Scraping *derstandard.at*

TODO: talk about some legal stuff with scraping

First we define the function `get_standard_soup()`, which sends a request to derstandard, along with a cookie that derstandard checks if a user has accepted their DSGVO notice. If this cookie is not sent, a banner is displayed and the html is only partially loaded.

Secondly, we define `get_frontpage_articles()`, which expects a bs4 soup object of a frontpage. While derstandard no longer offers an archive, the frontpage articles of a given day can be conveniently accessed with the pattern `frontpage/y/m/d`. Each frontpage contains article sections which contain the (sub)heading, lead, number of comments, storylabels etc.

We will start by pulling the frontpage of december 22th 2023.

In [17]:
```python
from bs4 import BeautifulSoup
import requests

# fetch the html content of a derstandard.at page
def get_standard_soup(link):
    response = requests.get(link, cookies={'DSGVO_ZUSAGE_V1': 'true'})
    return BeautifulSoup(response.content, 'html.parser')

# generate a dictionary of articles with title as key and the bs4 element as value
def get_frontpage_articles(soup):
    articles_dict = {}
    articles = soup.select('div.chronological>section article')
    for article in articles:
        title_tag = article.find('a')
        if title_tag and title_tag.has_attr('title'):
            title = title_tag['title']
            articles_dict[title] = article
    return articles_dict

# Generate the articles dictionary for an arbitrary frontpage
soup = get_standard_soup('https://www.derstandard.at/frontpage/2023/12/22')
articles_dict = get_frontpage_articles(soup)

print(f'We have fetched {len(articles_dict)} articles\n')
```

```
We have fetched 137 articles
```

In the next step, lets look at the information we can get from those article sections on the frontpage. By inspecting the html, we have already identified various elements that we will use in the subsequent steps:

- title
- subtitle
- article type
- link
- datetime
- kicker (like an additional tag, not 100% about its meaning yet)
- postingcount
- storylabels

while playing with the data, we noticed that not every article contains storylabels. We will check this in the following step, as well as if every article tag has a type.

In [18]:
```python
# Function to analyze attributes of specified tags and their attributes
def analyze_tag_attributes(articles_dict):
    no_data_type = set()
    no_story_label = set()

    for title, article in articles_dict.items():
        # Check if every article tag has a data-type attribute - basically the type of the article
        if not article.has_attr('data-type'):
            no_data_type.add(title)
        # search for <div class="storylabels"> in articles - the story labels
        if not article.find('div', class_='storylabels'):
            no_story_label.add(title)

    return no_data_type, no_story_label

no_data_type, no_story_label = analyze_tag_attributes(articles_dict)
print(f'Number of articles without data-type attribute: {len(no_data_type)}')
print(f'Number of articles without storylabels: {len(no_story_label)}')
```

```
# get articles that have a story label
has_label = set(articles_dict.keys()).difference(no_story_label)
print(f'Number of articles with story attribute: {len(has_label)}')

# a lot of articles do not have a story label, maybe an interesting goal for machine learning
```

```
Number of articles without data-type attribute: 0
Number of articles without storylabels: 104
Number of articles with story attribute: 33
```

All articles have a data-type, but only a few articles have story attributes. This could be an interesting labeling task for our machine learning project later. Next, we print out the html of two articles to show the data we are interested in.

In [19]:
```
# example of an article without story label
print(f'No storylabel:\n{articles_dict[list(no_story_label)[0]]}\n')
print(80*'-')

# articles with story label
print(f'With storylabel:\n{articles_dict[list(has_label)[0]]}')
```

```
No storylabel:
<article class="fig" data-dg="p1-43" data-dt="7x2" data-mg="p1-43" data-mt="4x4" data-type="story">
<div class="teaser-inner">
<a href="/story/3000000200853/trump-uebte-laut-medienbericht-druck-auf-wahlpruefer-in-michigan-aus" title="Trump übte laut
Medienbericht Druck auf Wahlprüfer in Michigan aus">
<figure data-type="image">
<picture>
<source data-lazy-srcset="https://i.ds.at/V3WkIw/c:1200:800:fp:0.500:0.500/rs:fill:280:187/g:fp:0.54:0.29/plain/lido-image
s/2023/12/22/3f9fca60-5573-4475-8e22-0cd35d00484b.jpeg" media="(min-width: 960px)"/>
<source data-lazy-srcset="https://i.ds.at/KnrHlA/c:1200:800:fp:0.500:0.500/rs:fill:750:375/g:fp:0.54:0.29/plain/lido-image
s/2023/12/22/3f9fca60-5573-4475-8e22-0cd35d00484b.jpeg" media="(max-width: 959px)"/>
<img alt="Election_2024-President-New_Mexico_69156" data-lazy-src="https://i.ds.at/WDo_zA/rs:fill:600:400/plain/lido-image
s/2023/12/22/3f9fca60-5573-4475-8e22-0cd35d00484b.jpeg" referrerpolicy="unsafe-url"/>
</picture>
</figure>
<header>
<time datetime="2023-12-22T14:38
">14:38
</time>
<p class="teaser-kicker">USA</p>
<div class="teaser-postingcount">25 <span>Postings</span>
</div>
<h1 class="teaser-title">Trump übte laut Medienbericht Druck auf Wahlprüfer in Michigan aus </h1>
<p class="teaser-subtitle">Der <span class="hyphenate">Ex-US-Präsident</span> soll versucht haben, zwei Wahlprüfer von ein
er Bestätigung der Wahlresultate abzuhalten </p>
</header>
</a>
</div>
</article>

--------------------------------------------------------------------------------
With storylabel:
<article class="fig" data-dg="p1-132" data-dt="7x2" data-mg="p1-132" data-mt="4x4" data-type="story">
<div class="teaser-inner">
<a href="/story/3000000200661/baerige-weihnachten-in-schoenbrunn" title="Bärige Weihnachten in Schönbrunn">
<figure data-type="image">
<picture>
<source data-lazy-srcset="https://i.ds.at/oT-PeQ/c:1620:1080:fp:0.500:0.500/rs:fill:280:187/plain/lido-images/2023/12/21/0
dc7ec6e-fd95-4be8-94ca-801ed067dfb6.jpeg" media="(min-width: 960px)"/>
<source data-lazy-srcset="https://i.ds.at/xmQSZA/c:1620:1080:fp:0.500:0.500/rs:fill:750:375/plain/lido-images/2023/12/21/0
dc7ec6e-fd95-4be8-94ca-801ed067dfb6.jpeg" media="(max-width: 959px)"/>
<img alt="Bärige Weihnachten in Schönbrunn" data-lazy-src="https://i.ds.at/bhX0lA/rs:fill:600:400/plain/lido-images/2023/1
2/21/0dc7ec6e-fd95-4be8-94ca-801ed067dfb6.jpeg" referrerpolicy="unsafe-url"/>
</picture>
<time class="duration">01:03</time>
</figure>
<header>
<time datetime="2023-12-22T06:00
">06:00
</time>
<p class="teaser-kicker">Weihnachten</p>
<div class="teaser-postingcount">1 <span>Posting</span>
</div>
<h1 class="teaser-title">Bärige Weihnachten in Schönbrunn </h1>
<p class="teaser-subtitle">Für die Schönbrunner Brillenbären hat es bereits vor den Feiertagen eine weihnachtliche Überras
chung vom <span class="hyphenate">Tierpflegerteam</span> gegeben: einen Christbaum geschmückt unter anderem mit <span clas
s="hyphenate">Karottenlametta</span> und <span class="hyphenate">Paprikaringerln</span> </p>
<time class="duration">01:03</time>
<div class="storylabels">
<span data-label-category="indepth" data-label-name="Video">Video</span>
</div>
</header>
</a>
</div>
</article>
```

This is the information one can get from the frontpage. Later we will also follow the links and scrape additional data from the articles, but lets first focus on getting a dataset just based on the information that can be attained from the frontpage.

To this end, we define `extract_article_data()`, which uses the dictionary following the pattern `article_title: article_section_soup`. From the html (soup), we will extract and clean:

- title
- teaser-subtitle
- link
- time
- teaser-kicker
- n_posts
- storylabels

In [20]:
```python
# Function to extract specific data from each article
def extract_article_data(articles_dict):
    HOST = 'https://www.derstandard.at'
    article_data = []

    for title, article in articles_dict.items():
        data = {
            'title': title,
            'teaser-subtitle': None,
            'link': None,
            'time': None,
            'teaser-kicker': None,
            'n_posts': None,
            'storylabels': None
        }

        # most links are relative, so we need to add the host
        link = article.find('a')['href']
        if not link.startswith(HOST):
            link = HOST + link
        data['link'] = link

        # for live articles, there is a second time tag with the duration of the live post
        # however, we only care about the time of publication here
        time = [tag for tag in article.find_all('time') if 'datetime' in tag.attrs][0]
        data['time'] = time['datetime'].rstrip('\r\n')

        # if there are no comments, the string is empty so set it to 0
        n_posts = article.find('div', 'teaser-postingcount')
        try: data['n_posts'] = int(n_posts.get_text(strip=True).rstrip('Posting').replace('.', ''))
        except: data['n_posts'] = 0

        # Extracting other specified tags
        for tag, class_name in [('p', 'teaser-kicker'),
                                ('p', 'teaser-subtitle'),
                                ('div', 'storylabels')]:
            found_tag = article.find(tag, class_=class_name)
            if found_tag:
                data[class_name] = found_tag.get_text(strip=True)

        article_data.append(data)

    return article_data

article_data = extract_article_data(articles_dict)
# last 5 articles, of which some have a story label
article_data[-5:]
```

```
Out[20]: [{'title': 'Jesus-Geburt unter Palmen',
           'teaser-subtitle': 'Auch der Koran erzählt über die Geburt von Jesus Christus. Nicht als Sohn Gottes, aber als Sohn Ma
          rias, die als Frau im Koran eine besondere Stellung einnimt',
           'link': 'https://www.derstandard.at/story/3000000200744/jesus-geburt-unter-palmen',
           'time': '2023-12-22T06:00',
           'teaser-kicker': 'Wussten Sie schon?',
           'n_posts': 1,
           'storylabels': None},
          {'title': '"Aquaman and the Lost Kingdom" scheitert an versuchter Schadensbegrenzung',
           'teaser-subtitle': 'Der Erfolg derSuperheldenfilmeleidet immer öfter unter den privaten Problemen ihrerHauptdarstelle
          r.Der Fortsetzung von "Aquaman" droht auch deshalb ein Flop',
           'link': 'https://www.derstandard.at/story/3000000200724/aquaman-and-the-lost-kingdom-scheitert-an-versuchter-schadensb
          egrenzung',
           'time': '2023-12-22T06:00',
           'teaser-kicker': 'Im Kino',
           'n_posts': 242,
           'storylabels': None},
          {'title': 'One-Man-Show mit fatalen Folgen für die Demokratie in Serbien',
           'teaser-subtitle': 'Die Wahlen in Serbien waren eine Farce. Die Europäische Union hat bisher auf einen pragmatischen K
          uschelkurs gesetzt. Damit könnte es jetzt aber vorbei sein. Für eine härtere Gangart wäre es auch höchste Zeit',
           'link': 'https://www.derstandard.at/story/3000000200698/one-man-show-mit-fatalen-folgen-fuer-die-demokratie-in-serbie
          n',
           'time': '2023-12-22T06:00',
           'teaser-kicker': 'Vedran Džihić',
           'n_posts': 112,
           'storylabels': 'Kommentar der anderen'},
          {'title': 'David Alaba zum zehnten Mal zu Österreichs Fußballer des Jahres gekürt',
           'teaser-subtitle': 'Zehn von zwölf Trainern wählten den derzeit verletztenReal-Verteidigerauf Rang eins. Für den Wiene
          r ist es der vierte Erfolg in Serie',
           'link': 'https://www.derstandard.at/story/3000000200745/david-alaba-zum-10-mal-fu223baller-des-jahres-riesige-ehre',
           'time': '2023-12-22T05:46',
           'teaser-kicker': 'Fußball',
           'n_posts': 33,
           'storylabels': None},
          {'title': 'Kreuzworträtsel F 10571',
           'teaser-subtitle': 'Täglich neu, exklusiv fürSmart-Abonnent:innen:Das kniffligephoenixen-Rätseldes STANDARD',
           'link': 'https://www.derstandard.at/story/3000000199163/kreuzwortraetsel-f-10571',
           'time': '2023-12-22T00:01',
           'teaser-kicker': 'Kreuzworträtsel',
           'n_posts': 4,
           'storylabels': 'Spiel'}]
```

The various attributes will be analyzed once we convert our data to a dataframe.

But before we start scraping like mad, lets check robots.txt such that we can comply with derstandards scraping policies.

```
In [21]: print(get_standard_soup('https://www.derstandard.at/robots.txt'))
```

```
User-agent: *

Disallow: /profil/

Sitemap: https://www.derstandard.at/sitemaps/news.xml
Sitemap: https://www.derstandard.at/sitemaps/sitemap.xml

Crawl-delay: 1
C:\Users\Paul\AppData\Local\Temp\ipykernel_24276\4133154167.py:7: MarkupResemblesLocatorWarning: The input looks more like
a filename than markup. You may want to open this file and pass the filehandle into Beautiful Soup.
  return BeautifulSoup(response.content, 'html.parser')
```

`Craw-delay: 1`, so lets be nice and wait 1 second between requests. Then we'll scrape the frontpage of every day in 2023 until the 20th of december and save the data as a csv.

Caution: you might not want to run this cell, as it takes about ~45 minutes to run. The data has already been extracted once and has been saved to `data/derstandard_frontpage_data.csv`.

```
In [28]: from datetime import datetime, timedelta
         from time import sleep
         from tqdm import tqdm


         def scrape_frontpage(start_date: str, end_date: str, logging=False):
             # Validate that dates follow the pattern YYYY-MM-DD
             try:
                 start = datetime.strptime(start_date, '%Y-%m-%d')
                 end = datetime.strptime(end_date, '%Y-%m-%d')
             except ValueError:
                 print("Invalid date format. Please use YYYY-MM-DD.")
                 return
```

```
    data = []
    # all dates between start and end (inclusive)
    delta = end - start
    for i in tqdm(range(delta.days + 1)):
        # generate link for each day
        day = start + timedelta(days=i)
        date = day.strftime('%Y/%m/%d')
        link = f'https://www.derstandard.at/frontpage/{date}'
        # make a request to the link and extract the data
        article_dict = get_frontpage_articles(get_standard_soup(link))
        articles = extract_article_data(article_dict)
        if logging:
            print(f'Fetched {len(articles)} articles from {date}')
        data += articles
        # wait almost a second before next request, our data processing takes a bit of time as well
        sleep(0.8)

    return data

# scrape the data for 4 years
data = scrape_frontpage('2019-12-22', '2023-12-22')
```

```
100%|██████████| 1462/1462 [46:44<00:00,  1.92s/it]
```

Okay, this cell took a while to run obviously, but we finally have our precious data. Lets convert it to a dataframe and see what we have.

In [29]:
```
import pandas as pd

df = pd.DataFrame(data)
df.columns = df.columns.str.replace('teaser-', '')
df.rename(columns={'time': 'datetime'}, inplace=True)
df
```

Out[29]:

	title	subtitle	link	datetime	kicker
0	Real Madrid stolpert mit Aluminiumpech im Tite...	Die Königlichen können Bilbao daheim nicht bes...	https://www.derstandard.at/story/2000112599363...	2019-12-22T23:44	Primera Division
1	Bolivien weist venezolanische Diplomaten aus	InterimspräsidentinJeanine Áñez wirft denBotsc...	https://www.derstandard.at/story/2000112598924...	2019-12-22T22:50	Übergangsregierung
2	Erdoğan warnt vor neuer Flüchtlingswelle aus S...	Türkischer Präsident: "80.000 Menschen Richtun...	https://www.derstandard.at/story/2000112598130...	2019-12-22T21:43	Bürgerkrieg
3	Massenkarambolage mit 63 Fahrzeugen in Virginia	Autos stießen auf vereister Brücke zusammen	https://www.derstandard.at/story/2000112597972...	2019-12-22T21:29	Weihnachtsverkehr
4	Salzburg schlägt Caps, Meister KAC mit vierter...	Die Bullen sind damit der Gewinner der Runde: ...	https://www.derstandard.at/story/2000112595206...	2019-12-22T20:54	Eishockey
...	...	...	...	...	...
182102	Wer braucht die Kirche?	Dass sich die Kirche nach soschwerwiegendenVer...	https://www.derstandard.at/story/3000000200743...	2023-12-22T06:00	Dominik Straub
182103	Sonderregelung verlängert: Mehr als 1.000 Ärzt...	Der"Pandemieparagraf"im Ärztegesetz hat mehr a...	https://www.derstandard.at/story/3000000200621...	2023-12-22T06:00	Pandemieparagraf
182104	Stadtforscher: "Architektur ist Teil unserer W...	Jetzt anhören: In Zukunft müssen Städte wieder...	https://www.derstandard.at/story/3000000200499...	2023-12-22T06:00	Edition Zukunft
182105	David Alaba zum zehnten Mal zu Österreichs Fuß...	Zehn von zwölf Trainern wählten den derzeit ve...	https://www.derstandard.at/story/3000000200745...	2023-12-22T05:46	Fußball
182106	Kreuzworträtsel F 10571	Täglich neu, exklusiv fürSmart-Abonnent:innen:...	https://www.derstandard.at/story/3000000199163...	2023-12-22T00:01	Kreuzworträtsel

182107 rows × 7 columns

over 182 thousand rows, this should give us plenty data to analyze!

Lets save it to a csv, totalling 57mb

In [30]:
```
df.to_csv('data/4yrs_derstandard_frontpage_data.csv', index=False)
```

Data Processing Notebook

EDA

Lets load the frontpage data of 4 years going from december 22. 2019 to december 22. 2023.

