



Sonstige Beteiligung

Verwendeter Code zur Analyse von Artikeln des Nachrichtensenders *n-tv*

Kai Herbst, Manuel Zeh, Henrik Popp

August 2024

Inhaltsverzeichnis

1	Einleitung	1
2	Datenerhebung	2
2.1	Sammlung und Speicherung der Nachrichtenartikel	2
2.2	Datenvorbereitung	7
3	Analysevorbereitung	9
3.1	Vorbereitung der Upload-Analyse	10
3.2	Vorbereitung zur Themenanalyse	11
3.3	Durchführung Sentiment-Analyse	12
4	Analyse der Daten	14
4.1	Analyse des Uploadverhaltens	14
4.2	Analyse der Sentiments	19
4.3	Topic-Analyse über Stichwörter	23
4.4	Topic-Analyse über Topic Modeling	24
5	Fazit	27

1 Einleitung

Im Rahmen des Moduls *Analyse semi- & unstrukturierter Daten* wurden die von *n-tv* veröffentlichten Artikel im Zeitraum des 17.04.2024 - 17.07.2024 gesammelt und nachfolgend analysiert. Das vorliegende Dokument enthält den Code zusammen mit dazugehörigen kurzen Beschreibungen, der zur Sammlung, Verarbeitung und Analyse der Daten verwendet wurde.

Der Code ist in die Abschnitte Datenerhebung, Datenvorbereitung, Upload-Zeiten-Analyse, Sentiment-Analyse und Topic Modeling unterteilt.

Jeglicher Code lässt sich ebenfalls in Form von Jupyter Notebooks im GitHub-Repository *n-tv Datenanalyse* finden.

2 Datenerhebung

Die folgenden Unterkapitel dokumentieren den Code, der zur Sammlung, Speicherung und Vorverarbeitung für die späteren Analysen verwendet wurde.

2.1 Sammlung und Speicherung der Nachrichtenartikel

Abbildung 1 zeigt den grundlegenden Ablauf, durch den die Nachrichtenartikel gespeichert werden. Zunächst wurde in über *AWS Eventbridge* ein Scheduler definiert, der täglich um 00:00 Uhr Lambda-Funktion startet, die wiederum auf die von *n-tv* angebotenen RSS-Schnittstellen zugreift. In den jeweiligen RSS-Feeds für jede Kategorie werden die zuletzt veröffentlichten Artikel mit Links und weiteren Metadaten aufgelistet. Über die Links werden die Artikel anschließend heruntergeladen und in einem *AWS S3-Bucket* gespeichert.

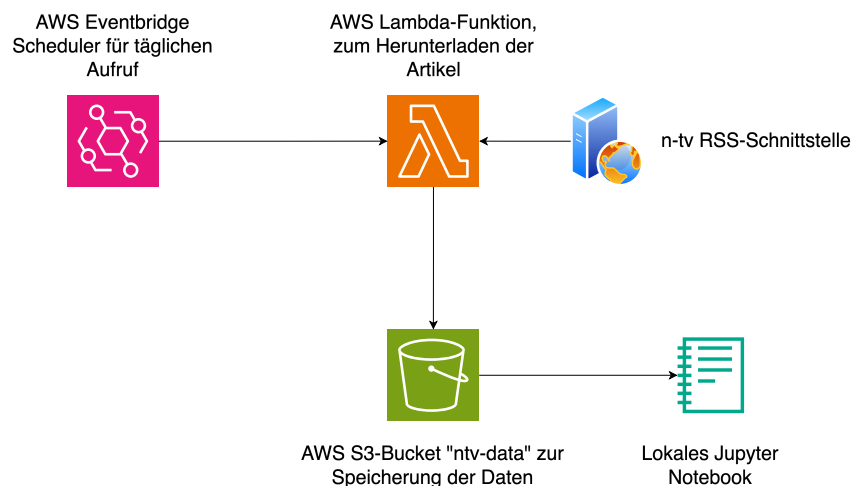


Abbildung 1: Datenpipeline zur Sammeln der *ntv*-Artikel

Die Lambda-Funktion zum Sammeln der Artikel besteht aus vier Dateien. Nachfolgend zu sehen ist die Hauptdatei mit der Einstiegsfunktion:

Listing 1: index.mjs

```
1 import helper from "./helper.mjs";
2 import categories from "./categories.mjs";
3 import S3Uploader from "./s3Helper.mjs";
4
5 export const handler = async (event, context) => {
6     const s3Uploader = new S3Uploader();
7     const bucket = "ntv-data";
8     const baseKey = helper.getYesterdaysDate().toISOString().split("T")[0];
9     for (let category of categories) {
```

```

10     let { links, rss } = await
        helper.getLinksOfYesterdaysArticelsAndRssObjectOf(category);
11     if (links.length == 0) {
12         continue;
13     }
14     let rssKey = baseKey + "/" + category + "/rss.json";
15     let rssJSONString = JSON.stringify(rss);
16     await s3Uploader.uploadContent(bucket, rssKey, rssJSONString);
17     for (let link of links) {
18         let articleKey = baseKey + "/" + category + "/" +
            helper.getFileNameFromLink(link);
19         let htmlContent = await helper.downloadContentFrom(link);
20         let cleanedHTML = helper.cleanHTMLContent(htmlContent);
21         await s3Uploader.uploadContent(bucket, articleKey, cleanedHTML);
22     }
23 }
24 return {
25     statusCode: 200,
26     body: JSON.stringify("Done!"),
27 };
28 };

```

Die Hauptfunktion (async handler) ruft verwendet wiederum die in den anderen drei Dateien *helper.mjs*, *categories.mjs* und *s3Helper.mjs* definierten Funktionen auf, die nachfolgend aufgelistet werden.

