**ChatGPT – Sentiment Analysis DataFrame**

for my digital humanities project where i had to explore and visualize dataframes made through different processes and stored as different dataframes such as tf-idf, topic modeling, n grams and article lengths. Now i also have an option to take the articles and make my own dataframe. Could you suggest me a method through which i can make my own dataframe using python

**ChatGPT said:**

That’s a great opportunity! If you’re allowed to create your **own DataFrame from the articles**, you can take a creative approach depending on what you'd like to analyze or visualize that isn't already covered by the TF-IDF, topic modeling, n-grams, or article lengths. Here's a beginner-friendly method you can follow using Python and pandas, including suggestions on what kind of custom analysis you could do.

**✅ Goal: Create Your Own Article-Based DataFrame**

You can extract various features from the articles to create a new DataFrame. Here's one approach:

**🔍 Method: Sentiment Analysis + Metadata Features**

**🎯 What this DataFrame could include:**

| **Article** | **Date** | **Title** | **Sentiment** | **Polarity** | **Subjectivity** | **Word Count** | **Keyword Count (e.g., “climate”)** |
| --- | --- | --- | --- | --- | --- | --- | --- |

**🔧 Step-by-Step Guide in Python**

**🔽 Step 1: Import Libraries**

python

CopyEdit

import pandas as pd

from textblob import TextBlob # For sentiment analysis

**🔽 Step 2: Sample Articles Input (replace with your real data)**

Let’s say your article dataset looks like this:

python

CopyEdit

# Sample list of articles

data = [

{"title": "Climate change impacts rising", "date": "2024-05-01", "text": "The climate crisis is intensifying..."},

{"title": "Economic growth slows down", "date": "2024-05-02", "text": "Recent figures show economic uncertainty..."},

]

# Convert to a DataFrame

df\_articles = pd.DataFrame(data)

**🔽 Step 3: Define a Function to Analyze Each Article**

python

CopyEdit

def analyze\_article(text):

blob = TextBlob(text)

sentiment = blob.sentiment

polarity = sentiment.polarity # -1 = negative, +1 = positive

subjectivity = sentiment.subjectivity # 0 = objective, 1 = subjective

word\_count = len(text.split())

keyword = "climate"

keyword\_count = text.lower().count(keyword)

return pd.Series([polarity, subjectivity, word\_count, keyword\_count])

**🔽 Step 4: Apply the Function and Add Columns to Your DataFrame**

python

CopyEdit

df\_articles[['polarity', 'subjectivity', 'word\_count', 'climate\_mentions']] = df\_articles['text'].apply(analyze\_article)

**📊 Step 5: Visualize or Analyze Your Custom DataFrame**

You can now use df\_articles to plot sentiment trends over time, word counts vs. sentiment, climate-related mentions, etc.

**📌 Additional Ideas for Custom DataFrames**

Here are more ideas if you want to go beyond sentiment:

| **Method** | **Feature** |
| --- | --- |
| Named Entity Recognition | Count of people, places, organizations |
| Readability Scores | Using libraries like textstat |
| Custom Keyword Themes | Count of words from user-defined theme lists |
| Paragraph Count | Number of paragraphs or sections in an article |

Would you like help implementing one of these specific ideas (e.g., readability, named entities, or custom themes)? I can walk you through those too.

**You said:**

are there other ways i can make a dataframe

**ChatGPT said:**

Yes, there are **many creative ways** to make your own DataFrame depending on what you want to explore in your digital humanities project. Since you're working with **articles**, here are several different **categories of methods** to build a new DataFrame, each with a brief example idea.

**🧠 1. Semantic Features DataFrame**

Analyze deeper meaning from text.

**Example:**

| **Article** | **Title** | **Topic** | **Most Frequent Verb** | **Named People** | **Summary** |
| --- | --- | --- | --- | --- | --- |

**Tools:**

* spaCy (for Named Entity Recognition, verb extraction)
* sumy or gensim (for text summarization)

**📏 2. Structural Features DataFrame**

Focus on how articles are written or organized.