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('./data/4yrs_derstandard_frontpage_data.csv')

# prepare time data
df['datetime'] = pd.to_datetime(df['datetime'])
df.head()
```

Out[1]:

	title	subtitle	link	datetime	kicker	n_post
0	Real Madrid stolpert mit Aluminiumpech im Tite...	Die Königlichen können Bilbao daheim nicht bes...	https://www.derstandard.at/story/2000112599363...	2019-12-22 23:44:00	Primera Division	3
1	Bolivien weist venezolanische Diplomaten aus	InterimspräsidentinJeanine Áñez wirft denBotsc...	https://www.derstandard.at/story/2000112598924...	2019-12-22 22:50:00	Übergangsregierung	1
2	Erdoğan warnt vor neuer Flüchtlingswelle aus S...	Türkischer Präsident: "80.000 Menschen Richtun...	https://www.derstandard.at/story/2000112598130...	2019-12-22 21:43:00	Bürgerkrieg	10
3	Massenkarambolage mit 63 Fahrzeugen in Virginia	Autos stießen auf vereister Brücke zusammen	https://www.derstandard.at/story/2000112597972...	2019-12-22 21:29:00	Weihnachtsverkehr	3
4	Salzburg schlägt Caps, Meister KAC mit vierter...	Die Bullen sind damit der Gewinner der Runde: ...	https://www.derstandard.at/story/2000112595206...	2019-12-22 20:54:00	Eishockey	

182 thousand rows, lets inspect the data a bit

In [2]:
```python
print('All variables:\n', df.dtypes, '\n')
print('N_posts percentiles:\n', df['n_posts'].describe(), '\n')
print('n_posts 0-values:\n', len(df[df['n_posts']==0]))
```

```
All variables:
 title                  object
subtitle               object
link                   object
datetime       datetime64[ns]
kicker                 object
n_posts                 int64
storylabels            object
dtype: object

N_posts percentiles:
 count    1.821070e+05
mean     3.342677e+02
std      7.621049e+03
min      0.000000e+00
25%      1.400000e+01
50%      7.200000e+01
75%      2.490000e+02
max      3.000166e+06
Name: n_posts, dtype: float64

n_posts 0-values:
 10417
```

Data correctness

While manually analyzing a sample of the over 10.000 articles with 0 posts, We noticed that there were a lot of articles that generated 0 posts. In the scraping script, we gathered the post count in the function `extract_article_data()`, which searched for the span containing the post count. Sometimes 0 posts is explicitely mentioned, sometimes the span is empty.

We analyzed a sample our data on articles with 0 posts and found that the information we could get from the frontpage seems to be correct and that the articles with 0 posts did indeed generate 0 posts. With one exception - the data on Live tickers was sometimes incorrect and indicated 0 posts when there were many posts in actuality.

The cause for this mismatch is unclear to us, but we were able to find a pattern which helps us find those wrongly labeled discussion forums.

In [3]:
```
print('posts which contain ticker in the title:')
print(df[df['title'].str.contains('Ticker')]['storylabels'].value_counts(dropna=False), '\n')
print('posts which contain live in the title:')
print(df[df['title'].str.contains('Live')]['storylabels'].value_counts(dropna=False), '\n')
```

```
posts which contain ticker in the title:
storylabels
NachleseLiveticker     271
LivetickerNachlese      30
LiveLiveticker           6
NaN                      6
NachleseLivebericht      5
Forum+Nachlese           3
LivetickerLive           2
LiveberichtNachlese      2
NachleseForum+           1
LiveForum+               1
LiveLivebericht          1
Forum+Live               1
Name: count, dtype: int64

posts which contain live in the title:
storylabels
NaN                    502
NachleseLiveticker      64
NachleseLivebericht     12
LivetickerNachlese       9
Kolumne                  7
Interview                5
LiveberichtNachlese      4
Video                    3
Analyse                  3
Blog                     3
Podcast                  2
Porträt                  2
Bericht                  2
Ansichtssache            2
Essay                    1
LiveLiveticker           1
Kopf des Tages           1
Kommentar                1
KolumneVideo             1
LivetickerLive           1
Rezension                1
Reportage                1
Name: count, dtype: int64
```

While there are still some NA values in the set of posts that contain the words Live or Ticker, closer inspection has shown that those articles were not actually discussion forums/live tickers but rather articles like `Livestreams, Ticker und Spielpläne: Wie man die EM 2021 online verfolgt`.

But as it turns out, livetickers seem to be among the rare data points that have all been assigned a storylabel. We will thus first identify the storylabels that live articles have been tagged with.

In [4]:
```
# all the Live storylabels (I checked, they are all capitalized)
live_labels = df[(df['storylabels'].str.contains('Live')) & (~df['storylabels'].isna())]['storylabels'].value_counts()
live_labels = list(live_labels.index)
print('All Live storylabels:\n*', '\n* '.join(live_labels))
```

```
All Live storylabels:
* NachleseLivebericht
* NachleseLiveticker
* LiveberichtNachlese
* LivetickerNachlese
* LiveForum+
* LiveLiveticker
* Forum+Live
* LiveLivebericht
* LivetickerLive
* LiveberichtLive
* LiveberichtVorschau
* VorschauLiveticker
* Liveticker
* LiveberichtVideo
* Livebericht
```

We will now use this list of storylabels and simply drop all articles tagged with them that contain 0 posts. Of course this approach is just a heuristic and there is a very good chance there are still wrongly labeled datapoints, but for the purposes of this project we will leave it at that. Those >900 articles are very likely to be wrongly labeled, and making another request to fetch the actual comment count would be too time-consuming for the purposes of this project.

In [5]:
```
forums = df[df['storylabels'].isin(live_labels)]
no_post_forums = forums[forums['n_posts'] == 0]
print(f'Forums wrongly labeled as 0 posts: {len(no_post_forums)} out of {len(forums)}')

# drop all rows from the df that are in no_post_forums
df.drop(no_post_forums.index, inplace=True)
print(f'Dropped {len(no_post_forums)} rows. {len(df)} rows remaining.')
```

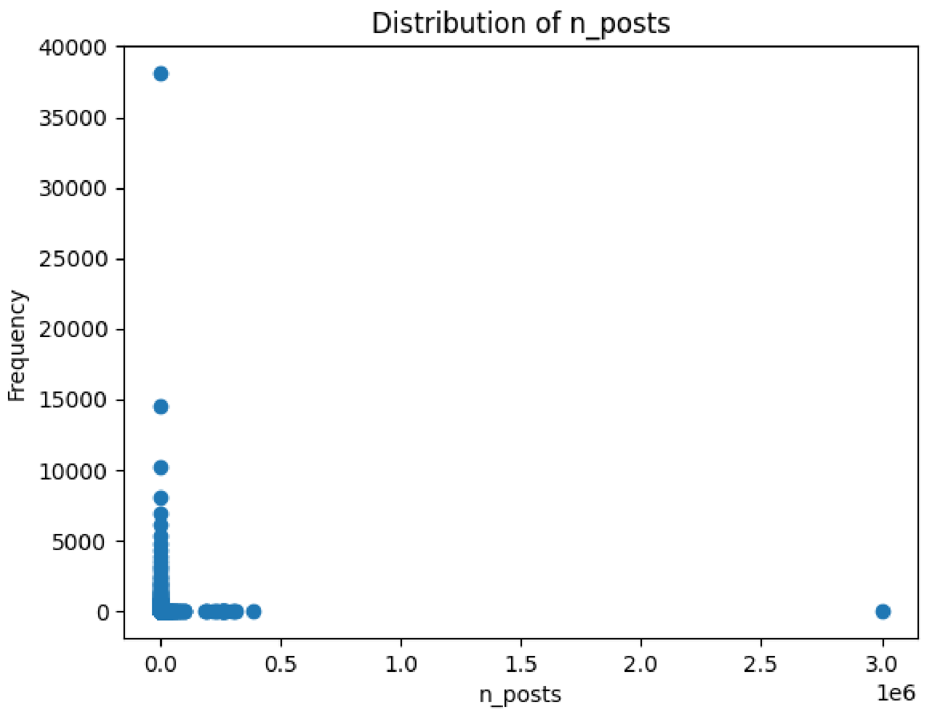
```
Forums wrongly labeled as 0 posts: 907 out of 2422
Dropped 907 rows. 181200 rows remaining.
```

Post count distribution

Next we will look at the distribution of our intended target variable, `n_posts`. For a first look, we bin the articles in ranges of 10. The following scatterplot shows the number of articles that generate between n and n+10 posts on the y axis and the lower bound of the bin (n) of posts on the x-axis.

In [6]:
```
# print n_posts distribution
max_posts = np.ceil(df['n_posts'].max() / 10) * 10 # rounded to nearest multiple of 10
bins = np.arange(0, max_posts + 10, 10) # steps of 10
binned_table = df.groupby(pd.cut(df['n_posts'], bins, include_lowest=True), observed=False).size().reset_index(name='coun

binned_table = binned_table[binned_table['counts'] != 0]
binned_table['bin_middle'] = binned_table['n_posts'].apply(lambda x: x.mid)

# Scatterplot of table
plot = plt.scatter(binned_table['bin_middle'], binned_table['counts'])
plt.title('Distribution of n_posts')
plt.xlabel('n_posts')
plt.ylabel('Frequency')
```

Out[6]:
```
Text(0, 0.5, 'Frequency')
```



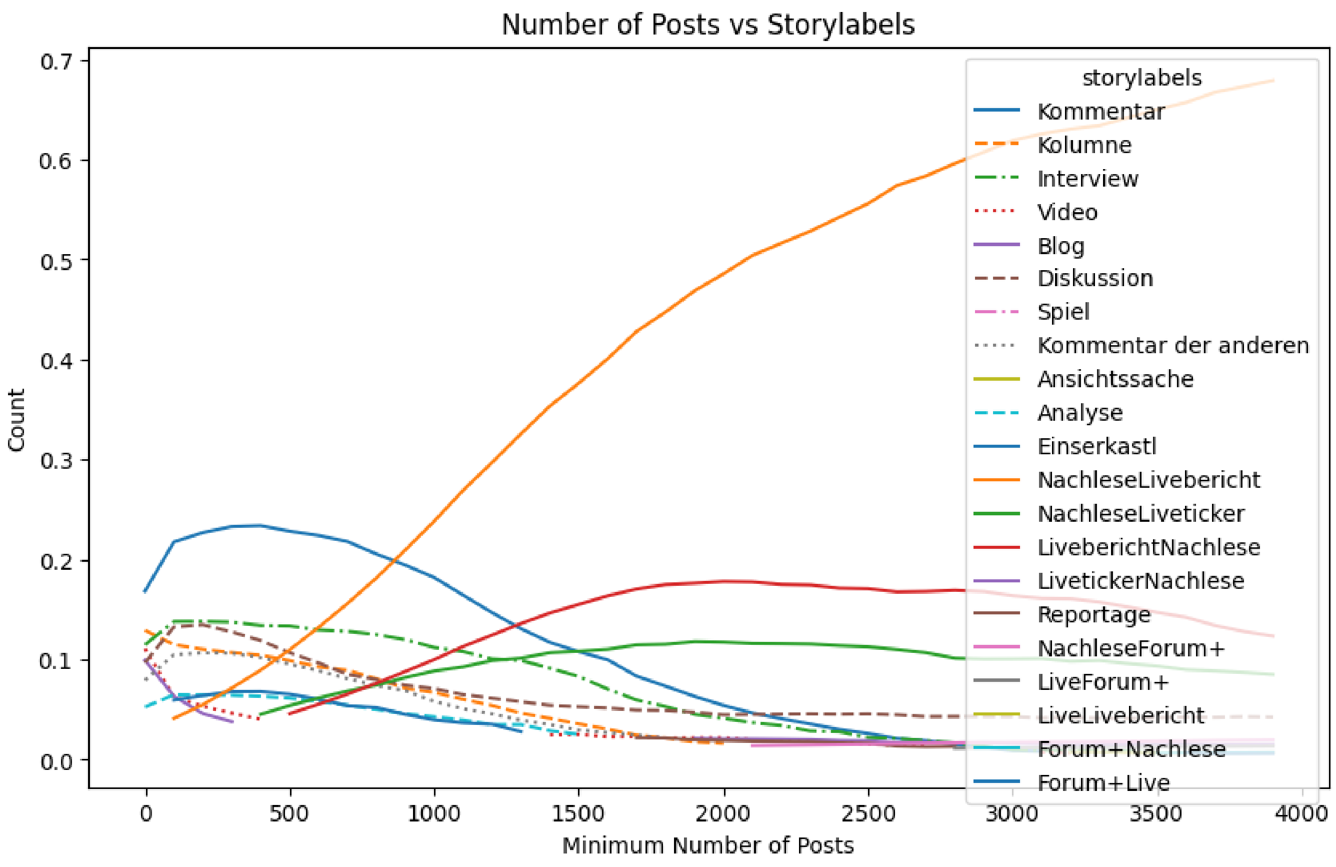
This is a very zoomed out look. Before we focus this scatterplot, we'll check out the outlier that generated 3 million posts:

```
In [7]: print('Articles with the most posts')
        print(df.sort_values(by='n_posts', ascending=False)[['title', 'n_posts']].head(10))
```

```
Articles with the most posts
                                                  title  n_posts
108017                Off-Topic-Ticker mit TickerOfLove  3000166
108018  Off-Topic-Ticker mit Ticker of Love and Laughter   389259
102886                         Seuchenticker-Basislager   314567
109509                       Seuchenticker-Basislager 3   308522
170225                             Ticker-Basislager 13   300140
149054                   (Seuchen)ticker-Basislager 10   275232
135239                       Seuchenticker-Basislager 7   270924
115763                       Seuchenticker-Basislager 4   268672
157329                             Ticker-Basislager 11   263017
149052                       Seuchenticker-Basislager 9   262740
```

It is the love forum that generated 3 million posts, after that, the next post also deals with topics of love but has 400 thousand posts. The rest of the top 10 seem to also all be forum posts. Lets visualize the relative share of the different storylabels when the post count increases (keeping in mind that only a minority of posts has a storylabel):

```
In [8]: r = range(0, 4_000, 100)
        label_tracker = []
        for min_posts in r:
            d = df[df['n_posts'] > min_posts].storylabels.value_counts()[:10]
            label_tracker.append(d / d.sum())

        df_label_tracker = pd.concat(label_tracker, axis=1).T
        df_label_tracker.reset_index(drop=True, inplace=True)
        df_label_tracker['min_posts'] = r
        df_label_tracker.set_index('min_posts', inplace=True)
        styles = ['-', '--', '-.', ':', '-', '--', '-.', ':', '-', '--']
        df_label_tracker.plot(kind='line', style=styles, figsize=(10, 6))

        plt.title('Number of Posts vs Storylabels')
        plt.xlabel('Minimum Number of Posts')
        plt.ylabel('Count')
        plt.show()
```

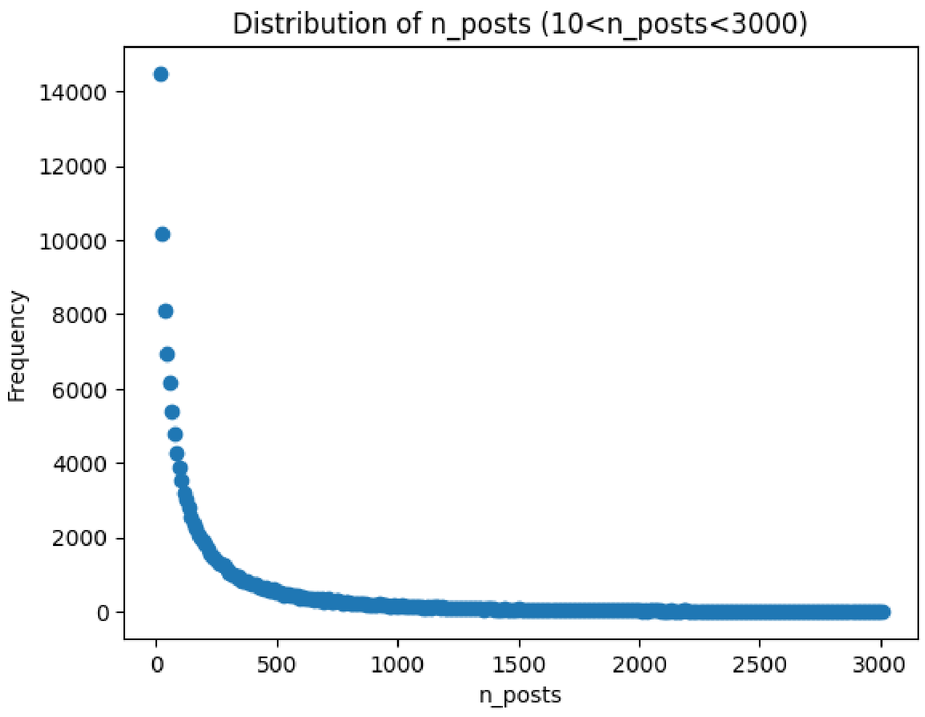


We can see that the relative share of articles with tagged with forum associated story labels increases above 1.500 posts. If we also plot the na, values, we can see them actually becoming the minority above ~2.700 posts, but we left out those Na in order to better illustrate the relative share of other storylabels decreasing relative to Liveposts.

But lets return to the distribution of posts and focus the scatterplot a bit. This time we will omit posts within the 0-10 range as that bucket dwarfs the scale of the following buckets (we're avoiding a log scale for now). We'll also only plot the buckets up to 3.000 posts:

```
In [9]: plt.scatter(binned_table['bin_middle'][1:301], binned_table['counts'][1:301])
        plt.title('Distribution of n_posts (10<n_posts<3000)')
        plt.xlabel('n_posts')
        plt.ylabel('Frequency')
```