In der *categories.mjs* werden die abzurufenden Kategorien definiert:

Listing 2: categories.mjs

```

1 export default [
2     "dasbeste",
3     "politik",
4     "politik/US-wahl-2024",
5     "wirtschaft",
6     "boersenkurse",
7     "sport",
8     "panorama",
9     "leute",
10    "technik",
11    "ratgeber",
12    "wissen",
13    "auto",
14    "infografik",

```

```

15     "regionales",
16     "der_tag/alle_tage",
17     "meinung",
18 ];

```

Die *s3Helper.mjs*-Datei stellt Funktionen für den Zugriff auf die entsprechenden S3-Buckets bereit:

Listing 3: s3Helper.mjs

```

1  import { S3Client, PutObjectCommand } from "@aws-sdk/client-s3";
2
3  export default class S3Uploader {
4      constructor() {
5          this.s3Client = new S3Client({ region: "eu-central-1" });
6      }
7
8      async uploadContent(bucket, key, contentString) {
9          const command = new PutObjectCommand({
10              Bucket: bucket,
11              Key: key,
12              Body: contentString,
13          });
14
15          try {
16              console.log("Uploading", bucket + "/" + key, "...");
17              const response = await this.s3Client.send(command);
18              console.log("Finished!");
19          } catch (err) {
20              console.log(err);
21          }
22      }
23  }

```

In der *helper.mjs*-Datei werden schließlich noch Hilfsfunktionen definiert. Unter anderem Funktionen, die einen Großteil des HTML-Codes aus den Artikeln bereits im Vorfeld entfernt, um Speicherplatz zu sparen. Dazu zählen beispielsweise alle *Script*-Tags, da diese für die weitere Bearbeitung nicht benötigt werden. Durch diese Vorgehensweise wird viel Speicherplatz und damit auch Kosten eingespart.

Listing 4: helper.mjs

```

1  import { XMLParser } from "fast-xml-parser";
2

```

```

3  function buildRSSLinkFromCategory(category) {
4      let domain = "https://www.n-tv.de/";
5      let postfix = "/rss";
6      return domain + category + postfix;
7  }
8
9  function getYesterdaysDate() {
10     const date = new Date();
11     date.setHours(date.getHours() + 2);
12     date.setDate(date.getDate() - 1);
13     return date;
14 }
15
16 function dateWasYesterday(datestring) {
17     const yesterdaysDate = getYesterdaysDate();
18     const parsedDate = new Date(datestring);
19     return (
20         yesterdaysDate.getFullYear() == parsedDate.getFullYear() &&
21         yesterdaysDate.getMonth() == parsedDate.getMonth() &&
22         yesterdaysDate.getDate() == parsedDate.getDate()
23     );
24 }
25
26 function parseXMLString(xmlstring) {
27     const parser = new XMLParser();
28     return parser.parse(xmlstring);
29 }
30
31 function getRssItemsOfYesterday(rssObject) {
32     if (rssObject.rss.channel.item == undefined) {
33         return [];
34     }
35     let items = rssObject.rss.channel.item;
36     return items.filter((item) => dateWasYesterday(item.pubDate));
37 }
38
39 async function downloadRssXMLOfCategory(category) {
40     let url = buildRSSLinkFromCategory(category);
41     return await downloadContentFrom(url);
42 }
43
44 async function downloadContentFrom(url) {
45     let response = await fetch(url);

```

```

46     return await response.text();
47 }
48
49 function getFileNameFromLink(link) {
50     return link.split("/").at(-1);
51 }
52
53 function getLinksOfRssItems(rssItems) {
54     let links = [];
55     for (let item of rssItems) {
56         links.push(item.link);
57     }
58     return links;
59 }
60
61 function cleanHTMLContent(htmlString) {
62     let cleaned = htmlString.replace(
63         /<script\b[^\<]*(?:(!<\</script>>)<[^\<]*)*<\</script\s*>/gi, ""
64     );
65     cleaned = cleaned.replace(
66         /<nav\b[^\<]*(?:(!<\</script>>)<[^\<]*)*<\</nav\s*>/gi, ""
67     );
68     cleaned = cleaned.replace(
69         /<style\b[^\<]*(?:(!<\</script>>)<[^\<]*)*<\</style\s*>/gi, ""
70     );
71     cleaned =
72         cleaned.replace(/<link\b[^\<]*(?:(!<\</script>>)<[^\<]*)*<\</link\s*>/gi,
73             "");
74     return cleaned;
75 }
76
77 async function getLinksOfYesterdaysArticelsAndRssObjectOf(category) {
78     let rssXMLString = await downloadRssXMLOfCategory(category);
79     let rss = parseXMLString(rssXMLString);
80     let yesterdaysItems = getRssItemsOfYesterday(rss);
81     let links = getLinksOfRssItems(yesterdaysItems);
82     rss.rss.channel.item = yesterdaysItems;
83     rss.rss.channel.category = category;
84     return { links: links, rss: rss };
85 }
86
87 export default {
88     getYesterdaysDate,
89     getLinksOfYesterdaysArticelsAndRssObjectOf,
90     getFileNameFromLink,
91     downloadContentFrom,

```



```
88     cleanHTMLContent,  
89 };
```

2.2 Datenvorbereitung

Dieser Code dient dem Erstellen einer CSV-Datei, die anschließend als Grundlage für das weitere Vorgehen verwendet wird. Die Funktionen der einzelnen Methoden sollten dabei aus den Funktionsnamen hervorgehen. Grundsätzlich wird über die Ordner und Dateien des AWS lokal heruntergeladenen Buckets iteriert und die Artikel gesammelt. Mit Hilfe von *BeautifulSoup* werden Datum und Artikeltext extrahiert und gespeichert. Anschließend werden die gesammelten Daten in eine CSV-Datei geschrieben. Die CSV-Datei enthält die Spalten date (Datum), time (Uhrzeit), category (Kategorie), headline (Überschrift), filename (Dateiname) und text (Artikeltext).