**Example:**

| **Article** | **Paragraph Count** | **Sentence Count** | **Avg Sentence Length** | **Title Length** |
| --- | --- | --- | --- | --- |

**Tools:**

* Basic Python string methods
* nltk or spaCy for sentence/paragraph tokenization

**📈 3. Time-based Trends DataFrame**

Track changes in article features over time.

**Example:**

| **Date** | **Avg Word Count** | **Avg Polarity** | **Climate Mentions** | **Top Topic of the Month** |
| --- | --- | --- | --- | --- |

**Tools:**

* pandas groupby() on date/month/year
* Combine with existing topic modeling or TF-IDF columns

**🧮 4. Custom Wordlist Matching**

Create features based on **themes or topics** you're personally interested in.

**Example:**

| **Article** | **Mentions of “justice”** | **Mentions of “freedom”** | **Mentions of “climate”** | **Human Rights Theme Score** |
| --- | --- | --- | --- | --- |

**Tools:**

* Manual or CSV list of thematic keywords
* Use .count() or regex matching per article

**🧪 5. Stylometry / Authorial Style**

If your articles are from multiple sources/authors, you can compare style.

**Example:**

| **Author** | **Avg Sentence Length** | **Lexical Diversity** | **Passive Voice Usage** | **Readability Score** |
| --- | --- | --- | --- | --- |

**Tools:**

* textstat for readability
* nltk or spaCy for passive voice detection

**📚 6. Intertextuality / Citation Mentions**

If your articles reference people, documents, or previous articles.

**Example:**

| **Article** | **Mentions of UN** | **Mentions of Constitution** | **Quotation Count** | **Mentions of Other Authors** |
| --- | --- | --- | --- | --- |

**Tools:**

* Regex
* Named Entity Recognition with spaCy

**⚙️ 7. Combined Features from Multiple DataFrames**

You can **merge** your TF-IDF, n-gram, topic, and article-length DataFrames into one unified table with shared columns like Title, Date, Article.

Then you can add new custom features:

* Total topic overlap
* Length-weighted keyword density
* TF-IDF of only nouns/adjectives

**💡 Want an Example?**

Let me know if you'd like to:

* Extract keywords based on themes
* Count stylistic features (e.g., rhetorical questions)
* Summarize articles into one-sentence abstracts
* Merge multiple dataframes into one

I can walk you through **exact code** for any of these ideas based on what you want to explore. What direction sounds interesting to you?

**You said:**

could you please explain the method for sentiment analysis in a bit more detail

**ChatGPT said:**

Absolutely! Let's break down the **sentiment analysis method** step-by-step so you understand exactly how it works and how you can create a custom DataFrame using it in your Digital Humanities project.

**🧠 What is Sentiment Analysis?**

Sentiment analysis tells us whether a piece of text is:

* **Positive**, **Negative**, or **Neutral** (this is called **polarity**)
* **Objective** or **Subjective** (this is called **subjectivity**)

📌 **Polarity** ranges from -1 (very negative) to +1 (very positive)  
📌 **Subjectivity** ranges from 0 (very objective/factual) to 1 (very subjective/opinionated)

**🧰 Tool We’ll Use: TextBlob**

TextBlob is a simple Python library that uses natural language processing (NLP) to analyze text. It works great for small and medium-sized text projects like analyzing articles.

**✅ Step-by-Step Implementation**

**🔽 1. Install TextBlob (only once)**

If you haven’t installed TextBlob yet, you can do so in your terminal or notebook:

bash

CopyEdit

pip install textblob

You may also need to download some basic NLP data:

bash

CopyEdit

python -m textblob.download\_corpora

**🔽 2. Import Required Libraries**

python

CopyEdit

import pandas as pd

from textblob import TextBlob

**🔽 3. Prepare Your Articles Data**

Here’s an example of a simple dataset with a few sample articles. You should replace this with your own article data (titles, dates, texts):

python

CopyEdit

data = [

{

"title": "Climate Crisis Worsens",

"date": "2024-10-01",

"text": "The effects of climate change are becoming more severe each year. People are suffering, and action is urgently needed."