```
Out[9]: Text(0, 0.5, 'Frequency')
```

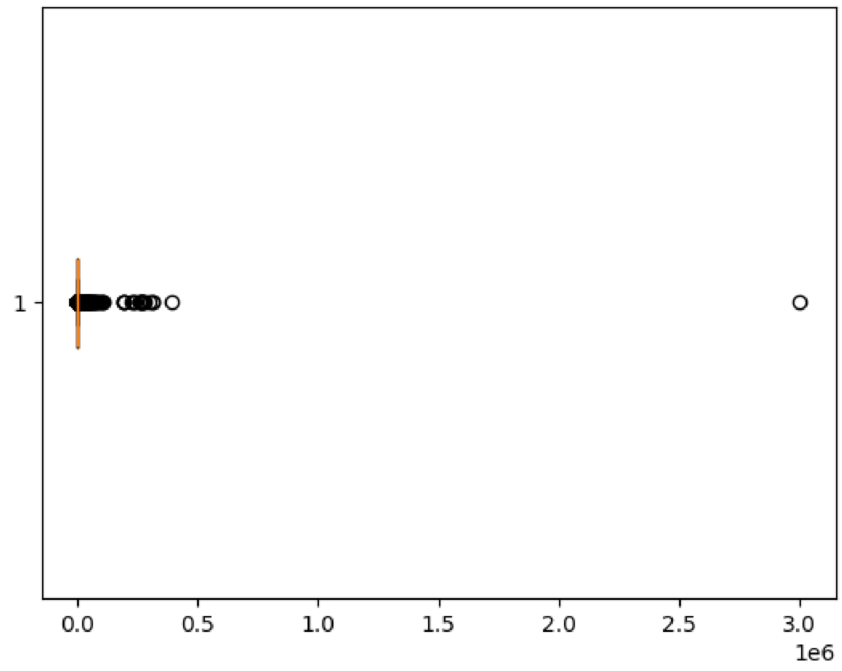


The distribution of posts seems to exhibit a power law distribution. The curve very beautifully follows some logarithmic function.

As seen in the first distribution plot, the distribution has an extremely long tail that gradually thins out. The 3million ticker is a very strong outlier here, but even up to 400 thousand posts there are still a few articles in our large data set. Before we decide on a strategy on how to deal with those, lets check out the boxplot:

In [10]:
```python
plt.boxplot(df['n_posts'], vert=False, )
print('Five-point summary:\n', df['n_posts'].describe(), '\n')
print('Increasingly smaller buckets:\n', df['n_posts'].quantile([.25, .5, .75, .9, .95, .99, .999, .9999]), '\n')
```

```
Five-point summary:
 count    1.812000e+05
mean     3.359409e+02
std      7.640062e+03
min      0.000000e+00
25%      1.500000e+01
50%      7.300000e+01
75%      2.510000e+02
max      3.000166e+06
Name: n_posts, dtype: float64

Increasingly smaller buckets:
 0.2500        15.000000
0.5000        73.000000
0.7500       251.000000
0.9000       637.000000
0.9500      1047.000000
0.9900      2751.020000
0.9990     20050.811000
0.9999    181071.088898
Name: n_posts, dtype: float64
```



This again confirms that there is a very long tail. For the purposes of our machine learning project, we have to consider how we'll deal with this long stretch. Initally the idea was to produce a regression neural network, with a single output neuron that would output continuous values. In this case, the outliers would have caused much trouble with scaling the data and we would have probably trimmed our dataset above a certain percentile-based threshold e.g. 99.99%.

We decided to simplify our regression problem by using (roughly) equally sized bins which are still an ordinal variable so we can still perform a regression without loosing the valuable information for the gradient of how wrong a given prediction is. I.e. instead of just using categories, we can still perform regression over those bins and tell the network in our loss function how far a predicted bin is from the actual bin. Those bins should cover meaningful ranges in the number of posts, reflecting the logarithmic distribution of posts.

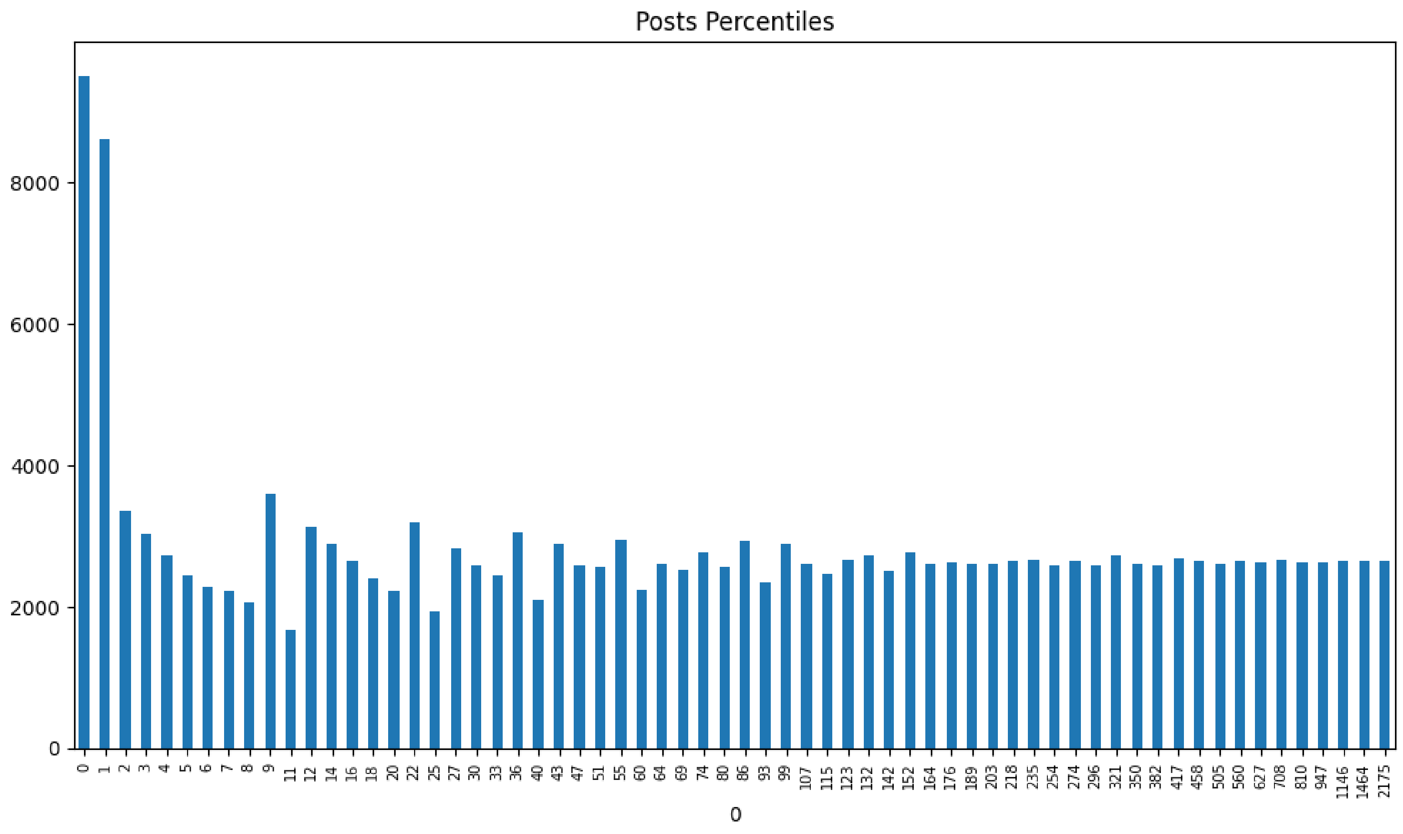
For this we use the qcut pandas function. This function however throws an error if the bins are not equally sized. Because there are so many articles with 0 posts (5% of our data), it will be impossible to make a bin that contains 1 % of the data withing those bounds. Similarly the articles with 1 post are also a bin that would be larger than the other bins. Thus we:

1. create a special bin for all articles with 0 posts
2. we set the `duplicates=drop` kwarg which allows for uneven sized bins (it will drop overlapping buckets)

Considering the power-law distribution of our data, it is inevitable that the first buckets consist of very small ranges. Even with manually assigning 0-posts to their own bucket, the first set of buckets are actually also just single-value (e.g. '2', '3') ranges with a disproportionally large number of observations in them. For training purposes it of course would be good if all our bins are equally sized - but maybe it is also important to make the network good at spotting unsucessful posts.

First we tried 100 buckets but saw that the qcut function dropped a lot of buckets. Thus we decided for a smaller number of buckets that could still meaningfully disect the data and just went with 64 buckets because it's a nice number. Furthermore you will notice that q is set to 65, because there are 2 overlapping buckets which will be dropped.

In [11]:
```python
nonzero = df['n_posts'] > 0

# Apply qcut to these rows and convert the result to string
df.loc[nonzero, 'n_posts_percentile'] = pd.qcut(df.loc[nonzero, 'n_posts'], q=65,
                                                duplicates='drop').astype(str)

# Set the 0 posts to '0'
df.loc[df['n_posts'] == 0, 'n_posts_percentile'] = '(0.0, 1.0]'

# extract the lower bound of the bins and convert to float
float_pattern = r'\d+\.\d+'
bounds = df['n_posts_percentile'].value_counts().index.str.extract(
    f'({float_pattern}), ({float_pattern})').astype(float)
bounds = bounds.round().astype(int)

s = pd.Series(df['n_posts_percentile'].value_counts().values, index=bounds[0])
s.sort_index().plot(kind='bar', figsize=(10, 6))
plt.xticks(fontsize=7)
plt.title('Posts Percentiles')
plt.tight_layout()
plt.show()
print(f'There are {len(df['n_posts_percentile'].value_counts())} buckets')
```



```
There are 64 buckets
```

The first set of buckets turn out to just be the range from 0 to 12, and we can see that the 0 and 1 buckets are both very large bins. We can also see that our percentiles cut off at around 2.100 posts, after which the long tail begins. Compared to the initial distribution plots though, we have mostly flattened the curve.

To produce the final target variable, we map those buckets to the range from 0 to 63. We won't transform this data any further as we hope that a sufficiently large network will be able to cope with that range without centering the values as well. As the buckets are roughly evenly distributed now (save for 0 and 1), we will not normalize our data.

In [12]:
```python
target_map = pd.DataFrame({
    'lower_b': bounds[0],
    'upper_b': bounds[1],
    'bounds': df['n_posts_percentile'].value_counts().index
})

target_map.sort_values(by='lower_b', inplace=True)
target_map.reset_index(drop=True, inplace=True)

# Map 'n_posts_percentile' to 'bounds' and assign the index of the matching row to 'target'
df['target'] = df['n_posts_percentile'].map(target_map.reset_index().set_index('bounds')['index'])
# save the target map
target_map.to_csv('./data/target_map.csv', index=False)
target_map
```

```
Out[12]:
```

	lower_b	upper_b	bounds
0	0	1	(0.0, 1.0]
1	1	2	(0.999, 2.0]
2	2	3	(2.0, 3.0]
3	3	4	(3.0, 4.0]
4	4	5	(4.0, 5.0]
...	...	...	...
59	810	947	(810.0, 947.0]
60	947	1146	(947.0, 1146.0]
61	1146	1464	(1146.0, 1464.0]
62	1464	2175	(1464.0, 2175.0]
63	2175	3000166	(2175.0, 3000166.0]

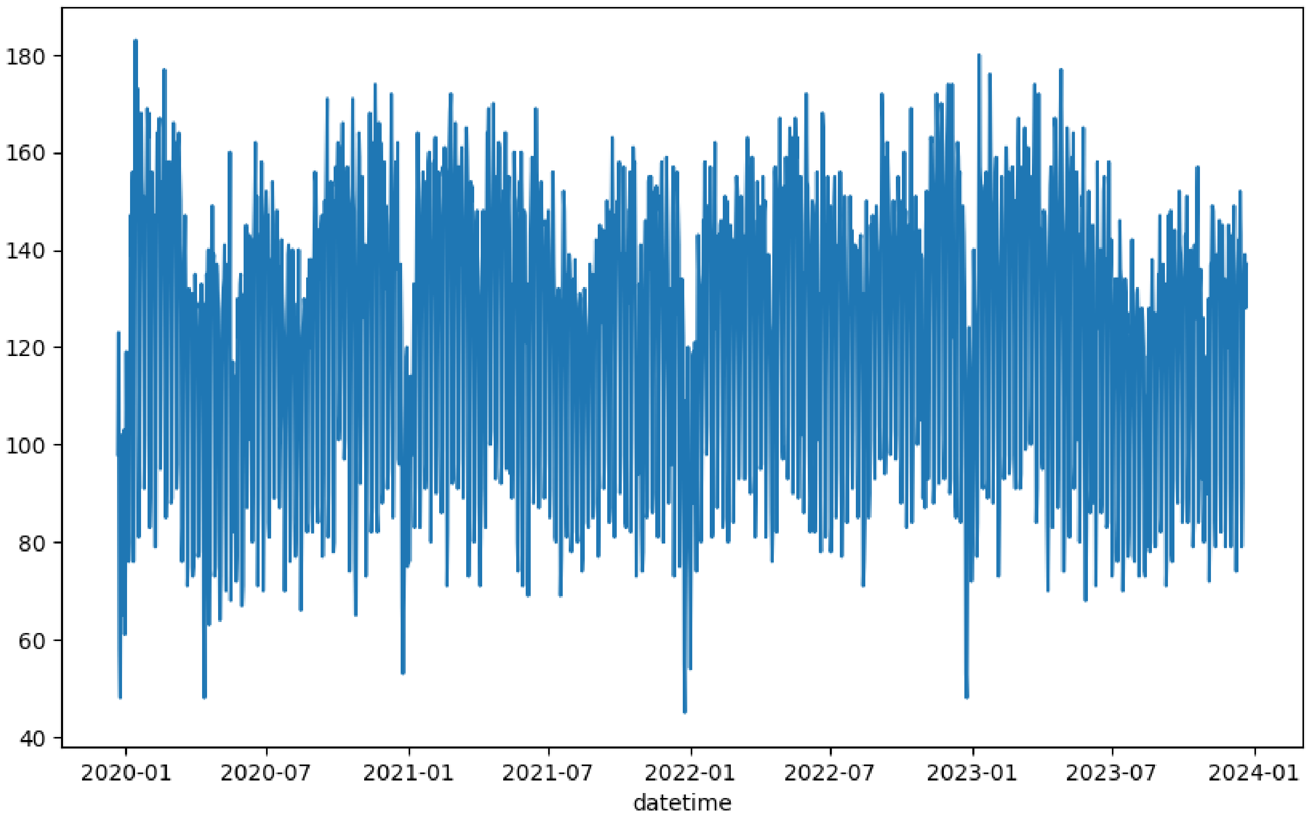
64 rows × 3 columns

Time series analysis

First, lets' plot the number of articles on the frontpage per day

```
In [13]: df.groupby(df['datetime'].dt.date).count()['title'].plot(figsize=(10, 6))
```

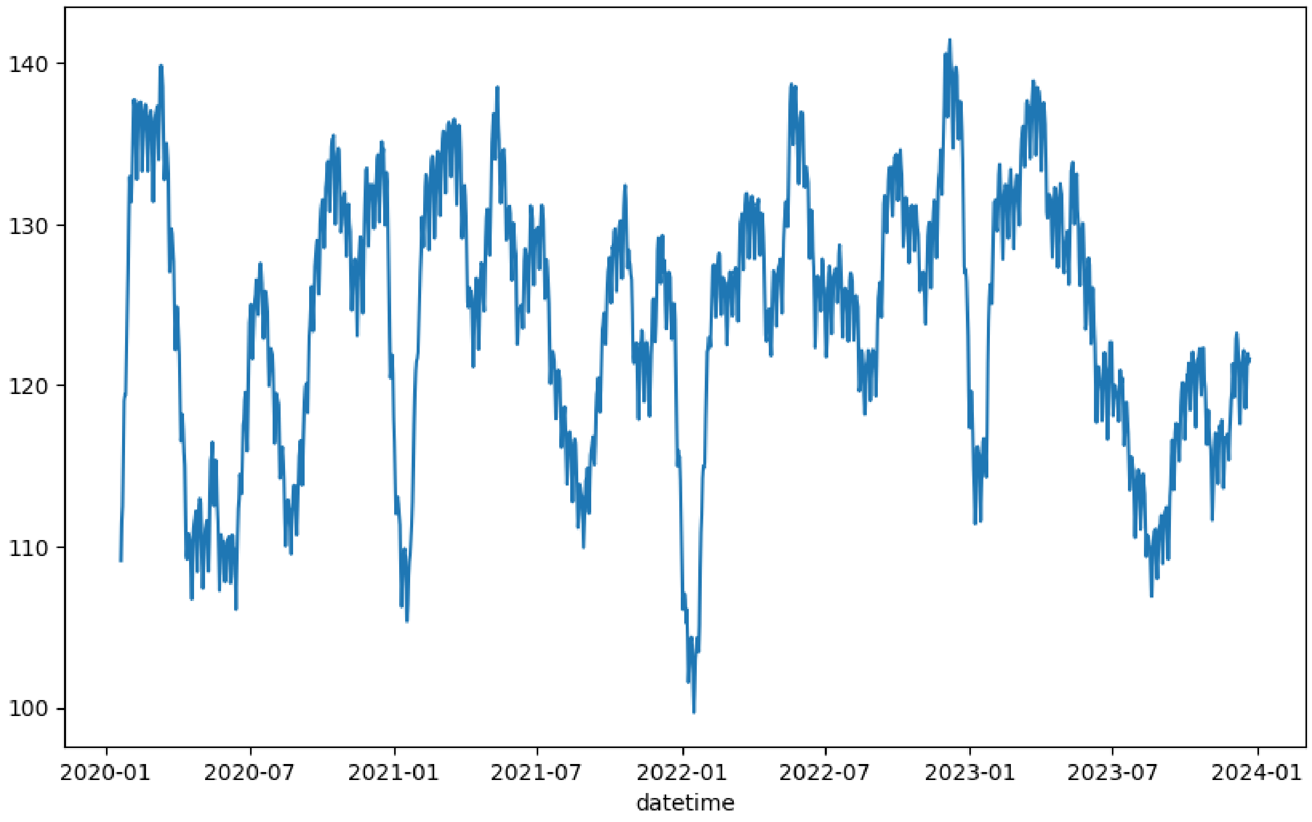
```
Out[13]: <Axes: xlabel='datetime'>
```



Too many fluctuations, let's look at the rolling 30 day average