Listing 5: Datengenerierung

```
1  import csv  
2  import os  
3  import re  
4  from datetime import datetime  
5  from typing import Dict, List  
6  
7  from bs4 import BeautifulSoup  
8  
9  
10 def get_content_of(article_path: str) -> str:  
11     result = ""  
12     with open(article_path, "r", encoding="utf-8") as article:  
13         result = article.read()  
14     if result != "":  
15         return result  
16     raise Exception("File not found")  
17  
18  
19 def extract_text_from_html_article(html_text: str) -> str:  
20     soup = BeautifulSoup(html_text, "html.parser")  
21     article_text = soup.find("div", class_="article__text")  
22     if article_text:  
23         soup = BeautifulSoup(article_text.text, "html.parser")  
24         for aside in soup.find_all("div", class_="article__aside"):  
25             aside.decompose()  
26         output = soup.text.strip()
```

```

27         output = re.sub(" +", " ", output)
28         output = output.replace("\n", "")
29         return output
30     raise Exception("No article text was found")
31
32
33 def extract_exact_date_from_html_article(html_text: str) -> str | None:
34     soup = BeautifulSoup(html_text, "html.parser")
35     date = soup.find("span", class_="article__date")
36     if date:
37         output = date.get_text()
38         return output
39     print("No date was found")
40     return None
41
42
43 def convert_filename_to_article_headline(article_filename: str) -> str:
44     headline = (" ").join(article_filename.split("-")[:-1])
45     return headline
46
47
48 def parse_ntv_datetime(date_string: str):
49     try:
50         datetime_obj = datetime.strptime(date_string, "%d.%m.%Y, %H:%M Uhr")
51         date = datetime_obj.date()
52         time = datetime_obj.time()
53         return date, time
54     except:
55         return None, None
56
57
58 def collect_article_contents_from(path: str) -> List[Dict[str, str]]:
59     collection = []
60     dates = os.listdir(path)
61     count_articles = 0
62     print()
63     for date in dates:
64         categories = os.listdir(f"{path}/{date}")
65         for category in categories:
66             article_list = os.listdir(f"{path}/{date}/{category}")
67             for article in article_list:
68                 if article != "rss.json" and article != ".ipynb_checkpoints":
69                     count_articles += 1

```

```

70         print("\r", end="")
71         print(f"Scanned articles: {count_articles} - scanning:
              {article}", end="")
72         article_content =
              get_content_of(f"{path}/{date}/{category}/{article}")
73         article_text = extract_text_from_html_article(article_content)
74         article_extracted_date =
              extract_exact_date_from_html_article(article_content)
75         article_date, article_time =
              parse_ntv_datetime(article_extracted_date)
76         collection.append(
77             {
78                 "date": article_date,
79                 "time": article_time,
80                 "category": category,
81                 "headline":
                      convert_filename_to_article_headline(article),
82                 "filename": article,
83                 "text": article_text,
84             }
85         )
86     return collection
87
88
89 def main():
90     content = collect_article_contents_from("ntv-data")
91     with open("2-1-data.csv", "w", encoding="utf-8", newline="") as output_file:
92         fc = csv.DictWriter(output_file, fieldnames=content[0].keys())
93         fc.writeheader()
94         fc.writerows(content)
95
96
97 if __name__ == "__main__":
98     main()

```

3 Analysevorbereitung

Bevor die Daten auf einzelne Aspekte hin analysiert werden, müssen die Daten noch weiter aufbereitet werden und die Sentiment-Analyse und Stichwortgenerierung durchgeführt werden.

3.1 Vorbereitung der Upload-Analyse

Für die Upload-Analyse werden die CSV-Datei um die Spalten upload-hour (Stunde des Uploads), weekday (Wochentag) und length (Artikellänge) erweitert.

Listing 6: Vorbereitung zur Uploadanalyse

```
1 import pandas as pd
2
3
4 def main():
5     articles_data = pd.read_csv("2-1-data.csv")
6     articles = pd.DataFrame(articles_data)
7     articles = articles.drop("filename", axis=1)
8     articles.head()
9
10    articles["upload"] = pd.to_datetime(articles["time"] + " " +
        articles["date"])
11    articles = articles.dropna()
12    articles.head()
13
14    articles["upload-hour"] = articles["upload"].dt.hour
15    articles["upload-hour"] = articles["upload-hour"].astype(int)
16    articles.head()
17
18    articles["weekday"] = articles["upload"].dt.dayofweek
19    wochentag_map = {0: "Monday", 1: "Tuesday", 2: "Wednesday", 3: "Thursday",
        4: "Friday", 5: "Saturday", 6: "Sunday"}
20    articles["weekday"] = articles["weekday"].map(wochentag_map)
21    articles.head()
22
23    articles["length"] = articles["text"].apply(len)
24    articles.head()
25
26    articles.to_csv("2-2-data-extended.csv", index=False)
27
28
29 if __name__ == "__main__":
30     main()
```

3.2 Vorbereitung zur Themenanalyse

Für den ersten Ansatz der Themenanalyse müssen Stichwörter zu jedem Artikel generiert werden. Dies geschieht mit Hilfe des OpenAI-Sprachmodells *GPT-4o mini*, auf das über die OpenAI-REST-API zugegriffen wird und das zu jedem Artikel zehn Stichwörter generiert. Die Stichwörter werden anschließend der CSV-Datei hinzugefügt:

Listing 7: Vorbereitung zur Stichwortanalyse

```
1 import pandas as pd
2 from openai import OpenAI
3
4
5 def create_keywords_for_articles(openai_client, articles_df, start=0, end=5):
6     print()
7     for i in range(start, end):
8         article_text = articles_df.iloc[i]["text"]
9         completion = openai_client.chat.completions.create(
10             model="gpt-4o-mini",
11             messages=[
12                 {
13                     "role": "system",
14                     "content": "Du fasst Nachrichtenartikel des
                                Nachrichtensenders NTV anhand von Stichpunkten
                                zusammenzufassen. Du gibst ausschliesslich die
                                Stichwoerter durch Kommata getrennt zurueck. Die
                                Stichwoerter sollen den Inhalt mit uebergeordneten
                                Kategorien wie Politik, Wirtschaft, Gesellschaft, Auto,
                                Sport, Unterhaltung und dergleichen beschreiben. Es
                                sollen exakt 10 Stichwoerter generiert werden.",
15                 },
16                 {
17                     "role": "user",
18                     "content": article_text,
19                 },
20             ],
21         )
22         keywords = completion.choices[0].message.content
23         articles_df.at[i, "keywords"] = keywords
24         print("\r", end="")
25         print(f"Generated Keywords for {i + 1} articles", end="")
26     return articles_df
27
28
```

```

29 def main():
30     openai_client = OpenAI()
31     articles_data = pd.read_csv("2-2-data-extended.csv")
32     articles = pd.DataFrame(articles_data)
33     articles["keywords"] = None
34     articles = create_keywords_for_articles(openai_client, articles, 0, 10678)
35     articles.to_csv("2-3-data-extended-keywords.csv", index=False)
36
37
38 if __name__ == "__main__":
39     main()

```

3.3 Durchführung Sentiment-Analyse

Um die Sentiments der Artikel genauer untersuchen zu können, müssen diese zunächst identifiziert werden. Dazu wird die CSV-Datei um die Spalten `sentiment_headline`, `sentiment_text`, `sentiment_prop_headline_positive`, `sentiment_prop_headline_neutral`, `sentiment_prop_headline_negative`, `sentiment_prop_text_positive`, `sentiment_prop_text_neutral`, `sentiment_prop_text_negative` erweitert.