},

{

"title": "Economic Growth in Developing Countries",

"date": "2024-10-02",

"text": "Economic indicators show positive growth in several developing nations. Many experts are optimistic about continued progress."

},

{

"title": "Corruption Scandal Hits Local Government",

"date": "2024-10-03",

"text": "A major scandal has shocked the city, with several officials accused of embezzlement and fraud."

}

]

# Convert list to pandas DataFrame

df = pd.DataFrame(data)

**🔽 4. Define a Function to Analyze Sentiment**

We’ll use TextBlob to get polarity and subjectivity for each article.

python

CopyEdit

def get\_sentiment(text):

blob = TextBlob(text) # Create a TextBlob object

polarity = blob.sentiment.polarity # Sentiment score (-1 to 1)

subjectivity = blob.sentiment.subjectivity # Subjectivity score (0 to 1)

return pd.Series([polarity, subjectivity])

**🔽 5. Apply the Function to Your Articles**

This will add new columns to your DataFrame:

python

CopyEdit

df[['polarity', 'subjectivity']] = df['text'].apply(get\_sentiment)

**🖼️ 6. Resulting DataFrame**

You now have:

| **title** | **date** | **text** | **polarity** | **subjectivity** |
| --- | --- | --- | --- | --- |
| Climate Crisis Worsens | 2024-10-01 | "The effects of climate..." | -0.25 | 0.6 |
| Economic Growth in Developing... | 2024-10-02 | "Economic indicators show..." | 0.5 | 0.35 |
| Corruption Scandal Hits Local Gov. | 2024-10-03 | "A major scandal has shocked..." | -0.3 | 0.5 |

**📊 What You Can Do with This**

Now that your DataFrame includes sentiment data, you can:

* **Plot trends over time**  
  e.g., polarity by date → Line chart of mood in media over months
* **Compare themes**  
  Group by keywords or topics and compare subjectivity levels
* **Identify extreme articles**  
  Filter for articles with polarity < -0.5 (very negative) or subjectivity > 0.8 (very opinionated)

**✨ Optional Add-Ons**

You can also add:

* word\_count → len(text.split())
* climate\_mention\_count → text.lower().count('climate')

Let me know if you want a combined version of this script with all of these features and chart examples using Plotly or matplotlib.

**You said:**

i will not have to write the article data as i can set path to that folder where the articles are present

**ChatGPT said:**

Perfect — if your articles are stored in files (like .txt, .csv, .json, or .html) in a **folder**, you can absolutely set a path to that folder and automatically read the articles into your DataFrame.

Below is a **step-by-step method** to load articles from a folder and perform **sentiment analysis** on them using TextBlob.

**✅ Goal**

* Load all text files from a folder (each file = one article)
* Create a DataFrame with file name, content, and sentiment scores
* (Optional) Add word count, subjectivity, keyword mentions

**🧰 Libraries Required**

python

CopyEdit

import os

import pandas as pd

from textblob import TextBlob

**📂 Folder Setup**

Let’s assume your folder path is:

bash

CopyEdit

/Users/yourname/Documents/articles/

And it contains files like:

CopyEdit

article1.txt

article2.txt

article3.txt

**🔽 Step-by-Step Code**

**🔹 1. Set the Path to Your Folder**

python

CopyEdit

folder\_path = '/Users/yourname/Documents/articles/' # Change this to your actual path

**🔹 2. Read All .txt Files in That Folder**

python

CopyEdit

articles = []

for filename in os.listdir(folder\_path):

if filename.endswith('.txt'):

with open(os.path.join(folder\_path, filename), 'r', encoding='utf-8') as file:

text = file.read()

articles.append({

'file': filename,

'text': text

})

**🔹 3. Convert to a DataFrame**

python

CopyEdit

df = pd.DataFrame(articles)

**🔹 4. Define a Function for Sentiment and Other Features**

python

CopyEdit

def analyze\_text(text):

blob = TextBlob(text)

polarity = blob.sentiment.polarity

subjectivity = blob.sentiment.subjectivity

word\_count = len(text.split())

climate\_mentions = text.lower().count("climate")

return pd.Series([polarity, subjectivity, word\_count, climate\_mentions])