```
In [14]: df.groupby(df['datetime'].dt.date).count()['title'].rolling(30).mean().plot(figsize=(10, 6))
```

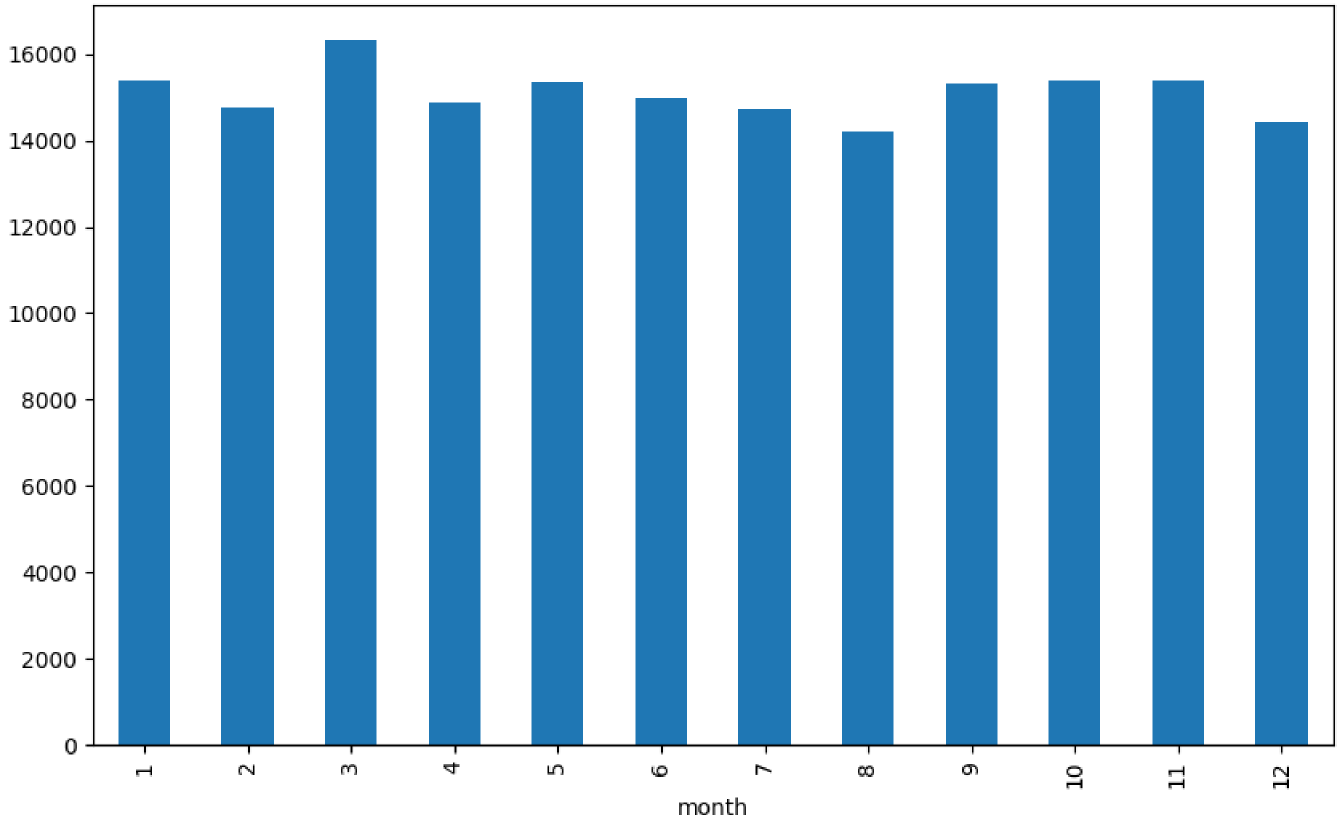
```
Out[14]: <Axes: xlabel='datetime'>
```



The number of articles per day does seem to follow some seasonal trends Maybe at the end of the year the number of articles drops, as well as during the summer months due to the Sommerloch. Let's investigate this by aggregating the number of articles by month

```
In [15]: df['month'] = df['datetime'].dt.month
         df.groupby('month').count()['title'].plot(kind='bar', figsize=(10, 6))
```

```
Out[15]: <Axes: xlabel='month'>
```

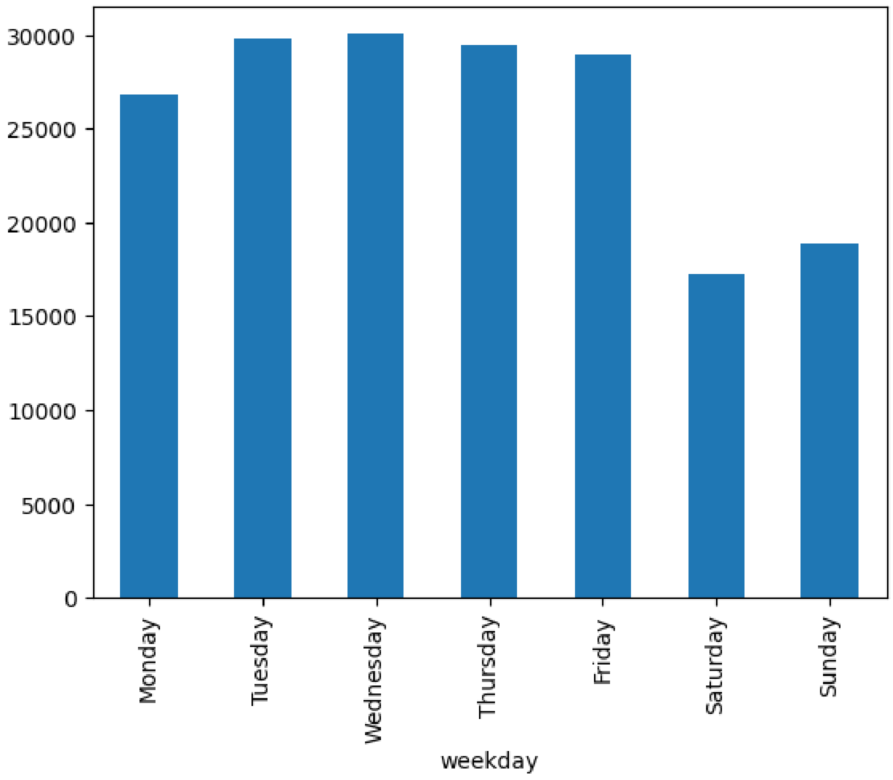


Well, that rejects that hypotheses, the articles seem to even out over 4 years by month.

Next, lets add a weekday column to our data

```
In [16]: df['weekday'] = df['datetime'].dt.weekday
         df.groupby('weekday').count()['title'].plot.bar()
         plt.xticks(np.arange(7), ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
         plt
```

Out[16]:
```
<module 'matplotlib.pyplot' from 'c:\\Users\\Paul\\AppData\\Local\\Programs\\Python\\Python312\\Lib\\site-packages\\matp
lotlib\\pyplot.py'>
```

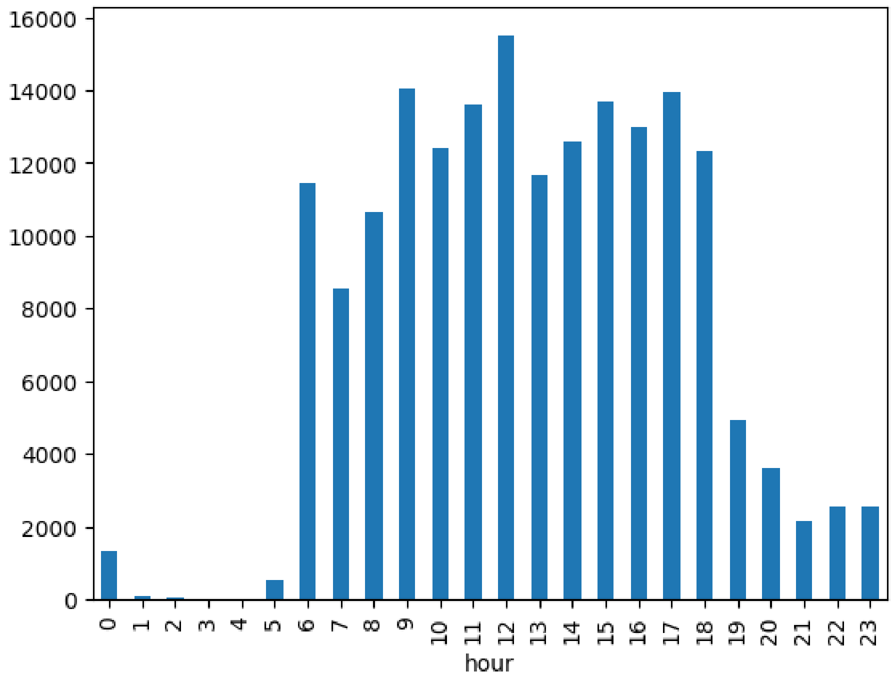


less datapoints for the weekends it seems. We will make use of this weekday variable in our predictions as well.

We'll also add a time of day column and plot this.

In [17]:
```
# add a hour column
df['hour'] = df['datetime'].dt.hour
# which hour has the most articles?
df.groupby('hour').count()['title'].plot.bar()
```

Out[17]:
```
<Axes: xlabel='hour'>
```



Most articles are posted during the day from 6am to 6pm. Continuosly adding new articles during the day probably helps engagement as well.

Label analysis

We already mentioned in the scraping functions that most articles seem to lack storylabels, but let's investigate this a bit more.

In [18]:
```
label_counts = df['storylabels'].value_counts(dropna=False)
print(label_counts)
print(f'\nPercentage of articles with a story label: {100-(label_counts.iloc[0]/len(df)) * 100:.2f}%')
```

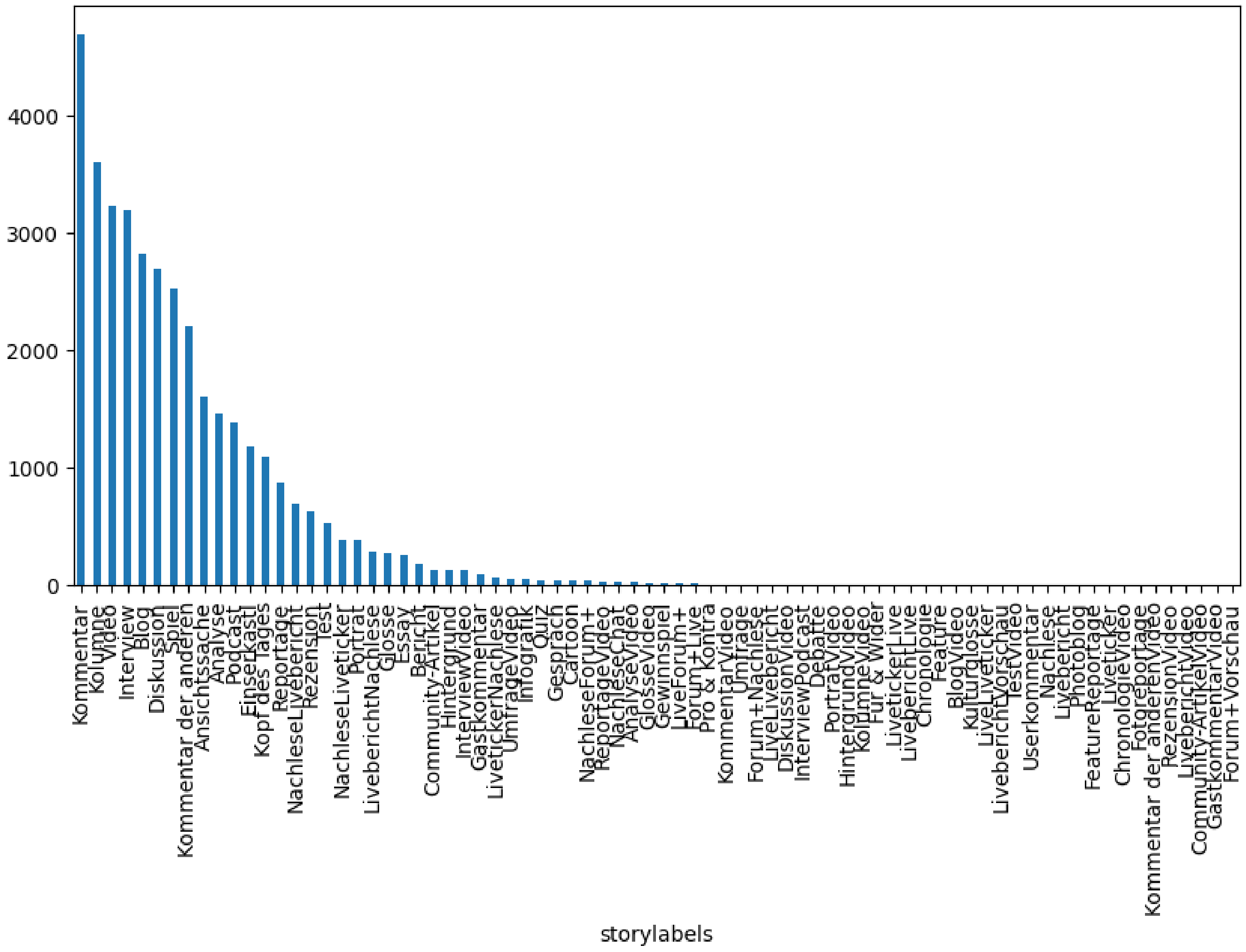
```
storylabels
NaN                 143784
Kommentar             4694
Kolumne               3608
Video                 3238
Interview             3199
                     ...
Liveticker               1
Userkommentar            1
FeatureReportage         1
Photoblog                1
Forum+Vorschau           1
Name: count, Length: 77, dtype: int64

Percentage of articles with a story label: 20.65%
```

only around 20% of articles are labeled. We will not use those labels for our prediction but they were useful for filtering out Ticker posts before. As mentioned, it would also be an interesting labelling task to find labels for the remaining 80% but for now we will focus on the n_posts variable.

Next we will not look at the number of posts a given label is going to generate, just at which are the most common storylabels.

In [19]:
```python
# Plot the number of articles per story label
plt.figure(figsize=(10, 5))
df['storylabels'].value_counts().plot.bar()
```

Out[19]:
```
<Axes: xlabel='storylabels'>
```



This time the Ticker labels are further behind the distibution, the list being headed by periodicals like Kommentar, Kolumne etc.

Lastly, lets look at the kicker labels, for which there are no na values.

In [20]:
```python
kickers = df['kicker'].value_counts(dropna=False)
print(f'Number of unique kicker labels: {len(kickers)}\n')
print(f'Most common kicker labels:\n{kickers[:20]}\n')

print(f'There are only {df['kicker'].isna().sum()} na kickers, lets drop them')
# drop kicker na values
df.dropna(subset=['kicker'], inplace=True)
```

```
Number of unique kicker labels: 44720

Most common kicker labels:
kicker
Fußball                 3133
Nachrichtenüberblick    3099
Netzpolitik             2674
Sudoku                  2414
Bundesliga              1711
Sport                   1651
USA                     1515
IT-Business             1464
Coronavirus             1375
Games                   1356
Tennis                  1244
Switchlist              1208
Krieg in der Ukraine    1203
Deutsche Bundesliga     1180
Etat-Überblick          1161
Hans Rauscher           1153
Wintersport             1123
TV-Tagebuch             1080
Eishockey               1058
Thema des Tages         1032
Name: count, dtype: int64

There are only 84 na kickers, lets drop them
```

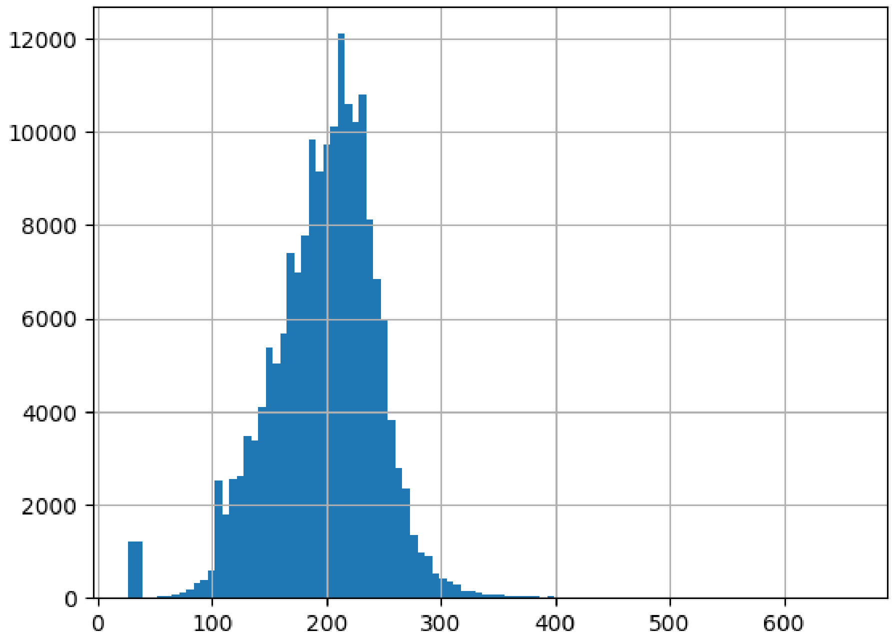
Those also are dominated by periodicals like Fußball, Nachrichtenüberlick (daily summary of posted articles) etc.

The title along with the subtitle will be our most important predictors. In the following codecell, we concatenate those into a long string along with the kicker-label. We will later use embeddings to encode those strings as numeric vectors.

First, let's plot the distribution of string lengths.

In [21]:
```python
df['text'] = df['title'] + ' ' + df['kicker'].fillna('') + ' ' + df['subtitle'].fillna('')

print('Example of concatted string:\n', df['text'][0], '\n')
df['text'].str.len().hist(bins=100)
short_txt = df['text'].apply(len).nsmallest(3).index.map(df['text']).tolist()
print('\nShortest titles:\n', '\n'.join(list(short_txt)))

print(f'\nMean title length: {df["text"].str.len().mean():.2f} characters')
print(f'Standard deviation: {df["text"].str.len().std():.2f} characters')
print(f'Median title length: {df["text"].str.len().median():.2f} characters')
```