Listing 8: Vorbereitung zur Sentiment-Untersuchung

```

1 import pandas as pd
2 from germansentiment import SentimentModel
3
4 sentiment_model = SentimentModel()
5
6
7 def sentiment_on_texts(articles_df):
8     length = len(articles_df)
9     sentiments = []
10    for index, row in articles_df.iterrows():
11        print(f"\rRunning sentiment analysis on article {index + 1} of
              {length}", end="")
12        text_class, text_probabilities =
            sentiment_model.predict_sentiment([row["text"]],
            output_probabilities=True)
13        sentiments.append((text_class, text_probabilities))
14    return sentiments
15
16

```

```

17 def main():
18     articles_data = pd.read_csv("2-3-data-extended-keywords.csv")
19     articles = pd.DataFrame(articles_data)
20
21     headlines = articles["headline"]
22     headlines_classes, headlines_probabilities =
        sentiment_model.predict_sentiment(headlines, output_probabilities=True)
23
24     articles = articles.assign(sentiment_headline=headlines_classes)
25     articles = articles.assign(sentiment_prob_headline=headlines_probabilities)
26
27     text_sentiments = sentiment_on_texts(articles)
28     texts_classes = [x[0][0] for x in text_sentiments]
29     texts_probabilities = [x[1][0] for x in text_sentiments]
30     articles["sentiment_text"] = texts_classes
31     articles["sentiment_prob_text"] = texts_probabilities
32     articles["sentiment_prob_headline_positive"] = articles.apply(
33         lambda row: row["sentiment_prob_headline"][0][1], axis=1
34     )
35     articles["sentiment_prob_headline_negative"] = articles.apply(
36         lambda row: row["sentiment_prob_headline"][1][1], axis=1
37     )
38     articles["sentiment_prob_headline_neutral"] = articles.apply(
39         lambda row: row["sentiment_prob_headline"][2][1], axis=1
40     )
41     articles["sentiment_prob_text_positive"] = articles.apply(lambda row:
        row["sentiment_prob_text"][0][1], axis=1)
42     articles["sentiment_prob_text_negative"] = articles.apply(lambda row:
        row["sentiment_prob_text"][1][1], axis=1)
43     articles["sentiment_prob_text_neutral"] = articles.apply(lambda row:
        row["sentiment_prob_text"][2][1], axis=1)
44     articles = articles.drop("sentiment_prob_headline", axis=1)
45     articles = articles.drop("sentiment_prob_text", axis=1)
46     articles.to_csv("3-4-data-extended-keywords-sentiment.csv", sep=",",
        encoding="utf-8", index=False)
47
48
49 if __name__ == "__main__":
50     main()

```

4 Analyse der Daten

4.1 Analyse des Uploadverhaltens

Im nachfolgenden sind die Funktionen, die zur Erstellung der Diagramme verwendet wurden. Diese sollten in einzelnen Zellen in einem Jupyter-Notebook ausgeführt werden, um die Ergebnisse direkt sehen zu können.

Listing 9: Analyse der Uploadzeiten

```
1 import pandas as pd
2 import seaborn as sns
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 articles_data = pd.read_csv('2-4-data-extended-keywords-sentiment.csv')
7 articles = pd.DataFrame(articles_data)
8
9 def create_line_chart_for_uploads_per_hour(articles):
10     uploads_per_hour = articles['upload-hour'].value_counts().sort_index()
11     plt.figure(figsize=(12, 6))
12     plt.plot(uploads_per_hour.index, uploads_per_hour.values, marker='o',
13             linestyle='-', color='b')
14     plt.title('Uploads per hour')
15     plt.xlabel('Hour of day')
16     plt.ylabel('Uploads')
17     plt.xticks(range(24))
18     plt.grid(True)
19     plt.show()
20 create_line_chart_for_uploads_per_hour(articles)
21
22 def create_line_chart_for_uploads_per_weekday(articles):
23     uploads_per_weekday = articles['weekday'].value_counts().reindex(['Monday',
24     'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'],
25     fill_value=0)
26     plt.figure(figsize=(12, 6))
27     plt.plot(uploads_per_weekday.index, uploads_per_weekday.values, marker='o',
28             linestyle='-', color='b')
29     plt.title('Anzahl der Uploads pro Wochentag')
30     plt.xlabel('Wochentag')
31     plt.ylabel('Anzahl der Uploads')
32     plt.grid(True)
33     plt.show()
```



```

31 create_line_chart_for_uploads_per_weekday(articles)
32
33
34 def create_heatmap_of_hour_per_weekday(articles):
35     heatmap_data = articles.groupby(['weekday', 'upload-hour'],
36                                     observed=False).size().unstack(fill_value=0).reindex([
37         'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
38         'Sunday'
39     ], fill_value=0)
40     plt.figure(figsize=(12, 5))
41     sns.heatmap(heatmap_data, annot=True, fmt='d', cbar=False, rasterized=True,
42                 cmap="crest")
43     plt.title('Uploads per weekday and hour')
44     plt.xlabel('Hour')
45     plt.ylabel('Weekday')
46     plt.show()
47 create_heatmap_of_hour_per_weekday(articles)
48
49 def create_bar_chart_for_median_by_hour_and_by_weekday(articles):
50     weekday_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
51                     'Saturday', 'Sunday']
52     articles['weekday'] = pd.Categorical(articles['weekday'],
53                                         categories=weekday_order, ordered=True)
54     median_by_hour = articles.groupby('upload-hour')['length'].median()
55     median_by_weekday = articles.groupby('weekday',
56                                         observed=False)['length'].median()
57     median_by_weekday_hour = articles.groupby(['weekday', 'upload-hour'],
58                                                observed=False)['length'].median().unstack()
59     fig, axs = plt.subplots(2, 1, figsize=(10, 8))
60     axs[0].bar(median_by_hour.index, median_by_hour.values)
61     axs[0].set_title('Median length by hour of upload')
62     axs[0].set_xlabel('Upload hour')
63     axs[0].set_ylabel('Median length')
64     axs[1].bar(median_by_weekday.index, median_by_weekday.values)
65     axs[1].set_title('Median length by weekday')
66     axs[1].set_xlabel('Weekday')
67     axs[1].set_ylabel('Median length')
68     axs[1].set_xticks(range(len(weekday_order)))
69     axs[1].set_xticklabels(weekday_order)
70     plt.tight_layout()
71     plt.show()
72 create_bar_chart_for_median_by_hour_and_by_weekday(articles)
73