**🔹 5. Apply the Function to Your DataFrame**

python

CopyEdit

df[['polarity', 'subjectivity', 'word\_count', 'climate\_mentions']] = df['text'].apply(analyze\_text)

**✅ Final DataFrame Example**

| **file** | **polarity** | **subjectivity** | **word\_count** | **climate\_mentions** |
| --- | --- | --- | --- | --- |
| article1.txt | 0.25 | 0.55 | 542 | 3 |
| article2.txt | -0.3 | 0.6 | 489 | 0 |
| ... | ... | ... | ... | ... |

**📊 Optional: Visualize Results**

You can now easily visualize trends, for example:

python

CopyEdit

import plotly.express as px

px.histogram(df, x='polarity', title='Distribution of Article Polarity')

**💬 Want to Read .csv, .json, or .html Instead?**

Let me know the format of your article files (e.g., .csv with text column, .json, etc.) and I can give you an adapted version for that file type.

Would you like help customizing the script to your exact folder or file format?

**You said:**

def analyze\_text(text): blob = TextBlob(text) polarity = blob.sentiment.polarity subjectivity = blob.sentiment.subjectivity word\_count = len(text.split()) climate\_mentions = text.lower().count("climate") return pd.Series([polarity, subjectivity, word\_count, climate\_mentions]) could you explain this code in more detail and also i don't want to find climate mentions

**ChatGPT said:**

Absolutely! Let's go through your function line-by-line in a very **beginner-friendly and detailed** way, and we’ll remove the climate\_mentions part as requested.

**✅ Your Cleaned Function (without climate mentions):**

python

CopyEdit

def analyze\_text(text):

blob = TextBlob(text)

polarity = blob.sentiment.polarity

subjectivity = blob.sentiment.subjectivity

word\_count = len(text.split())

return pd.Series([polarity, subjectivity, word\_count])

**🔍 Detailed Line-by-Line Explanation**

**🧱 def analyze\_text(text):**

* This line defines a new **function** called analyze\_text.
* It takes one **input parameter**: text, which is expected to be a **string** (like an article or paragraph).
* You will later use this function on every article in your DataFrame.

**🧠 blob = TextBlob(text)**

* This line creates a **TextBlob object** using the article text.
* TextBlob is a tool that helps analyze text using NLP (Natural Language Processing).
* Once you turn the text into a blob, you can ask it questions like:
  + "How positive or negative is this?"
  + "How opinionated is this?"

**📊 polarity = blob.sentiment.polarity**

* This line **extracts the polarity score** of the text.
* polarity is a **number between -1 and +1**:
  + **-1** → very negative (e.g., “terrible” or “disaster”)
  + **0** → neutral (e.g., “the sky is blue”)
  + **+1** → very positive (e.g., “amazing” or “happy”)

**📣 subjectivity = blob.sentiment.subjectivity**

* This line **extracts the subjectivity score** of the text.
* subjectivity is a **number between 0 and 1**:
  + **0** → very objective (factual or neutral tone)
  + **1** → very subjective (opinionated, emotional, or personal tone)

**🧮 word\_count = len(text.split())**

* This counts how many **words** are in the article.
* Here's how it works:
  + text.split() splits the text into **a list of words** using spaces
  + len(...) counts how many words are in that list
* So this line tells you how long the article is in terms of **word count**

**📦 return pd.Series([polarity, subjectivity, word\_count])**

* This line returns the results as a **pandas Series**, which is like a small labeled list.
* Each value (polarity, subjectivity, word\_count) will go into a **new column** in your DataFrame when you apply this function.

**🧪 Example Output**

If you use this function on an article like:

python

CopyEdit

text = "The event was incredible and inspiring. Everyone was happy and hopeful."