```
Example of concatted string:
 Real Madrid stolpert mit Aluminiumpech im Titelrennen Primera Division Die Königlichen können Bilbao daheim nicht besiege
n


Shortest titles:
 Sudoku mittel 4493a Sudoku
Sudoku mittel 4497a Sudoku
Sudoku mittel 4503a Sudoku

Mean title length: 197.88 characters
Standard deviation: 47.80 characters
Median title length: 203.00 characters
```



Again a distribution with a long tail that thins out very gradually. During our nlp processing, we will generate vectors of uniform length - thus there is no need to trim our texts.

The first large bin are the sudokus - short texts of roughly the same length. We will also keep them as they are also a part of the frontpage

Vectorizing our features

As inputs for our network, we need numeric values. The datetime information is already basically numeric, but the text will need some processing.

Text Processing

For the following steps, we will be using the spacy library which will help us with all the lemmatization and tokenization we need to turn our strings into tokens. We will also be using the german 'news' (large) pipeline, available here - Warning, it is 500mb of size. Its vocabulary consists of many newspaper headlines, so it should be perfect for our application. It also has embedding vectors for a large vocabulary (more on that in the next cell).

In case you want to run this locally, it can be installed using `python -m spacy download de_core_news_md`.

In [22]:
```
import spacy

# nlp = spacy.load("de_core_news_md")
nlp = spacy.load("de_core_news_lg")

print("Pipeline Components:\n", ", ".join([n for n, p in nlp.pipeline]))
```

```
Pipeline Components:
 tok2vec, tagger, morphologizer, parser, lemmatizer, attribute_ruler, ner
```

Generating document vectors

This step will add a new column `spacy_vector`, which contains the mean of all token vectors of a document (headline+subtitle+tags) generated by the embeddings of the spacy model. Spacy uses *Word2VecEmbeddings*, which encodes semantic meaning based on the word's contextual use in the `de_core_news` dataset.

These document vectors provide a numerical representation of our headlines, while still retaining information and at the same time producing vectors of uniform size.

In [23]:
```
from tqdm import tqdm
tqdm.pandas() # for progress_apply

def process_text(text):
    doc = nlp(text)
    return doc.vector

df['doc_vector'] = df['text'].progress_apply(process_text)
```

```
  0%|          | 0/181116 [00:00<?, ?it/s]
```

```
100%|██████████| 181116/181116 [29:10<00:00, 103.45it/s]
```

In case we want to work with the information of our model again, we save our nlp object as well.

In [24]:
```python
print('The vectors are of length ', df['doc_vector'][0].shape)
print('The vectors are of type', df['doc_vector'][0].dtype)
print('\n', df['doc_vector'].head())
```

```
The vectors are of length  (300,)
The vectors are of type float32

 0    [-0.37552637, 0.20481217, -0.216765, 0.0486718...
1    [-0.080757536, 0.54684085, -0.5815048, 0.41144...
2    [0.69385666, -0.0836207, -0.5870395, 0.8096802...
3    [-0.14959173, -0.72091854, -1.3704777, -1.0522...
4    [-0.021010712, -0.18174057, -0.4104177, -1.398...
Name: doc_vector, dtype: object
```

We will finally create our inputs column which also contains the month, weekday and hour. We think this information might be useful, e.g. if certain stories are posted in the summer months/ winter, if it is a particular day of the week, if it was posted in the morning or evening. We omit the year to limit overfitting, as well as it's limited usefulness when we consider that those headlines are newsstories and we now have 2024.

We will also normalize those date vectors which means their elements should be roughly the same size as the elements of the document vectors.

In [25]:
```python
import numpy as np

def process_row(row):
    date_vector = np.array([
        row['datetime'].month,
        row['datetime'].weekday(),
        row['datetime'].hour
        ])
    # Normalize the date_vector, ensure it is float32 like the doc vectors
    norm_date_vector = (date_vector / np.linalg.norm(date_vector)).astype(np.float32)
    return np.concatenate((norm_date_vector, row['doc_vector']))

df['inputs'] = df.apply(process_row, axis=1)

print(f'Sequence length: {len(df['inputs'][0])}')
```

```
Sequence length: 303
```

Saving to parquet

The data processing is thus completed. We will save our data using the parquet file format which not only handles dtypes like lists natively, it is also a binary format that is much more efficient at storing all our data. You will need to install `pyarrow` if you want to run this yourself.

In [26]:
```python
df[['inputs', 'target', 'n_posts']].to_parquet('data/ml_data.parquet', index=False)
```

Machine Learning

Loading Data

We open our ML-ready dataset

- **inputs:** First we concatenated the title, subtitle and article labels. Then we tokenized & lemmatized individual words, and used Spacys large german model to look up embedding vectors for every token. Those vectors were originally generated using the word2vec algorithm, where the goal is to place words in a 300-dimensional space where words that are used in similar contexts have a smaller euclidean distance in the vector space. Then we average all word vectors for the tokens in our concatenated text, which for our short texts should roughly correspond to the overal position of the text in the 300-dimensional space expressed by the vector. To this we append the year, month and weekday as components of a normalized vector.

- **targets:** The targets correspond to 64 bins, each bin representing a range in the number of posts an article has generated. This was done in order to:

  1. Have our target variable be roughly evenly distributed, as the bins grow in size to mitigate the power-law distribution present in the actual values.
  2. Limit the range of values the neural network needs to predict.

  In effect, we have turned a regression problem into a classification, and then back into a regression again. This way we can still preserve the meaning if a prediction closely misses the correct bin, which lets the neural network adjust more precisely than in a classification where this distance information would be lost.

In [1]:
```
import pandas as pd

df = pd.read_parquet('data/ml_data.parquet') # requires pyarrow
df.head()
```

Out[1]:

	inputs	target	n_posts
0	[0.45066947, 0.22533473, 0.8637831, -0.3755263...	18	30
1	[0.4656903, 0.23284516, 0.8537656, -0.08075753...	12	16
2	[0.48154342, 0.24077171, 0.84270096, 0.6938566...	34	104
3	[0.48154342, 0.24077171, 0.84270096, -0.149591...	20	35
4	[0.49827287, 0.24913643, 0.8304548, -0.0210107...	3	4

Loading df into torch, train test split

Now we start using pytorch. I was able to finally play with cuda on my trusty old gtx970, but this code should be agnostic to the type of device.

In the following cell, we create tensors for our inputs and target values, and combine them into a tensor dataset, on which we perform an 80/20 test split. We chose batch size 64, meaning 64 datapoints are pushed to the gpu in one training run. This value utilizes my gpu slightly more (cuda usage according to task manager at 40%), while at the same time hopefully not beeing too large to cause overfitting. The train and test loader objects will be responsible for pushing the data to the gpu.

In [2]:
```
import torch
from torch.utils.data import DataLoader, TensorDataset, random_split

# Using cuda
my_device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print('We are using ', my_device)

# Create a TensorDataset
inputs = torch.tensor(df['inputs'].tolist(), device=my_device)
targets = torch.tensor(df['target'].values, device=my_device)
```

```
dataset = TensorDataset(inputs, targets)

# Split into training and test set
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Create DataLoaders
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

```
We are using  cuda
```

```
C:\Users\Paul\AppData\Local\Temp\ipykernel_12152\2500532675.py:9: UserWarning: Creating a tensor from a l
ist of numpy.ndarrays is extremely slow. Please consider converting the list to a single numpy.ndarray wi
th numpy.array() before converting to a tensor. (Triggered internally at ..\torch\csrc\utils\tensor_new.c
pp:278.)
  inputs = torch.tensor(df['inputs'].tolist(), device=my_device)
```

Model definitions

Deep regression network

Here we define our regression neural network. It takes as an argument the layer sizes and generates `len(layers_sizes)` layers. Furthermore we also add a dropout probability, meaning that during training, with a set probability, a neurons output will randomly be ignored, meaning its output will not be fed as input to the next layer. This pruning technique should hopefully make our model more robust to overfitting as the model should not become overly reliant on single neurons or pathways through the network, encouraging a more distributed and robust internal representation.

As activation functions, we use the nonlinear ReLu function, which worked decently in our testing.

The output layer consists of a single neuron, which should predict values in the range 0-63. We do not perform activation on this final output as to not squash its output.

In [3]:
```
import torch.nn as nn
import torch.nn.functional as F


class RegressionNN(nn.Module):
    def __init__(self, drop, layer_sizes):
        super(RegressionNN, self).__init__()

        # Create layers dynamically
        self.layers = nn.ModuleList()
        for i in range(len(layer_sizes)-1):
            self.layers.append(nn.Linear(layer_sizes[i], layer_sizes[i+1]))

        self.dropout = nn.Dropout(drop)

        self.output_layer = nn.Linear(layer_sizes[-1], 1)

    def forward(self, x):
        for layer in self.layers:
            x = F.relu(layer(x))
            x = self.dropout(x)

        x = self.output_layer(x)
        return x
```

Training loop

Here we define the training loop, which takes as parameters the number of epochs and learning rate. An evaluation criterion can optionally be specified but we decided to go with the standard mean-squared-error approach as it is a staple in regression problems and maps well to our intention of minimizing the error of our predictions.

In [4]:
```
import torch
import torch.optim as optim
```

```python
from tqdm import tqdm

def train(model, epochs, learning_rate, criterion=nn.MSELoss()):
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    losses = []
    for epoch in tqdm(range(epochs)):
        model.train()
        total_loss = 0

        for inputs, targets in train_loader:
            inputs, targets = inputs.to(my_device), targets.to(my_device)
            optimizer.zero_grad()
            outputs = model(inputs.float())
            loss = criterion(outputs.squeeze(), targets.float())
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        epoch_loss = total_loss/len(train_loader)
        losses.append(epoch_loss)
    return losses
```

Evaluation

On test set

This function will print the average mean squared error among all batches of the training set (20% of our data).

In [5]:
```python
def eval_on_test(model, criterion=nn.MSELoss()):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for inputs, targets in test_loader:
            inputs, targets = inputs.to(my_device), targets.to(my_device)
            outputs = model(inputs.float())
            loss = criterion(outputs.squeeze(), targets.float())
            test_loss += loss.item()
    print(f"Test Loss: {test_loss/len(test_loader)}")
```

Plotting the loss curve

This function simply plots the loss as a function of the number of epochs during training, allowing us to monitor our training efficiency and check once the model stops improving.

In [6]:
```python
import matplotlib.pyplot as plt

def plot_losses(losses):
    # Plot the losses
    plt.plot(losses)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Loss Curve')
    plt.show()
```

Testing on the current frontpage

Finally, we thought it would be interesting to fetch a given day's actual frontpage and see how our model would fare.

For this, 4 functions are defined:

- `get_final_articles_df():` Unfortunately I found no good way to import functions from other notebooks as you would normally do with python modules. Thus this method violates DRY and is a copy-paste of everything needed from the scraping and processing notebook to return a dataframe of a given day's frontpage, along with processing the text into a vector.
- `plot_preds():` This plots our predicted bounds as a bar, along with the actual values as a red line, sorted in ascending order of actual number of posts.

- `print_preds()`: This prints all the articles that were within our predicted bounds, as well as the two articles furthest away from our prediction.
- `compare_prediction_for_date()`: This is just a wrapper for the above three functions, which takes a model as well as a date as parameters.

In [7]:
```python
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup
from functools import cache
import numpy as np
import requests
import spacy

@cache # We cache the results of this function to avoid fetching the same data multiple times
def get_final_articles_df(date):
    link = f'https://www.derstandard.at/frontpage/{date.strftime('%Y/%m/%d')}'
    # fetch the html content of a derstandard.at page
    response = requests.get(link, cookies={'DSGVO_ZUSAGE_V1': 'true'})
    soup = BeautifulSoup(response.content, 'html.parser')
    # get the articles
    articles_dict = {}
    articles = soup.select('div.chronological>section article')
    for article in articles:
        title_tag = article.find('a')
        if title_tag and title_tag.has_attr('title'):
            title = title_tag['title']
            articles_dict[title] = article
    # make a list of the articles
    HOST = 'https://www.derstandard.at'
    article_data = []
    for title, article in articles_dict.items():
        data = {
            'title': title,
            'teaser-subtitle': None,
            'link': None,
            'time': None,
            'teaser-kicker': None,
            'n_posts': None,
            'storylabels': None
        }
        link = article.find('a')['href']
        if not link.startswith(HOST):
            link = HOST + link
        data['link'] = link
        time = [tag for tag in article.find_all('time') if 'datetime' in tag.attrs][0]
        data['time'] = time['datetime'].rstrip('\r\n')
        n_posts = article.find('div', 'teaser-postingcount')
        try: data['n_posts'] = int(n_posts.get_text(strip=True).rstrip('Posting').replace('.', ''))
        except: data['n_posts'] = 0
        for tag, class_name in [('p', 'teaser-kicker'),
                                ('p', 'teaser-subtitle'),
                                ('div', 'storylabels')]:
            found_tag = article.find(tag, class_=class_name)
            if found_tag:
                data[class_name] = found_tag.get_text(strip=True)
        article_data.append(data)
    # make a df
    df = pd.DataFrame(article_data)
    df.columns = df.columns.str.replace('teaser-', '')
    df.rename(columns={'time': 'datetime'}, inplace=True)
    df['datetime'] = pd.to_datetime(df['datetime'])
    df['text'] = df['title'] + ' ' + df['kicker'].fillna('') + ' ' + df['subtitle'].fillna('')
    # add embeddings
    nlp = spacy.load("de_core_news_lg")
    df['doc_vector'] = df['text'].apply(lambda t: nlp(t).vector)
    # add date
    def process_row(row):
        date_vector = np.array([
            row['datetime'].month,
            row['datetime'].weekday(),
            row['datetime'].hour
            ])
        # Normalize the date_vector, ensure it is float32 like the doc vectors
        norm_date_vector = (date_vector / np.linalg.norm(date_vector)).astype(np.float32)
        return np.concatenate((norm_date_vector, row['doc_vector']))
    df['inputs'] = df.apply(process_row, axis=1)
    return df


def plot_preds(preds):
    # Create a copy of preds and sort by actual value
    preds_copy = sorted(preds, key=lambda x: x[0])
    actual, lower_bound, upper_bound = zip(*preds_copy)
    # Create the plot
    plt.figure(figsize=(10, 6))
    plt.plot(actual, color='red', label='Actual')
    plt.fill_between(range(len(actual)), lower_bound, upper_bound, color='grey', alpha=0.5, label='Predi
    plt.xlabel('Articles')
    plt.ylabel('number of posts')
    plt.title('Actual vs Predicted')
    plt.legend()
    plt.show()


def print_preds(preds, articles):
    within_bounds, outside_bounds = [], []
    for i, (act, lower, upper) in enumerate(preds):
        if lower <= act < upper: # upper bound is exclusive
            within_bounds.append(i)
        else:
            distance = min(abs(act - lower), abs(act - upper))
            outside_bounds.append((i, distance))
    # Sort the outside_bounds list by distance in descending order and get the first two indices
    furthest_indices = [i for i, _ in sorted(outside_bounds, key=lambda x: x[1], reverse=True)[:2]]
    # Fetch the rows from the articles DataFrame for within_bounds
    def print_in_color(indices, articles, preds, prediction_type):
        for i in indices:
            row = articles.iloc[i]
            print(f"{prediction_type} Prediction: {row['text']}")
            print(f"Number of Posts: {row['n_posts']}")
            print(f"Predicted Bounds: ({preds[i][1]}, {preds[i][2]})")
            print(f"Link: {row['link']}\n")
    # Print the correct predictions in green
    print("\033[92m")
    print_in_color(within_bounds, articles, preds, "Correct")
    # Print the incorrect predictions in red
    print("\033[91m")
    print_in_color(furthest_indices, articles, preds, "Incorrect")
    # Reset the color back to normal
    print("\033[0m")


@cache
def compare_prediction_for_date(model, date):
    # fetch articles for the date
    articles = get_final_articles_df(date)
    target_map = pd.read_csv('data/target_map.csv')
    predicted = [] # list of tuples (actual, pred_lower_bound, pred_upper_bound, row)
    for i, row in articles.iterrows():
        input_data = torch.tensor(row['inputs'], dtype=torch.float)
        input_data = input_data.to(next(model.parameters()).device)
        # Reshape the input data and pass it through the model
        input_data = input_data.unsqueeze(0)  # Add a batch dimension
        with torch.no_grad():
            output = model(input_data)
        target_index = min(int(output.item()), 63) # ensure we don't go out of bounds
        tm = target_map.iloc[target_index]
        predicted.append((row['n_posts'], tm.lower_b, tm.upper_b))
    plot_preds(predicted)
    print_preds(predicted, articles)
```

Wrapping it all into a single function

And lastly, one function to wrap all the evaluations into one.

```
In [8]: def eval_trained_model(model, losses, date):
            plot_losses(losses)
            eval_on_test(model)
            compare_prediction_for_date(model, date)
```

Testing different models

We will be testing regression models of different sizes. For our evaluations of a particular frontpage, we picked a frontpage that is a week old (31.12.23), giving the articles enough time to gather posts (as in our training data).

```
In [9]: # pick test date one week ago
        test_date = datetime.now() - timedelta(days=7)
```

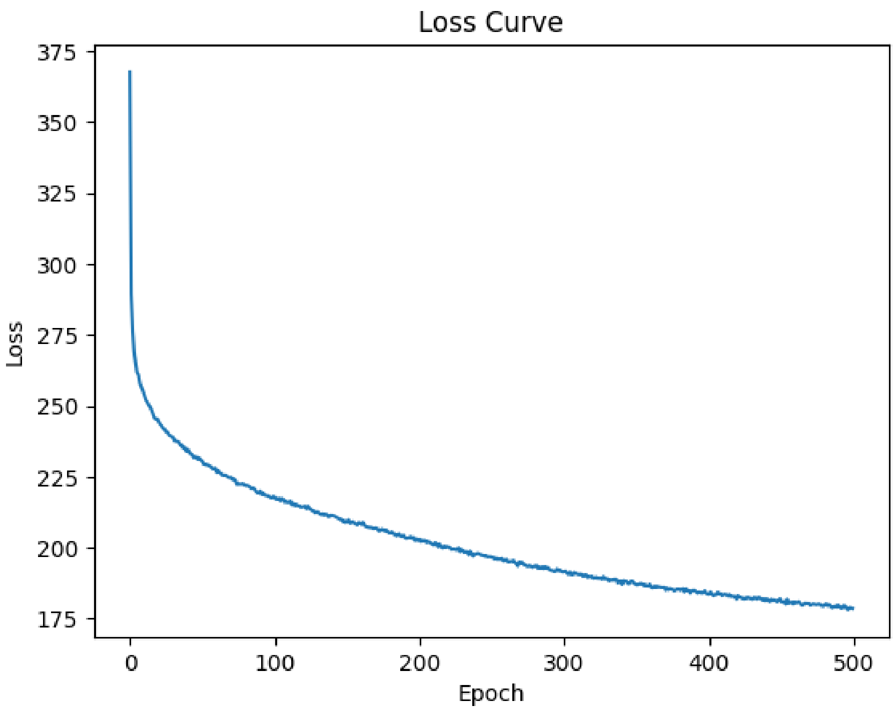
Small regression Model

First is a small-ish Regression network.