```

```

67 def create_heatmap_of_article_length_by_hour_and_weekday(articles):
68     median_data = articles.groupby(['upload-hour', 'weekday'],
        observed=False)['length'].median().reset_index()
69     pivot_data = median_data.pivot_table(index='weekday',
        columns='upload-hour', values='length', aggfunc='median', observed=False)
70     plt.figure(figsize=(22, 10))
71     sns.heatmap(pivot_data, annot=True, rasterized=True, cmap="crest",
        fmt='.1f')
72     plt.title('Median of length after upload hour and weekday')
73     plt.xlabel('Hour')
74     plt.ylabel('Weekday')
75     plt.show()
76 create_heatmap_of_article_length_by_hour_and_weekday(articles)
77
78 def create_heatmap_plots_for_hours_per_category(articles):
79     upload_counts = articles.groupby(['category',
        'upload-hour']).size().unstack(fill_value=0)
80     for category in upload_counts.index:
81         plt.figure(figsize=(24, 1))
82         sns.heatmap(upload_counts.loc[[category]], annot=True, fmt='d',
            cbar=False, rasterized=True, cmap="crest")
83         plt.title(f'Uploads per hour for category: {category}')
84         plt.xlabel('hour')
85         plt.yticks([])
86         plt.show()
87 create_heatmap_plots_for_hours_per_category(articles)
88
89 def create_heatmap_plots_for_articles_per_hour_unified(articles):
90     heatmap_data = articles.groupby(['category',
        'upload-hour']).size().unstack(fill_value=0)
91     scaled_heatmap_data = heatmap_data.div(heatmap_data.sum(axis=1), axis=0) *
        100
92     plt.figure(figsize=(10, 6))
93     sns.heatmap(scaled_heatmap_data, annot=False, cbar=False, rasterized=True,
        cmap="crest")
94     plt.title('Uploads by category and hour')
95     plt.xlabel('Hour')
96     plt.ylabel('Category')
97     plt.show()
98 create_heatmap_plots_for_articles_per_hour_unified(articles)
99
100 def create_bar_plots_for_articles_per_weekday(articles):

```

```

101     upload_weekday_counts = articles.groupby(['category', 'weekday'],
        observed=False).size().unstack(fill_value=0)
102     upload_weekday_counts = upload_weekday_counts.reindex(columns=['Monday',
        'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
103     for category in upload_weekday_counts.index:
104         plt.figure(figsize=(10, 2))
105         sns.barplot(x=upload_weekday_counts.columns,
        y=upload_weekday_counts.loc[category].values)
106         plt.title(f'Upload behavior category: {category}')
107         plt.xlabel('Wochentag')
108         plt.ylabel('Anzahl der Artikel')
109         plt.xticks(rotation=45)
110         plt.show()
111     create_bar_plots_for_articles_per_weekday(articles)
112
113     def create_heatmap_plots_for_articles_per_weekday_unified(articles):
114         heatmap_data = articles.groupby(['category', 'weekday'],
        observed=False).size().unstack(fill_value=0)
115         scaled_heatmap_data = heatmap_data.div(heatmap_data.sum(axis=1), axis=0) *
        100
116         plt.figure(figsize=(10, 6))
117         sns.heatmap(scaled_heatmap_data, annot=False, cbar=False, rasterized=True,
        cmap="crest")
118         plt.title('Uploads per category and hour')
119         plt.xlabel('Weekday')
120         plt.ylabel('Category')
121         plt.show()
122     create_heatmap_plots_for_articles_per_weekday_unified(articles)
123
124     def create_heatmap_plots_for_articles_per_hour_per_weekday(articles):
125         categories = articles['category'].unique()
126         weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
        'Saturday', 'Sunday']
127         hours = np.arange(0, 24)
128         index = pd.MultiIndex.from_product([categories, weekdays, hours],
        names=['Category', 'Weekday', 'Hour'])
129         upload_category_weekday_hour_counts = articles.groupby(['category',
        'weekday', 'upload-hour'], observed=False).size().reindex(index,
        fill_value=0).unstack(level='Hour')
130         upload_category_weekday_hour_counts =
        upload_category_weekday_hour_counts.reindex(weekdays, level='Weekday')
131         for category in categories:
132             plt.figure(figsize=(14, 4))

```

```

133     sns.heatmap(upload_category_weekday_hour_counts.loc[category],
134                  annot=True, fmt='d', cbar=False, rasterized=True, cmap="crest")
135     plt.title(f'Upload behavior for category: {category}')
136     plt.xlabel('hour of day')
137     plt.ylabel('weekday')
138     plt.show()
139
140 create_heatmap_plots_for_articles_per_hour_per_weekday(articles)
141
142 def create_bar_chart_for_length_per_category(articles):
143     median_by_category =
144         articles.groupby('category')['length'].median().reset_index()
145     plt.figure(figsize=(18, 6))
146     sns.barplot(x='category', y='length', data=median_by_category)
147     plt.title('Median length of category')
148     plt.xlabel('Category')
149     plt.ylabel('Median')
150     plt.show()
151
152 create_bar_chart_for_length_per_category(articles)
153
154 def create_heatmap_for_length_per_category_by_hour_and_weekday(articles):
155     categories = articles['category'].unique()
156     for category in categories:
157         category_data = articles[articles['category'] == category]
158         median_data = category_data.groupby(['upload-hour', 'weekday'],
159                                             observed=False)['length'].median().reset_index()
160         pivot_data = median_data.pivot_table(index='weekday',
161                                             columns='upload-hour', values='length', aggfunc='median',
162                                             observed=False)
163         plt.figure(figsize=(23, 5))
164         sns.heatmap(pivot_data, annot=True, rasterized=True, cmap="crest",
165                     fmt='.0f')
166         plt.title(f'Median of length for category "{category}" by upload hour
167                    and weekday')
168         plt.xlabel('Hour')
169         plt.ylabel('Weekday')
170         plt.show()
171
172 create_heatmap_for_length_per_category_by_hour_and_weekday(articles)
173
174 def create_bar_charts_for_length_of_hour_per_weekday_per_category(articles):
175     categories = articles['category'].unique()
176     for category in categories:
177         category_data = articles[articles['category'] == category]