The result might be:

| **Polarity** | **Subjectivity** | **Word Count** |
| --- | --- | --- |
| 0.8 | 0.9 | 10 |

**✅ Final Tip: Add Column Names When Applying**

To make sure the new columns have clear names in your DataFrame, you can do:

python

CopyEdit

df[['polarity', 'subjectivity', 'word\_count']] = df['text'].apply(analyze\_text)

Let me know if you’d like to:

* Visualize these columns
* Add more features like title length or reading difficulty
* Save your DataFrame to a CSV file

Happy to help!

**You said:**

could you please make the code a bit easier or is there another way to do sentiment analysis

**ChatGPT said:**

Absolutely! Here's a **simpler version** of your sentiment analysis code with **fewer steps** but still does the same job. It uses the same tools (TextBlob, pandas, os) but tries to be more direct and readable.

**✅ Easier Version of Your Code**

python

CopyEdit

# Import libraries

import os

import pandas as pd

from textblob import TextBlob

import nltk

nltk.download("punkt")

# Set the path to the folder containing your articles

folder\_path = "../articles"

# Create a list to hold dictionaries (each dictionary = one article)

data = []

# Loop through each text file in the folder

for filename in os.listdir(folder\_path):

if filename.endswith(".txt"):

with open(os.path.join(folder\_path, filename), "r", encoding="utf-8") as file:

text = file.read()

blob = TextBlob(text)

polarity = blob.sentiment.polarity

subjectivity = blob.sentiment.subjectivity

# Add the data as a dictionary

data.append({

"filename": filename,

"text": text,

"polarity": polarity,

"subjectivity": subjectivity

})

# Create the DataFrame from the list of dictionaries

df = pd.DataFrame(data)

# Save it as a CSV file

df.to\_csv("sentiment\_analysis\_dataframe.csv", index=False)

# Show the first few rows

print(df.head())

**🧠 What's Different or Easier?**

| **Feature** | **Original Code** | **Easier Version** |
| --- | --- | --- |
| Variables for each column | Separate lists (filenames, texts, etc.) | Just one list of dictionaries (data) |
| Data storing style | Add to each list separately | Add all values together using a dictionary |
| Overall size | More lines | Fewer lines |
| Clarity | Detailed step-by-step | Cleaner but still readable |

**✅ Alternative Method: Using VADER (for social media, reviews, etc.)**

If you're analyzing **short texts like tweets or news headlines**, you can also use VADER (Valence Aware Dictionary and sEntiment Reasoner). It's better at handling:

* Sarcasm
* Emojis
* Slang
* Exclamation marks

Let me know if you’d like me to show you the VADER version too — it’s also simple!

Would you like that?

**You said:**

could you explain the function of these two import nltk nltk.download("punkt")

**ChatGPT said:**

Absolutely! Let’s break it down in a very beginner-friendly way:

**📦 import nltk**

* This line **imports the NLTK library** (Natural Language Toolkit).
* NLTK is a powerful Python library used for **working with human language data** (also called *natural language processing* or NLP).
* It helps with things like:
  + Breaking text into words or sentences (tokenization)
  + Finding parts of speech (nouns, verbs, etc.)
  + Working with grammar and text analysis

**🌐 nltk.download("punkt")**

* This line **downloads a specific set of tools called "punkt"**.
* "punkt" is a **tokenizer** — it helps break text into:
  + **Sentences** → like splitting a paragraph into separate sentences.
  + **Words** → like splitting a sentence into individual words.

Think of it as downloading a dictionary and grammar rules that help Python understand where one sentence ends and another begins.

**❓ Why Do We Need It for TextBlob?**

* TextBlob internally uses NLTK’s **tokenizer** for some of its functions like:
  + Splitting text into words or sentences
  + Calculating sentiment (it needs to know where the words and sentences are)
* So when you write:

python

CopyEdit

from textblob import TextBlob

blob = TextBlob("This is good.")

blob.sentiment

Behind the scenes, TextBlob uses **punkt** to understand the structure of your sentence.

**✅ Summary**

| **Line** | **What It Does** |
| --- | --- |
| import nltk | Brings in the NLTK library to use in your code |
| nltk.download("punkt") | Downloads the sentence/word splitter so TextBlob can analyze your text |

Let me know if you want to go even deeper into what punkt does with examples — or if you're curious about other NLTK tools!