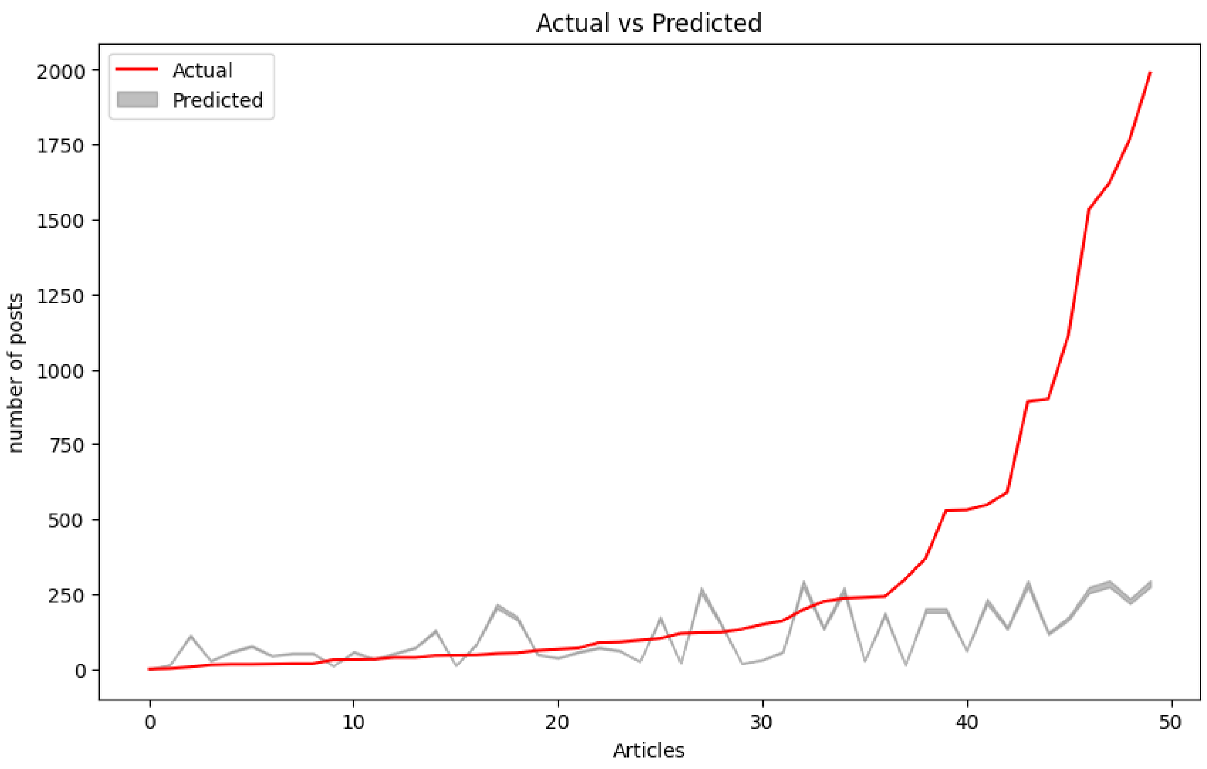
- `drop = 0.3`, as we found that higher dropout values took extremely long to converge and did not really help with the overfifting problem. Our training loss was always around 20% higher than our test-set loss.
- `layer_sizes = [303, 128, 64]`. The idea here is to create a sort of information funnel, where each layer should hopefully pick up on higher-level features in the data, with fewer neurons being responsible for aggregating information from the previous layer. If I were to fantasize what those higher-level-features could be - the 300-dimensional position of the text could represent the overall sentiment, picked up by higher layers? But really we have no idea how the network interprets those vectors. The idea behind having 64 neurons is to have one neuron per bucket- each one maybe contributing `1` to the final output neuron. But again we are speculating, let's see how it performs.

```
In [10]: small_model = RegressionNN(
             drop = 0.3,
             layer_sizes = [303, 128, 64]
             ).to(my_device)

         print('training small model')
         small_losses = train(small_model, epochs=500, learning_rate=0.0001)
         print('evaluating small model')
         eval_trained_model(small_model, small_losses, test_date)
```

```
training small model
100%|██████████| 500/500 [1:02:39<00:00,  7.52s/it]
evaluating small model
```



```
Test Loss: 217.58059333828228
```



```
Correct Prediction: Web- und Games-News: Gebrauchtwagen-Besitzer könnten bald Abogebühren bezahlen Nachri
chtenüberblick Das sind die aktuellen Schlagzeilen aus Web und Games
Number of Posts: 0
Predicted Bounds: (0, 1]
Link: https://www.derstandard.at/story/3000000201431/web-und-games-news-gebrauchtwagen-und-abo

Correct Prediction: Das Universum als Perspektivgeber Feelgood Manchmal hilft es, Dinge in Relation zu se
tzen – vom kleinsten Teilchen bis zum galaktischen Supercluster
Number of Posts: 33
Predicted Bounds: (33, 36]
Link: https://www.derstandard.at/story/3000000200558/das-universum-als-perspektivgeber


Incorrect Prediction: Klimawandel, Rechtsruck, Inflation: Die Sorgen der Jungen steigen weiter Zukunftsän
gste Serienweise Krisen, permanenterLeistungsdruck,das Streben nach Besonderem und der ständige Vergleich
über soziale Medien belasten junge Menschen zunehmend. Unbeschwert fühlen sich nur wenige. Aber es gibt a
uch Grund für Optimismus
Number of Posts: 1989
Predicted Bounds: (274, 296]
Link: https://www.derstandard.at/story/3000000201340/klimawandel-rechtsruck-inflation-die-sorgen-der-jung
en-steigen-weiter

Incorrect Prediction: Gebrauchtwagen-Besitzer könnten bald Abogebühren bezahlen Teure Zukunft Neue Autos
verrechnen für bestimmte Features mittlerweile monatliche Kosten. Nun droht demGebrauchtwagenmarktähnlich
es
Number of Posts: 1769
Predicted Bounds: (218, 235]
Link: https://www.derstandard.at/story/3000000201410/gebrauchtwagen-besitzer-koennten-bald-abogebuehren-b
ezahlen
```

I am actually kind of happy with this result. Of course the mean squared error of 217 on the test set is not particularly amazing (seeing as our values only have a range of 64), but the loss curve indicates that the model was actually able to learn some things about the data and was not just randomly guessing. It seems like there is some gradient in the vector space that can be learned.

Furthermore, the plot of articles on (31.12.23) shows us that our bar of predictions is actually pretty slim. With fewer than 64 buckets (maybe only 10), the predictions would probably hit the correct neighborhood more often. The problem here remains the distribution of posts. I have experimented with a smaller number of buckets, but as the vast majority of articles generates between 0 and 10 posts (as indicated by those buckets not actually being buckets but discrete values) - these low values would still overrepresented in the data and the cutoff to articles with higher post counts would be extremely early. The only alternative here would be to actually drop a lot of values from our test set - which is something that could be tested with more time.

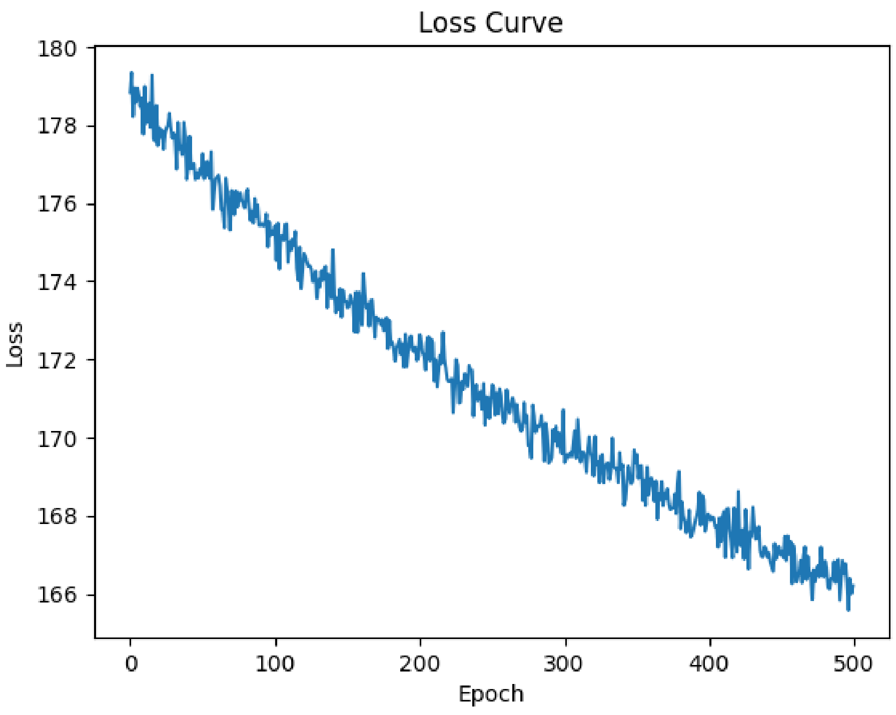
Maybe dropping the majority of low-postcount observations would also help with the overall performance of the model as the current distribution of observations might be lead the regression to trend lower than it should - as seen by the lineplot where the actual number of posts (red) trends much higher than the predictions past around ~250 posts.

Medium regression Model 'with a slimmer funnel'

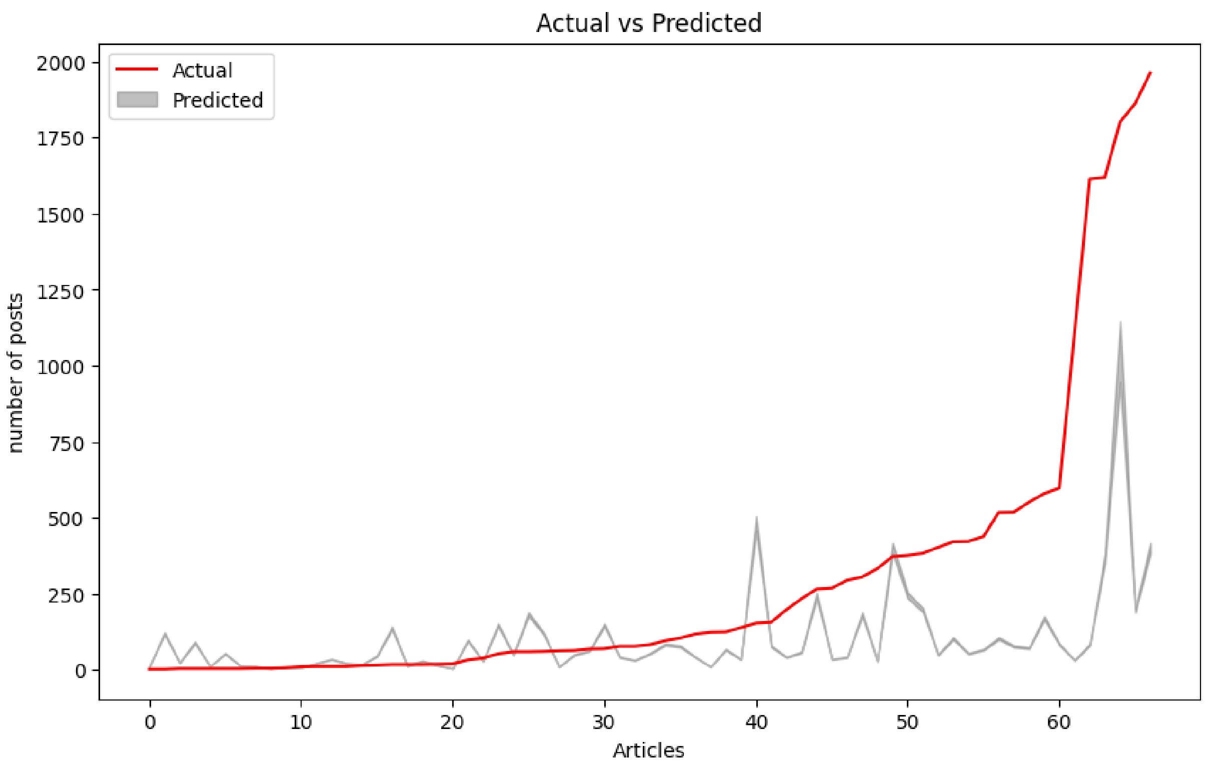
For the next experiment, we will append two final layers with 32 and 16 neurons respectively. Maybe the regression will work better with a longer funnel. To switch things up, we will be testing it on articles from (30.12.23)

In [11]:
```
medium_model = RegressionNN(
    drop = 0.3,
    layer_sizes = [303, 128, 64, 32, 16]
    ).to(my_device)

print('training medium model')
medium_losses = train(small_model, epochs=500, learning_rate=0.0001)
print('evaluating medium model')
eval_trained_model(small_model, medium_losses, test_date-timedelta(days=1))
```

```
training medium model
100%|██████████| 500/500 [1:03:27<00:00,  7.61s/it]
evaluating medium model
```



```
Test Loss: 220.54502462919524
```



```
Incorrect Prediction: Signa-Kollaps: Gusenbauer tritt als Millionen-Gläubiger auf Wirtschaft Laut Recherc
hen von "Profil" und der "Süddeutschen Zeitung" hat der Ex-Kanzler Alfred Gusenbauer (SPÖ) Forderungen in
Höhe von mehr als 6,3 Millionen Euro eingebracht
Number of Posts: 1862
Predicted Bounds: (189, 203]
Link: https://www.derstandard.at/story/3000000201381/signa-kollaps-chef-der-finanzprokuratur-sorgt-sich-u
m-staatsgeld

Incorrect Prediction: Frauen werden gleichgestellt, aber nicht gleich behandelt Petra Stuiber 2024 beginn
t die Anpassung des Pensionsalters an das der Männer. Sonst aber bleibt alles beim Alten
Number of Posts: 1963
Predicted Bounds: (382, 417]
Link: https://www.derstandard.at/story/3000000201355/frauen-werden-gleichgestellt-aber-nicht-gleich-behan
delt
```

Once more we need to accept that bigger is not necessarily better. In this case, extending our network by two smaller layers has not improved our test set performance at all, it is pretty much on par with the small model. Furthermore, for the 30.12., the model was not able to correctly categorize a single article :(

What is remarkable though is how different the loss curve looks this time. Instead of approaching some limit in a logarithmic fashion, the loss (on the training set) seems to decrease linearly. Also the loss starts at a much lower value, probably we just got lucky with the initialization of our weights. Normally this linear decrease should be a good sign that our model has still not found an optimum, but the test-set performance indicates that we were probably just overfitting to the training data (hard to call it overfitting with such a high loss).

We won't be discouraged by this, lets go one step bigger and make an even deeper network before we call it quits. This time we will still keep 64 neurons for the second-to-last layer.

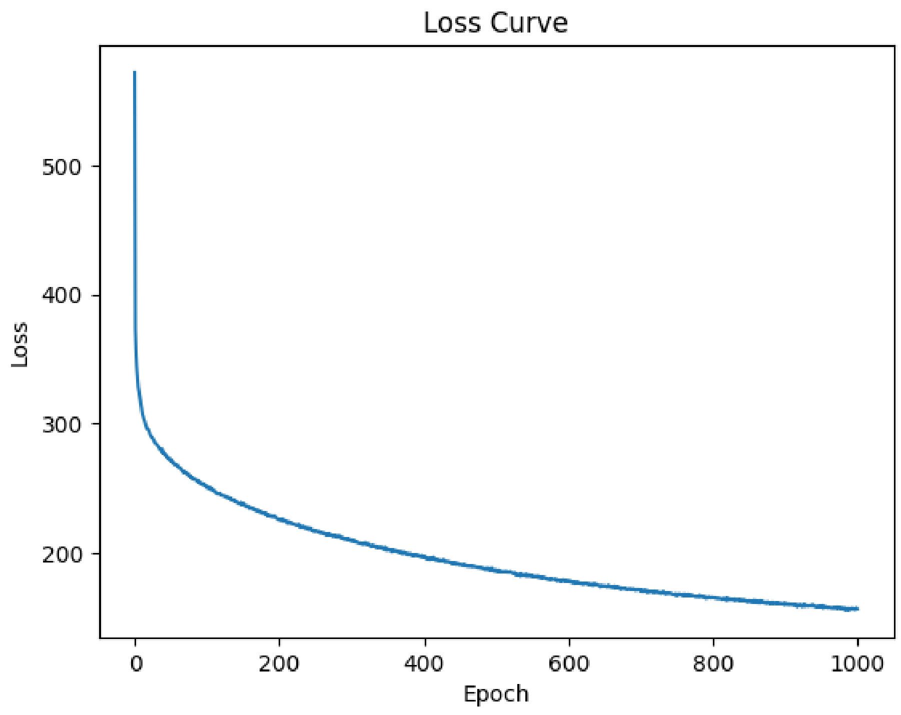
Big regression Model

8 hidden layers, decreasing in size. With more parameters, I have decided to increase the dropout probability to 40%, decrease the learning rate by a decimal place and train for 1.000 epochs.