```

```

168     median_by_hour = category_data.groupby('upload-hour',
        observed=False)['length'].median().reset_index()
169     plt.figure(figsize=(10, 4))
170     sns.barplot(x='upload-hour', y='length', data=median_by_hour)
171     plt.title(f'Median of length per hour and category {category}')
172     plt.xlabel('upload hour')
173     plt.ylabel('Median of length')
174     plt.xticks(rotation=45)
175     plt.show()
176 create_bar_charts_for_length_of_hour_per_weekday_per_category(articles)

```

4.2 Analyse der Sentiments

Der im Anschluss aufgeführte Code enthält die Funktionen zur Generierung von Visualisierungen zu den Sentiments. Auch diese sollten vorzugsweise in einem Jupyter Notebook in einzelnen Zellen ausgeführt werden, um die Diagramme sehen zu können.

Listing 10: Analyse der Sentiments

```

1  import pandas as pd
2  import matplotlib.pyplot as plt
3  import numpy as np
4
5  articles_data = pd.read_csv('2-4-data-extended-keywords-sentiment.csv')
6  articles = pd.DataFrame(articles_data)
7  articles.head()
8
9  def create_sentiment_result_plot(df, column):
10     sentiment_count = []
11     for sentiment_class, count in df[column].value_counts().items():
12         sentiment_count.append( (sentiment_class, count) )
13     labels, values = zip(*sentiment_count)
14     plt.xlabel('Sentiment class')
15     plt.ylabel('Count')
16     plt.title(f'Sentiment analyse of {column}')
17     plt.bar(labels, values)
18     for i, value in enumerate(values):
19         print(i, value)
20         plt.text(i, value + 0.1, str(value), ha='center', va='bottom')
21     plt.show()
22 create_sentiment_result_plot(articles, "sentiment_headline")

```

```

23
24 create_sentiment_result_plot(articles, "sentiment_text")
25
26 def create_category_result_plot(df, column, relative=False):
27     categories = []
28     category_count = []
29     negatives = []
30     neutrals = []
31     positives = []
32     for category, count in articles['category'].value_counts().items():
33         negatives.append(articles[column][(articles.sentiment_text ==
34             'negative') & (articles.category == category)].count())
35         neutrals.append(articles[column][(articles.sentiment_text == 'neutral')
36             & (articles.category == category)].count())
37         positives.append(articles[column][(articles.sentiment_text ==
38             'positive') & (articles.category == category)].count())
39         categories.append(category)
40     if relative:
41         for i, category in enumerate(categories):
42             count = articles[column][(articles.category == category)].count()
43             negatives[i] = negatives[i] / count
44             positives[i] = positives[i] / count
45             neutrals[i] = neutrals[i] / count
46     ind = np.arange(len(categories))
47     fig, ax = plt.subplots()
48     p1 = ax.bar(ind, positives, label='Positiv', color='seagreen')
49     p2 = ax.bar(ind, neutrals, bottom=positives, label='Neutral',
50         color='cornflowerblue')
51     p3 = ax.bar(ind, negatives, bottom=[i+j for i,j in zip(positives,
52         neutrals)], label='Negativ', color='lightcoral')
53     ax.set_xlabel('Kategorien')
54     ax.set_ylabel('Anteile')
55     ax.set_xticks(ind)
56     ax.set_xticklabels(categories)
57     plt.xticks(rotation=90)
58     ax.legend()
59     plt.tight_layout()
60     plt.show()
61 create_category_result_plot(articles, 'sentiment_headline', relative=True)
62
63 create_category_result_plot(articles, 'sentiment_text', relative = True)
64
65 def create_bar_chart_for_count_of_negative_articles_per_weekday(articles):

```

```

61     negative_articles = articles[articles["sentiment_text"] == "negative"]
62     negative_count_by_category = negative_articles.groupby("category").size()
63     negative_count_by_category.plot(kind="bar", color="red", alpha=0.7)
64     plt.title('Count of negative articles per category')
65     plt.xlabel('category')
66     plt.ylabel('count')
67     plt.xticks(rotation=45)
68     plt.grid(axis='y')
69     plt.tight_layout()
70     plt.show()
71 create_bar_chart_for_count_of_negative_articles_per_weekday(articles)
72
73 def create_bar_chart_for_percentage_of_negative_articles_per_weekday(articles):
74     grouped =
75         articles.groupby("weekday")["sentiment_text"].value_counts(normalize=True).unstack()
76     grouped["negative_share"] = grouped["negative"] * 100
77     grouped["negative_share"].plot(kind="bar", color="red", alpha=0.6)
78     plt.title('Percentage of negative articles per weekday')
79     plt.xlabel('weekday')
80     plt.ylabel('percentage (%)')
81     plt.xticks(rotation=45)
82     plt.ylim(0, 100)
83     plt.grid(axis='y')
84     plt.tight_layout()
85     plt.show()
86 create_bar_chart_for_percentage_of_negative_articles_per_weekday(articles)
87
88 def create_bar_chart_for_percentage_of_neutral_articles_per_weekday(articles):
89     grouped =
90         articles.groupby("weekday")["sentiment_text"].value_counts(normalize=True).unstack()
91     grouped["neutral_share"] = grouped["neutral"] * 100
92     grouped["neutral_share"].plot(kind="bar", color="blue", alpha=0.6)
93     plt.title('Percentage of neutral articles per weekday')
94     plt.xlabel('weekday')
95     plt.ylabel('percentage (%)')
96     plt.xticks(rotation=45)
97     plt.ylim(0, 100)
98     plt.grid(axis='y')
99     plt.tight_layout()
100     plt.show()
101 create_bar_chart_for_percentage_of_neutral_articles_per_weekday(articles)
102
103 def create_bar_chart_for_negative_articles_per_weekday_for_politics(articles):

