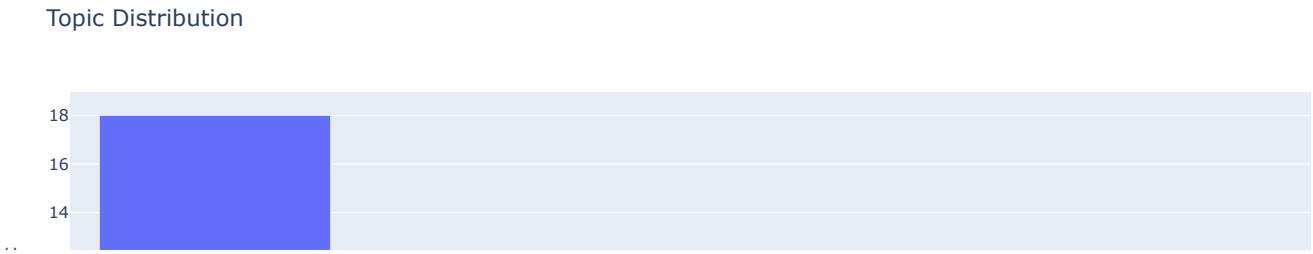
The extracted entities are stored in a new column called Named_Entities in the dataset. Then, a visualization is created to present the top 10 most frequently occurring named entities and their respective counts, allowing for a quick understanding of the prominent entities mentioned in the text data.

Now, let’s perform Topic Modelling:

```
# Topic Modeling
vectorizer = CountVectorizer(max_df=0.95, min_df=2, max_features=1000, stop_words='english')
tf = vectorizer.fit_transform(data['Article'])
lda_model = LatentDirichletAllocation(n_components=5, random_state=42)
lda_topic_matrix = lda_model.fit_transform(tf)

# Visualize topics
topic_names = ["Topic " + str(i) for i in range(lda_model.n_components)]
data['Dominant_Topic'] = [topic_names[i] for i in lda_topic_matrix.argmax(axis=1)]

fig = px.bar(data['Dominant_Topic'].value_counts().reset_index(),x='index',y='Dominant_Topic',title='Topic Distribution')
fig.show()
```



In the above code, we are performing Topic Modelling using Latent Dirichlet Allocation (LDA), a popular technique for uncovering latent topics within a corpus of text documents. First, we apply text vectorization using the CountVectorizer to convert the text data into a numerical format suitable for modelling.

Then we specified parameters such as maximum and minimum document frequency and the maximum number of features (words) to consider, while also removing common English stopwords.

Next, we applied LDA using the LatentDirichletAllocation model with five topics as an example. The resulting topic matrix represents each article's distribution across these five topics. Then, we assign a dominant topic to each article based on the topic with the highest probability, and a bar chart is generated to visualize the distribution of dominant topics across the dataset, providing an overview of the prevalent themes or subjects discussed in the articles.

So this is how we can perform Text Analysis using Python.

Summary

Text Analysis involves various techniques such as text preprocessing, sentiment analysis, named entity recognition, topic modelling, and text classification. Text analysis plays a crucial role in understanding and making sense of large volumes of text data, which is prevalent in various domains, including news articles, social media, customer reviews, and more. I hope you liked this article on Text Analysis using Python.