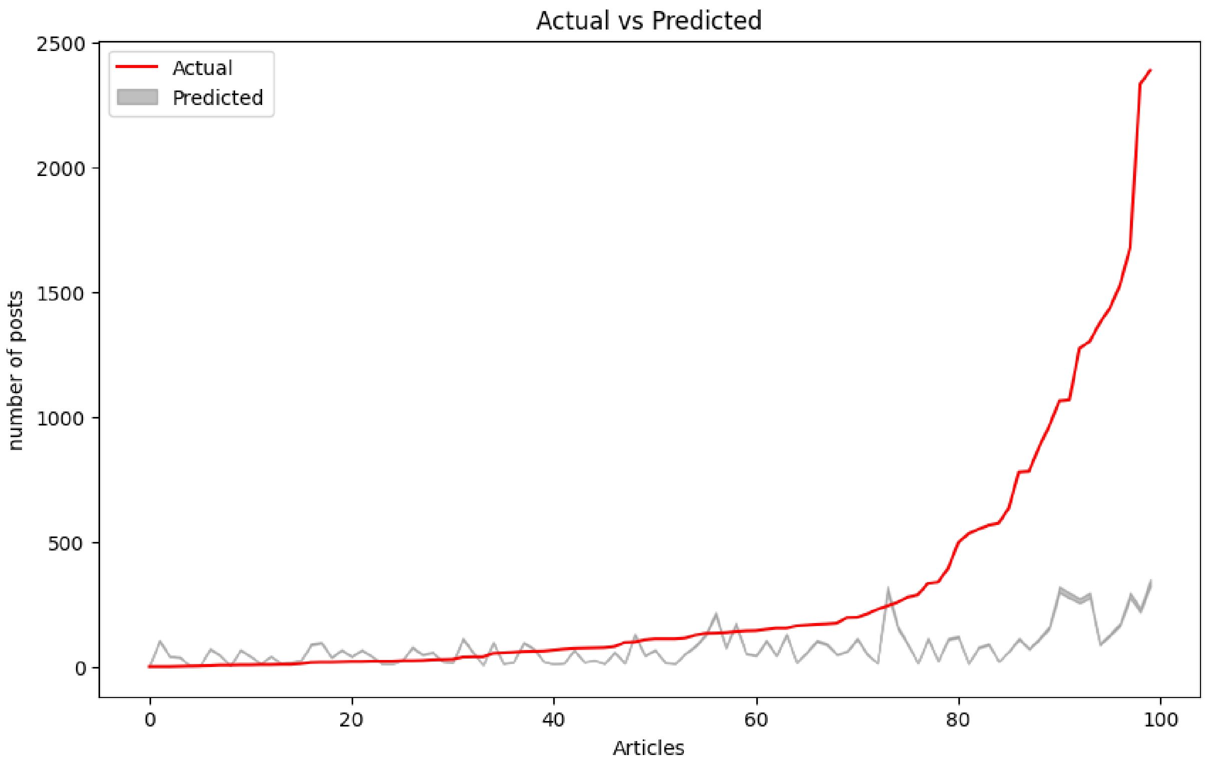
```
In [12]: big_model = RegressionNN(
             drop = 0.4,
             layer_sizes = [303, 303, 256, 256, 256, 128, 128, 64]
             ).to(my_device)
```

```python
print('training big model')
big_losses = train(big_model, epochs=1000, learning_rate=0.00001)
print('evaluating big model')
eval_trained_model(big_model, big_losses, test_date-timedelta(days=2))
```

```
training big model
100%|██████████| 1000/1000 [3:18:36<00:00, 11.92s/it]
evaluating big model
```



```
Test Loss: 267.1881389280933
```



```
Incorrect Prediction: Bereits 31 Tote nach massiver Angriffswelle auf die Ukraine Krieg in der Ukraine Un
ter anderem wurden die Städte Kiew, Charkiw, Dnipro, Lwiw und Odessa angegriffen. Die EU beteuerte trotz
ungarischen Widerstands ihre Unterstützung
Number of Posts: 2333
Predicted Bounds: (218, 235]
Link: https://www.derstandard.at/jetzt/livebericht/3000000201236/russische-angriffe-auf-staedte-charkiw-u
nd-lwiw

Incorrect Prediction: Gegen Woke und Wärmepumpen: Monika Gruber wettert in Büchern – und auf Demos Auswei
tung der Kampfzone Eine Bloggerin fühlt sich von Passagen des neuen Buches des Kabarett-Stars rassistisch
beleidigt. Die Bayerin wettert auch auf Bühnen gegen grünen, woken Fortschritt
Number of Posts: 2388
Predicted Bounds: (321, 350]
Link: https://www.derstandard.at/story/3000000201329/gegen-woke-und-waermepumpen-monika-gruber-wettert-in
-buechern--und-auf-demos
```

Even worse performance than the medium model on the test set. I have a feeling that the vector representation does not capture the message content well enough.

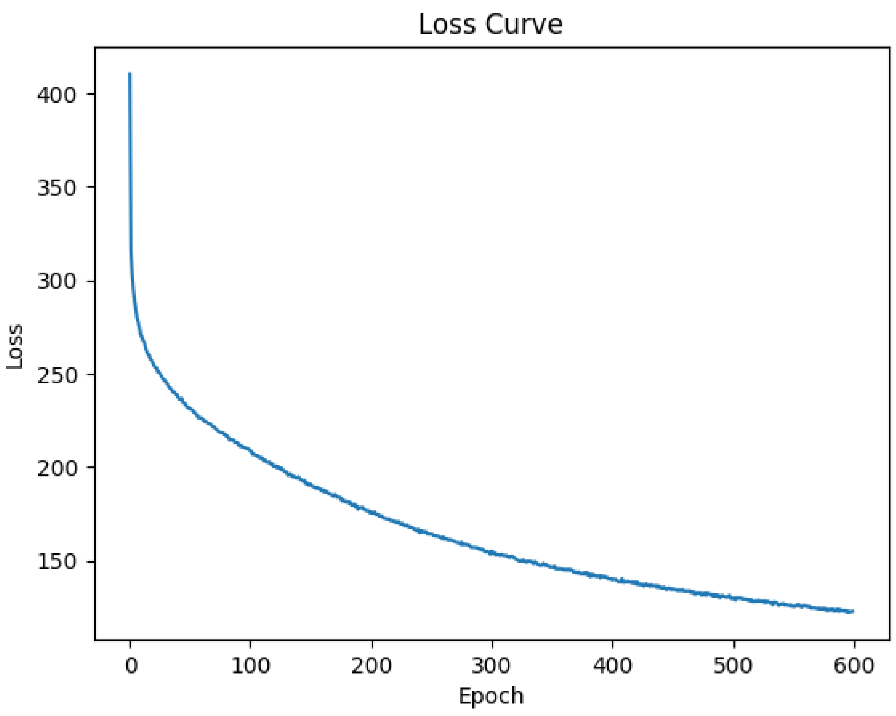
We will perform one last experiment. Scaling our neural networks hidden layers to be wider than the input.

Wide regression Model

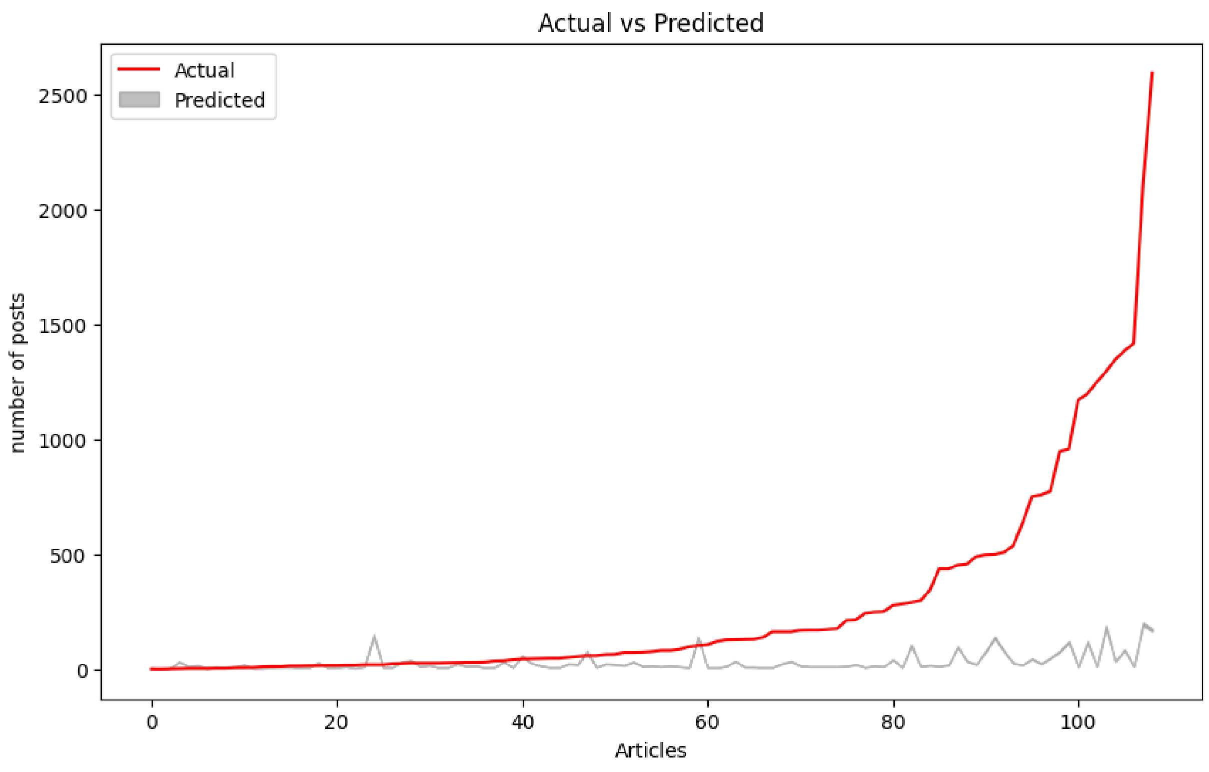
Here we go both wide and deep. This model should have over 3 million parameters. Because the last run was overfitting so terribly, I decided to only train this network for 600 epochs, with the same learning rate as before and increased dropout to 50%.

In [13]:
```python
wide_model = RegressionNN(
    drop = 0.5,
    layer_sizes = [303, 606, 909, 909, 909, 606, 303]
    ).to(my_device)

print('training wide model')
wide_losses = train(wide_model, epochs=600, learning_rate=0.00001)
print('evaluating wide model')
eval_trained_model(wide_model, wide_losses, test_date-timedelta(days=3))
```

```
training wide model
100%|██████████| 600/600 [2:49:28<00:00, 16.95s/it]
evaluating wide model
```



```
Test Loss: 496.87058120565786
```



```
Correct Prediction: Web- und Games-News: Ein Onlyfans-Model gegen die NASA Nachrichtenüberblick Das sind
die aktuellen Schlagzeilen aus Web und Games
Number of Posts: 0
Predicted Bounds: (0, 1]
Link: https://www.derstandard.at/story/3000000201208/web-und-games-news-das-comeback-eines-baby-ichhoernc
hens

Correct Prediction: Kreuzworträtsel I 10574 Kreuzworträtsel Täglich neu, exklusiv fürSmart-Abonnent:inne
n:Das kniffligephoenixen-Rätseldes STANDARD
Number of Posts: 3
Predicted Bounds: (3, 4]
Link: https://www.derstandard.at/story/3000000200358/kreuzwortraetsel-i-10574


Incorrect Prediction: Deutschland sichert Ukraine weitere EU-Finanzhilfe zu Krieg in der Ukraine Russland
hat ukrainischen Angaben zufolge das Land in der Nacht erneut mit Drohnen angegriffen. Zwei russische Dic
hter erhielten wegen kritischen Gedichts zum Ukrainekrieg lange Haftstrafen
Number of Posts: 2594
Predicted Bounds: (164, 176]
Link: https://www.derstandard.at/jetzt/livebericht/3000000201130/erneut-nacht-drohnen-angriff-russland

Incorrect Prediction: Später in Pension – Fortschritt oder Falle für Frauen? Gleichberechtigung Ab dem ne
uen Jahr steigt das Pensionsalter der Frauen auf 65 Jahre wie bei den Männern. Das könnte eineWin-win-win
-Situationwerden – aber auch viele zu Verliererinnen machen
Number of Posts: 2104
Predicted Bounds: (189, 203]
Link: https://www.derstandard.at/story/3000000200635/spaeter-in-pension-fortschritt-oder-falle-fuer-fraue
n
```

A test set loss that is almost 2.5x that of the small model. Not very promising signs to just scale the network.

Conclusion

There may be countless reasons why this project was not successful in reliably predicting post counts.

In all fairness, I think this is a really difficult problem. Ultimately it should be impossible to predict how many people will open a given article and feel compelled to add their 5 cents to the discussion. However as Isaac Asimov's psychohistory postulates: '*the laws of statistics as applied to large groups of people could predict the general flow of future events*', so we decided to give it a shot anyways.

Possible next steps

With even more time, I would try encoding the data differently. Maybe using an RNN architecture, feeding in embedding vectors one at a time. Or possibly instead of using Word2Vec, we could apply a transformer to get our embeddings for the whole set.

Perhaps traditional machine learning techniques could also work, maybe a simple clustering of the current vectors.

I am sceptical that simply changing the activation functions would do much, I experimented a bit with leaky ReLu but found no improvements.

As already mentioned, maybe filtering the data would prevent the network from trending so low for all posts.

Saving the 'best' model

To finish things up, we save the network that was most successful on the test data - the small model. It will also be available for download in the onedrive link

In [18]:
```
torch.save(small_model.state_dict(), 'data/small_model.pth')
```

This notebook processes the scraped data, converts text columns to numeric vectors (Part 1) and trains classification models using the processed text data as predictors.

Locally on Richard's computer, the notebook takes approximately 1.5 hrs to execute completely - Part 1 needs ~45 mins, Part 2 and 3 ~20 min each.

Besides usual numpy, pandas etc., needed modules are spacy and pytorch. The German NLP model needs to be downloaded separately.

Edit:

As we worked separately, the first part, text processing and conversion to embeddings was also done separately. Ideally, the processing would be replaced by a single script that creates processed file imported in all notebooks. The reason, why we wre unable to do this is the high cost, Richard's computer is unable to ork with the larger models.

Contents

Part 1: Text Processing

Here we import scraped data, splits text column to words, extracts and normalizes tokens. Next, tokens are converted to embeddings.

In [1]:
```
### Setup

# pip install spacy
# python -m spacy download de_core_news_sm  - for small model (13 MB)
# python -m spacy download de_core_news_md  - for medium model (42 MB)
# source - https://spacy.io/models/de


import spacy
from spacy.lang.de.examples import sentences


# Example from documentation
nlp = spacy.load("de_core_news_sm")
doc = nlp(sentences[0])
print(doc.text)
for token in doc:
    print(token.text, token.pos_, token.dep_)
```

```
Die ganze Stadt ist ein Startup: Shenzhen ist das Silicon Valley für Hardware-Firmen
Die DET nk
ganze ADJ nk
Stadt NOUN sb
ist AUX ROOT
ein DET nk
Startup NOUN pd
: PUNCT punct
Shenzhen NOUN sb
ist AUX cj
das DET nk
Silicon PROPN pnc
Valley PROPN sb
für ADP mnr
Hardware-Firmen NOUN nk
```

In [2]:
```
## Data loading

import numpy as np
import pandas as pd

standard_data = pd.read_csv('./data/4yrs_derstandard_frontpage_data.csv')
standard_data
```

Out[2]:

	title	subtitle	link	datetime	kicker	n_posts	sto
0	Real Madrid stolpert mit Aluminiumpech im Tite...	Die Königlichen können Bilbao daheim nicht bes...	https://www.derstandard.at/story/2000112599363...	2019-12-22T23:44	Primera Division	30	
1	Bolivien weist venezolanische Diplomaten aus	InterimspräsidentinJeanine Áñez wirft denBotsc...	https://www.derstandard.at/story/2000112598924...	2019-12-22T22:50	Übergangsregierung	16	
2	Erdoğan warnt vor neuer Flüchtlingswelle aus S...	Türkischer Präsident: "80.000 Menschen Richtun...	https://www.derstandard.at/story/2000112598130...	2019-12-22T21:43	Bürgerkrieg	104	
3	Massenkarambolage mit 63 Fahrzeugen in Virginia	Autos stießen auf vereister Brücke zusammen	https://www.derstandard.at/story/2000112597972...	2019-12-22T21:29	Weihnachtsverkehr	35	
4	Salzburg schlägt Caps, Meister KAC mit vierter...	Die Bullen sind damit der Gewinner der Runde: ...	https://www.derstandard.at/story/2000112595206...	2019-12-22T20:54	Eishockey	4	
...	...	...	...	...	...	...	
182102	Wer braucht die Kirche?	Dass sich die Kirche nach soschwerwiegendenVer...	https://www.derstandard.at/story/3000000200743...	2023-12-22T06:00	Dominik Straub	1	Ko
182103	Sonderregelung verlängert: Mehr als 1.000 Ärzt...	Der"Pandemieparagraf"im Ärztegesetz hat mehr a...	https://www.derstandard.at/story/3000000200621...	2023-12-22T06:00	Pandemieparagraf	148	
182104	Stadtforscher: "Architektur ist Teil unserer W...	Jetzt anhören: In Zukunft müssen Städte wieder...	https://www.derstandard.at/story/3000000200499...	2023-12-22T06:00	Edition Zukunft	54	
182105	David Alaba zum zehnten Mal zu Österreichs Fuß...	Zehn von zwölf Trainern wählten den derzeit ve...	https://www.derstandard.at/story/3000000200745...	2023-12-22T05:46	Fußball	34	
182106	Kreuzworträtsel F 10571	Täglich neu, exklusiv fürSmart-Abonnent:innen:...	https://www.derstandard.at/story/3000000199163...	2023-12-22T00:01	Kreuzworträtsel	4	

182107 rows × 7 columns

Tokenizing + lemmatizing, embeddings

First, title and subtitile columns are split into individual words. Second, words are replaced by their standard forms (großem - groß, rettete - retten). Finally, the standard forms are converted to numeric vectors using toc2vec embeddings.