```

```

102     negative_politik_articles = articles[(articles["category"] == "politik") &
      (articles["sentiment_text"] == "negative")]
103     negative_count_by_weekday = negative_politik_articles.groupby("weekday",
      observed=False).size()
104     ordered_days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
105     negative_count_by_weekday = negative_count_by_weekday.reindex(ordered_days)
106     negative_count_by_weekday.plot(kind="bar", color="red", alpha=0.6)
107     plt.title('Anzahl der negativen Artikel pro Wochentag (Kategorie
      "politik")')
108     plt.xlabel('Wochentag')
109     plt.ylabel('Anzahl der negativen Artikel')
110     plt.xticks(rotation=45)
111     plt.grid(axis='y')
112     plt.tight_layout()
113     plt.show()
114 create_bar_chart_for_negative_articles_per_weekday_for_politics(articles)
115
116 def
      create_bar_chart_for_count_of_negative_articles_per_hour_of_politics(articles):
117     negative_politik_articles = articles[(articles["category"] == "politik") &
      (articles["sentiment_text"] == "negative")]
118     negative_count_by_hour = negative_politik_articles.groupby("upload-hour",
      observed=False).size()
119     all_hours = range(24)
120     negative_count_by_hour = negative_count_by_hour.reindex(all_hours,
      fill_value=0)
121     negative_count_by_hour.plot(kind="bar", color="red", alpha=0.7)
122     plt.title('Count of negative articles per hour of category "politik"')
123     plt.xlabel('hour')
124     plt.ylabel('count')
125     plt.xticks(rotation=0)
126     plt.grid(axis='y')
127     plt.tight_layout()
128     plt.show()
129 create_bar_chart_for_count_of_negative_articles_per_hour_of_politics(articles)
130
131 def
      create_bar_chart_for_relative_count_of_negative_articles_per_hour_of_politics(articles):
132     politik_articles = articles[articles["category"] == "politik"]
133     total_count_by_hour = politik_articles.groupby("upload-hour",
      observed=False).size()
134     negative_count_by_hour =
      politik_articles[politik_articles["sentiment_text"] ==

```



```

    "negative"].groupby("upload-hour").size()
135 all_hours = range(24)
136 total_count_by_hour = total_count_by_hour.reindex(all_hours, fill_value=0)
137 negative_count_by_hour = negative_count_by_hour.reindex(all_hours,
    fill_value=0)
138 negative_share_by_hour = (negative_count_by_hour /
    total_count_by_hour).fillna(0) * 100
139 negative_share_by_hour.plot(kind="bar", color="red", alpha=0.6)
140 plt.title('Anteil der negativen Artikel pro Stunde (Kategorie "politik")')
141 plt.xlabel('Stunde')
142 plt.ylabel('Anteil der negativen Artikel (%)')
143 plt.xticks(rotation=0)
144 plt.ylim(0, 100)
145 plt.grid(axis='y')
146 plt.tight_layout()
147 plt.show()
148 create_bar_chart_for_relative_count_of_negative_articles_per_hour_of_politics(articles)

```

4.3 Topic-Analyse über Stichwörter

Ziel der Analyse über die Stichwörter ist die Generierung einer Karte, die die Topics als Netzwerk zeigt. Dazu wird eine Graphdatei aus den Daten erstellt, die anschließend in Gephi importiert werden kann. Die Gephi-Datei ist ebenso dem Upload angefügt. Die Stichwörter werden dazu zunächst auseinandergenommen und in einem neuen Dataframe gespeichert. Anschließend werden über die Zeilen iteriert und Informationen über Häufigkeiten der einzelnen Stichwörter und Häufigkeit über Kombinationen von Stichwörtern gesammelt. Diese Informationen werden anschließend *networkx* übergeben, der daraus ein GraphML-Datei erzeugt, die in Gephi importiert werden kann.

Listing 11: Erstellung der Graph-Datei

```

1 import pandas as pd
2 import networkx as nx
3 import matplotlib.pyplot as plt
4 from collections import defaultdict
5
6 articles_data = pd.read_csv('2-4-data-extended-keywords-sentiment.csv')
7 articles = pd.DataFrame(articles_data)
8 articles.head()
9
10 keywords_df = articles['keywords'].str.split(',', expand=True)
11 keywords_df = keywords_df.apply(lambda x: x.str.strip())

```

```

12 keywords_df.head()
13
14 invalid_rows = keywords_df[keywords_df[10].notna()]
15
16 keywords_df = keywords_df.iloc[:, :10]
17 keywords_df.head()
18
19 G = nx.Graph()
20
21 def create_graphml_file_for_gephi(filename: str):
22     word_count = {}
23     for index, row in keywords_df.iterrows():
24         words = row.dropna().values
25         for word in words:
26             if word not in word_count:
27                 word_count[word] = 0
28             word_count[word] += 1
29     for index, row in keywords_df.iterrows():
30         words = row.dropna().values
31         for i, word1 in enumerate(words):
32             if word_count[word1] > 10:
33                 for j in range(i + 1, len(words)):
34                     word2 = words[j]
35                     if word1 != word2 and word_count[word2] > 10:
36                         edge = tuple(sorted([word1, word2]))
37                         if not G.has_edge(word1, word2):
38                             G.add_edge(word1, word2, weight=1)
39                     else:
40                         G.edges[edge]['weight'] += 1
41     nodes_to_remove = [node for node in G.nodes if word_count[node] <= 10]
42     G.remove_nodes_from(nodes_to_remove)
43     nx.write_graphml(G, f"{filename}")
44
45 create_graphml_file_for_gephi("ntv-topic-graph.graphml")

```

4.4 Topic-Analyse über Topic Modeling

Als zweiter Ansatz zur Untersuchung der Themen wurden *BERTopic* verwendet. Bertopic bietet neben der Extrahierung der Themen auch zahlreiche Funktionen zur Visualisierung. Wie in den anderen Abschnitten sollte auch der Code zur Visualisierung in einem Jupyter Notebook in einzelnen Zellen aufgerufen werden. So können die Ergebnisse direkt

betrachtet und interaktiv verwendet werden.

Listing 12: Erstellung der Graph-Datei

```
1 import pandas as pd
2 import nltk
3 import matplotlib.pyplot as plt
4 from bertopic import BERTopic
5 from sklearn.feature_extraction.text import CountVectorizer
6 from nltk.corpus import stopwords
7 from nltk.tokenize import word_tokenize
8 from nltk.stem import SnowballStemmer
9
10 articles_data = pd.read_csv('2-4-data-extended-keywords-sentiment.csv')
11 articles = pd.DataFrame(articles_data)
12 articles.head()
13
14 # Bevor Texte mit Algorithmen wie 'BERTopic' analysiert werden, sollten sie
15 # vorverarbeitet werden. Dazu zaehlt beispielsweise das Entfernen von
16 # Stoppwoertern, da diese ansonsten eventuell eigene Kategorien erhalten
17 # koennen. Dafuer wird die Bibliothek 'nltk' verwendet, die entsprechende
18 # Datensaeetze zur Verfuegung stellt. Ausserdem werden Woerter auf ihren
19 # Wortstamm reduziert, was die Textmenge reduziert und dabei hilft gleiche
20 # Woerter unabhaengig ihrer Konjunktion auch als solche zu erkennen.
21 stemmer = SnowballStemmer('german')
22 stop_words = set(stopwords.words('german'))
23
24 def preprocess_text_nltk(text):
25     text = text.lower()
26     tokens = word_tokenize(text, language='german')
27     tokens = [word for word in tokens if word.isalnum() and word not in
28               stop_words]
29     tokens = [stemmer.stem(word) for word in tokens]
30     return ' '.join(tokens)
31
32 articles['processed_text'] = articles['text'].apply(preprocess_text_nltk)
33 articles.head()
34
35 # Nun kann das Topic Modeling mit 'BERTopic' durchgefuehrt werden. Dazu wird ein
36 # neues 'topic_model' erstellt und diesem ueber die 'fit_transform'-Funktion die
37 # Dokumente bzw. die Artikeltexte uebergeben. Fuer jeden Artikel wird
38 # anschliessend im DataFrame die von Bertopic ermittelte Kategorie angefuegt.
39 documents = articles['processed_text'].tolist()
40 topic_model = BERTopic(language="multilingual")
```

```

40 topics, probs = topic_model.fit_transform(documents)
41 articles['topic'] = topics
42 articles.head()
43
44 # Zur Veranschaulich der Verteilung der Themen wird eine Funktion definiert:
45 def visualize_topic_distribution(articles, topic_column_name):
46     topic_counts = articles[topic_column_name].value_counts()
47     plt.figure(figsize=(30, 6))
48     topic_counts.plot(kind='bar', color='skyblue')
49     plt.xlabel('Topic')
50     plt.ylabel('Count of articles')
51     plt.title('Distribution of topics')
52     plt.xticks(ticks=range(len(topic_counts)), labels=[f'{i}' for i in
53                 topic_counts.index], rotation=45)
54     plt.show()
55
56 # Der nachfolgende Graph zeigt die Verteilung der Topics. Wie man sehen kann,
57 # faellt ein Grossteil der Artikel, ungefaehrt die Haelfe, unter die Kategorie
58 # '-1'. Dieser Kategorie ordnet 'BERTopic' alle 'Outlier' zu, also Ausreisser
59 # zu, die keiner Kategorie zugeordnet werden konnten.
60 visualize_topic_distribution(articles, "topic")
61
62 # Dieses Problem zeigt sich ebenfalls in der von 'BERTopic' generierten
63 # Darstellung. Alle grauen Punkte stellen die Ausreisser dar.
64 topic_model.visualize_documents(documents, hide_document_hover=True,
65                                hide_annotations=True)
66
67 # Ein weiteres Problem stellt die Menge an gefunden Topics mit 100+
68 # verschiedenen Themen dar. Fuer beide Probleme bietet 'BERTopic' jedoch
69 # Loesungen. Die Menge an Kategorien kann jedoch ueber entsprechende
70 # Funktionsargumente selbststaendig angepasst werden. Im Folgenden wird die
71 # Anzahl an Topics auf 70 gesetzt, um die Menge an Themen leicht einzugrenzen
72 # und Topics, die nur aeusserst selten auftreten nicht zu beruecksichtigen.
73 documents = articles['processed_text'].tolist()
74 topic_model = BERTopic(language="multilingual", nr_topics=70)
75 topics, probs = topic_model.fit_transform(documents)
76 articles['topic'] = topics
77 articles.head()
78
79 visualize_topic_distribution(articles, "topic")
80
81 topic_model.visualize_documents(documents, hide_document_hover=True,
82                                hide_annotations=True)

```

```

80
81 # Als naechstes sollen die Ausreisser noch eingegrenzt werden. 'BERTopic' stellt
82 # dafuer mehrere Strategien zur Verfuegung. Hier verwendet wird die automatische
83 # Reduzierung durch 'BERTopic' ueber die Funktion
      'topic_model.reduce_outliers(documents, topics)'.
84 # Diese werden in der neuen Spalte 'new_topics' gespeichert.
85 articles['new_topic'] = topic_model.reduce_outliers(documents, topics)
86
87 visualize_topic_distribution(articles, "new_topic")
88 # Wie zu sehen ist, konnte eine Vielzahl an Ausreissern andern Topics zugeordnet
89 # werden. Die erweiterten Kategorien koennen ebenso im Cluster-Diagramm
      angezeigt werden.
90 topic_model.visualize_documents(documents, hide_document_hover=True,
      hide_annotations=True)
91 # Auch diese Uebersicht stellt eine gute Visualisierung zu den
      Themenschwerpunkten von 'n-tv' dar.
92 # Eine weitere von 'BERTopic' zur Verfuegung gestellte Visualisierung stellt
      ebenfalls gut die Themen dar:
93 topic_model.visualize_topics()

```

5 Fazit

Dank moderner Algorithmen und zahlreicher Python-Packages lassen sich solche Analyseprojekte heutzutage auch mit relativ wenigen Vorkenntnissen im Themenbereich gut umsetzen. Gerade die *Matplotlib* stellt bei der Erstellung Diagrammen bzw. der Visualisierung von Ergebnissen eine große Hilfe dar. Dank der modernen Entwicklung im Bereich KI lassen sich auch nachträglich zu fremden Texten Stichwörter generieren, was bei der Erforschung der Themen und der Erstellung des Netzwerks stark geholfen hat. *BERTopic* hat sich ebenfalls als gute Wahl zur Erforschung der Themen erwiesen, da es gleichzeitig zahlreiche Visualisierungsmöglichkeiten bietet.