In [3]:
```python
# Initializing two columns
standard_data['title_tokens'] = standard_data['title']
standard_data['subtitle_tokens'] = standard_data['subtitle']

standard_data['title_vectors'] = standard_data['title_tokens']
standard_data['subtitle_vectors'] = standard_data['subtitle_tokens']

# Load model
nlp = spacy.load("de_core_news_sm")
nlp.get_pipe('lemmatizer')
nlp.get_pipe("tok2vec")
```

Out[3]:
```
<spacy.pipeline.tok2vec.Tok2Vec at 0x17e1a8cbcb0>
```

In [4]:
```python
# Loops through rows

for index, row in standard_data.iterrows():

    # TITLE

    text = nlp(row['title_tokens'].replace("-", " ").replace(",", " ").replace(": ", " "))

    token_list = ' '.join([token.lemma_ for token in text]) # list of standardized tokens joined to a string

    vector_list = [token.vector for token in nlp(token_list)] # list of numeric vectors

    # store in the dataframe

    standard_data.at[index, 'title_tokens'] = token_list
    standard_data.at[index, 'title_vectors'] = vector_list
    # SAME FOR SUBTITLE

    # try except because subtitle is sometimes empty, then it would output error

    try:

        text = nlp(row['subtitle_tokens'])
        token_list = ' '.join([token.lemma_ for token in text]) # list of standardized tokens joined to a string
        vector_list = [token.vector for token in nlp(token_list)] # list of numeric vectors

    except:
        token_list = '' # if empty, token list empty
        vector_list = []

    standard_data.at[index, 'subtitle_tokens'] = token_list # store in the dataframe
    standard_data.at[index, 'subtitle_vectors'] = vector_list
```

In [5]:
```
standard_data.head(4)
```

Out[5]:

	title	subtitle	link	datetime	kicker	n_posts	storylab
0	Real Madrid stolpert mit Aluminiumpech im Tite...	Die Königlichen können Bilbao daheim nicht bes...	https://www.derstandard.at/story/2000112599363...	2019-12-22T23:44	Primera Division	30	Na
1	Bolivien weist venezolanische Diplomaten aus	InterimspräsidentinJeanine Áñez wirft denBotsc...	https://www.derstandard.at/story/2000112598924...	2019-12-22T22:50	Übergangsregierung	16	Na
2	Erdoğan warnt vor neuer Flüchtlingswelle aus S...	Türkischer Präsident: "80.000 Menschen Richtun...	https://www.derstandard.at/story/2000112598130...	2019-12-22T21:43	Bürgerkrieg	104	Na
3	Massenkarambolage mit 63 Fahrzeugen in Virginia	Autos stießen auf vereister Brücke zusammen	https://www.derstandard.at/story/2000112597972...	2019-12-22T21:29	Weihnachtsverkehr	35	Na

In [6]:
```
# Example of list of numeric vectors representing a subtitle

standard_data.iloc[1,10]
```

```
Out[6]: [array([ 1.1021554 ,  1.5151364 , -0.9405582 , -2.410718  , -1.1881657 ,
        -3.615175  ,  1.0234319 ,  4.1538424 , -1.5936106 , -2.785506  ,
         0.6842209 ,  0.07389921,  0.5360052 ,  0.31706142, -0.9196383 ,
        -0.82573223, -3.5580513 , -1.8286765 ,  0.10499629, -2.6490862 ,
         0.7699168 ,  0.9488483 ,  3.0474105 , -3.499149  , -1.3851355 ,
        -1.3532616 , -2.1080291 ,  1.4907815 ,  1.6701344 ,  0.32809997,
         3.3263462 ,  1.2118888 , -0.66293633, -0.6889895 , -0.9815758 ,
         7.8648634 , -2.7158377 ,  1.1486918 ,  1.389757  ,  2.5390139 ,
        -2.2970562 ,  1.9853625 , -0.53672844,  1.7880895 ,  1.0695057 ,
        -1.396502  , -0.44181645,  8.145238  , -6.583647  ,  2.6661644 ,
        -3.5900345 ,  5.988452  ,  0.7246865 ,  0.7114573 , -0.97419953,
        -0.59691894, -1.2567514 ,  1.227717  , -0.20339483,  1.2106575 ,
        -3.6484938 , -0.32804242, -0.22830245, -2.9978542 , -1.7434182 ,
         1.2077783 ,  1.3858008 , -2.3194008 ,  2.691605  , -1.2882311 ,
         0.36779276,  6.1339602 ,  5.0458665 ,  2.7170045 , -0.73983014,
        -1.4936604 , -3.691854  , -0.58311963, -2.758136  ,  1.3663645 ,
         0.7641036 , -0.25958115,  0.19299927, -0.36475182, -2.15046   ,
        -0.07376699,  1.3615923 ,  0.15364465,  0.43697536, -0.7795961 ,
        -0.2906142 , -1.3341428 , -1.1321028 ,  0.7692802 ,  4.4434543 ,
        -0.13221759], dtype=float32),
 array([ 2.0387900e-01, -3.6389467e-01,  9.1594264e-02,  3.0110793e+00,
         1.9099593e-01,  2.5002155e+00,  4.7174902e+00,  3.2845149e+00,
         2.4571249e+00, -3.9303410e-01,  2.4632785e+00,  3.5243196e+00,
         8.2635641e-02, -3.7615123e-01, -2.4238696e+00,  9.4931561e-01,
        -3.1734276e+00, -3.2585377e-01, -9.5212698e-02, -3.6530802e+00,
         2.6839254e+00,  3.8944585e+00,  2.8477988e+00,  9.7264910e-01,
        -3.4695044e+00, -1.8469485e+00, -1.1298934e+00, -1.7287492e+00,
        -1.7891788e+00, -1.2972705e+00, -4.0097499e+00, -1.6719009e+00,
        -2.6558471e-01,  4.8970847e+00, -1.6694819e+00,  2.8998227e+00,
         3.6650777e-02,  2.3830295e-02, -4.0626961e-01, -1.1825575e+00,
         4.5469692e-01,  7.1066767e-02,  1.1704326e-01,  2.2460222e+00,
        -2.2601731e+00, -2.4881077e-01,  5.1106195e+00, -7.0074636e-01,
         2.4542422e+00,  4.2827673e+00, -3.7211795e+00,  4.0359521e+00,
         1.0617015e+00, -1.5808861e+00,  2.8038654e+00,  2.0369215e+00,
        -2.3406210e+00,  1.6063156e+00,  1.2375314e+00, -3.9418101e+00,
        -3.1209874e+00,  5.3786767e-01, -7.0941699e-01, -8.3923918e-01,
         1.9914703e+00, -5.0472933e-01,  1.6171703e+00, -2.4083114e+00,
         1.1187898e+00,  1.1807323e+00,  3.7856526e+00, -3.2749174e+00,
         5.1386099e+00,  1.0498765e+00, -2.7171969e-03, -2.5394652e+00,
        -1.9865291e+00, -2.3191524e+00, -3.2128429e-01, -2.3197007e+00,
        -7.2254086e-01, -1.9952973e+00, -1.2792372e+00, -2.8368850e+00,
        -2.9223857e+00, -5.3396523e-01, -2.3760524e+00, -9.1870689e-01,
        -2.1093471e+00,  3.1699508e-01, -1.6469510e+00, -1.2593009e+00,
         2.1777494e-01,  4.9522644e-01,  1.0336641e+00,  2.7116790e-01],
       dtype=float32),
 array([-0.83021116,  1.497284  , -0.9227109 ,  1.802495  ,  2.49502   ,
         0.3021891 ,  1.4051473 ,  4.7342453 , -1.3231409 ,  0.7152008 ,
         1.3497535 , -0.68772495, -1.1351818 , -2.8685963 ,  0.19471988,
         1.866097  , -0.659291  , -2.4652512 , -3.6585813 , -1.970429  ,
         0.6091137 , -0.5410261 , -1.093161  , -0.93193674, -2.1877618 ,
        -0.34253335,  4.331459  ,  2.406736  , -1.5875481 , -1.2403198 ,
         0.3342938 ,  0.8339085 , -1.4179932 ,  2.086682  , -0.2932942 ,
         0.7463709 ,  1.54034   , -2.472604  , 10.581617  ,  1.2359589 ,
        -1.5131515 ,  5.729193  ,  3.1587148 , -2.8493028 ,  0.4284364 ,
        -1.7156467 , -1.3902086 , -0.8821014 ,  1.0003353 , -0.22503006,
        -3.058124  ,  1.2050474 ,  0.63327134, -3.7192645 ,  1.9260386 ,
         1.5659823 , -0.15807867,  0.9381081 , -2.2680752 ,  2.4811857 ,
         0.6327281 ,  2.806039  , -0.7336584 , -2.7372947 ,  1.6763592 ,
        -2.0626225 , -1.2927849 , -3.0047374 ,  2.9425187 ,  2.2545786 ,
        -5.1119127 ,  1.6616504 , -1.0005181 ,  0.44565445, -1.792507  ,
         1.9223206 , -1.522932  ,  1.6897063 , -0.14903557, -0.6437911 ,
        -3.3589764 , -1.3296492 ,  1.4645389 , -0.55227613,  0.96184725,
         4.250011  ,  0.8981428 , -3.2711775 ,  0.94647956, -2.0579686 ,
        -0.07125318, -1.0319518 ,  1.4655437 ,  0.73916465, -2.4736276 ,
        -2.4883735 ], dtype=float32),
 array([ 0.4268612 ,  0.11444321, -3.5183058 , -0.4342844 ,  0.51640785,
        -1.5647693 ,  2.8943887 ,  1.4340794 , -2.0140646 , -1.3430088 ,
        -0.60300016,  0.67714226,  0.73942363,  1.4505097 , -3.9787164 ,
        -1.6661923 ,  1.1756002 , -0.01579541, -1.4690632 , -4.521759  ,
         2.9466097 ,  0.6662991 ,  1.0837626 ,  3.397083  , -0.5540648 ,
        -0.9863453 ,  2.3200371 , -1.8510854 ,  1.4576211 , -0.9045007 ,
         0.96012497,  3.4490943 ,  3.8648047 , -0.08552533,  0.7867209 ,
        -0.08240172,  1.1028842 , -0.6129677 ,  0.56159705,  1.0788395 ,
         1.9978597 ,  0.7594922 , -0.02069485, -0.19586757,  3.6035194 ,
         2.6952    , -2.8366916 ,  1.1927326 ,  0.09903134, -3.0415542 ,
        -3.2372315 ,  2.9434052 ,  2.5885506 ,  3.2774305 ,  4.2413726 ,
        -3.212422  ,  3.0751781 ,  2.4350133 , -0.49164814,  0.11654021,
         4.640836  , -1.9098933 ,  0.22600174, -3.6791887 , -0.14912975,
         1.8631192 , -2.7025967 , -1.8053133 , -2.6565313 , -0.6381764 ,
        -0.61491895, -1.0311294 , -1.1377723 ,  3.2797484 , -2.7812965 ,
        -0.1707015 , -2.4011383 , -0.63554215, -2.9577036 , -2.9310107 ,
        -0.5228375 ,  0.47581732, -0.25083572,  1.7707635 , -0.58195096,
         1.2655902 , -0.68389213,  0.69857156, -2.7911675 ,  0.70694065,
        -3.957206  , -0.22527076,  3.1920824 , -1.3613758 ,  1.5655717 ,
        -1.1629789 ], dtype=float32),
 array([ 2.858104  , -0.4259818 ,  0.0533731 , -3.2954707 , -0.14817905,
         0.05753967,  1.0969156 , -0.34743267,  0.88959074,  0.5335532 ,
         1.0946918 , -2.226666  ,  0.514783  ,  0.73557144, -1.1123455 ,
         2.187058  ,  0.50760436, -0.5147372 ,  1.2438729 , -0.04412994,
         0.05263746,  1.4446843 ,  0.2778504 ,  1.341911  , -3.6685023 ,
         0.9865285 ,  1.3624797 , -0.0123758 , -2.691949  ,  4.033491  ,
        -0.56697977,  0.4989763 , -0.21788037,  4.2447743 , -1.2398992 ,
         1.0839074 , -2.477867  , -1.7621161 ,  1.8945113 ,  2.3866873 ,
        -2.59024   , -0.96871006, -0.33251345, -0.12506872,  4.552087  ,
         2.8513644 ,  0.41111633,  0.51917094,  0.02366787, -0.5274012 ,
        -2.7292676 , -2.0452867 ,  4.2623873 ,  0.3875122 , -0.8248315 ,
        -0.32203722, -1.1037903 , -3.0828161 ,  2.2062783 , -2.6692753 ,
        -2.0157883 , -1.058212  ,  1.3724246 , -1.9377549 , -2.2691765 ,
        -0.9465351 , -0.53197896, -1.8847213 , -3.646227  ,  2.8391743 ,
        -1.6809651 , -0.92708457, -1.2322042 ,  1.6195769 , -0.23415822,
         2.8377922 , -1.9586563 , -1.9909892 , -2.0246377 , -2.9430902 ,
        -1.6619481 ,  1.4801362 , -0.16447723,  1.5505334 ,  0.4808987 ,
         4.3188734 ,  0.46803015, -0.8805185 ,  0.80877286,  2.0628033 ,
         1.3530104 ,  3.0936437 ,  0.5739267 ,  1.0531263 , -2.2153168 ,
         1.1867065 ], dtype=float32),
 array([-3.8055253 , -3.5115027 , -1.0059981 , -3.6148624 , -0.45541   ,
         2.3010263 , -3.123283  ,  3.9836278 ,  3.305799  ,  3.148072  ,
        -1.4811625 , -3.2537093 , -2.457415  ,  1.2686638 , -0.13613607,
         2.6724596 ,  1.2367618 ,  3.622371  ,  3.260768  , -1.2518926 ,
        -1.6727774 ,  3.5320773 , -1.8793987 ,  1.3768798 , -2.7413633 ,
        -0.6037619 , -0.06114875, -0.48323965,  0.9015295 ,  2.0272603 ,
        -2.2566009 ,  0.5120633 , -1.2588562 ,  1.9404176 ,  3.8674512 ,
        -0.04215217, -2.9890358 ,  2.0556245 ,  0.22212723,  1.749618  ,
         0.33235598,  0.03336096,  1.8797287 ,  4.888695  , -0.01029098,
        -1.3796326 ,  1.8553174 , -1.8387939 , -1.1648893 ,  1.3825982 ,
        -2.707078  ,  3.1952686 ,  2.238944  , -2.453817  ,  1.5905297 ,
        -0.2021507 , -0.10128054, -1.0571215 ,  2.0745199 , -1.4071221 ,
        -3.8053017 ,  0.4611584 , -2.3974686 , -1.6399364 ,  2.0876296 ,
         0.16720504,  2.3801744 ,  1.154994  , -2.763418  , -0.97302604,
         3.1014602 , -1.7898778 ,  4.876471  , -2.9201698 ,  2.3444037 ,
        -4.772825  , -2.9049087 , -1.5523417 , -4.17309   ,  1.2448903 ,
        -1.6100448 ,  0.65438884, -0.91534674, -1.5658777 ,  1.8594563 ,
         0.8504069 , -1.5365098 ,  2.7906036 ,  0.7174556 , -0.85909116,
         2.2371955 ,  2.434044  , -1.1921433 , -0.3468234 ,  1.2002109 ,
        -2.4800344 ], dtype=float32)]
```

Defining predictors and dependent variable

In [7]:
```python
### sum vectors for words  to get a vector for a sentence

import numpy as np

standard_data_copy = standard_data.copy()

for index,row in standard_data_copy.iterrows():

    l = standard_data_copy.loc[index, 'title_vectors']
    standard_data_copy.at[index, 'title_vectors'] = np.sum(l, axis=0)

    l = standard_data_copy.loc[index, 'subtitle_vectors']
    standard_data_copy.at[index, 'subtitle_vectors'] = np.sum(l, axis=0)
```

In [8]:
```python
# reshaping

X = np.vstack(standard_data_copy['title_vectors'].to_numpy())
X[0:2]
```

```
Out[8]: array([[  9.119195  ,   0.4420882 ,  -6.585746  ,  -0.36528832,
            5.3427124 ,   8.7853775 ,  -5.9323215 ,  15.100472  ,
            8.413376  ,   0.498165  ,  15.515087  , -10.902954  ,
          -14.050985  ,   6.649028  , -11.754989  ,   6.057748  ,
            8.1888    ,  -2.3931763 ,  -3.9362216 ,  -2.4156115 ,
            1.1384144 ,  16.5233    ,  -3.1429567 ,  12.609358  ,
          -10.458169  , -14.882816  ,  -7.4182734 ,   2.4116597 ,
           -5.4921722 ,   7.045065  , -10.576113  ,  -2.9996653 ,
           -2.5590808 ,  15.348734  ,  17.18261   ,   6.2615204 ,
            1.5978868 ,   4.3599935 ,   6.792941  ,   0.34123358,
           -4.33022   ,   1.9403619 ,  -4.0108805 ,  -3.0241973 ,
            2.9737525 ,   5.95477   ,  -1.0211641 ,  -9.582537  ,
           16.537086  ,  -1.5229907 , -17.975307  ,  14.044141  ,
           17.179005  , -15.048843  ,  -3.5073388 ,  -9.0553465 ,
            7.744524  ,  -1.2955302 ,  -1.1938403 ,  -0.9916693 ,
           -5.2311893 , -15.524109  ,  -1.8139477 ,  -5.236508  ,
           -7.388012  ,   1.6616129 ,   9.769495  ,  -3.0856996 ,
            5.103663  , -10.281055  ,  -8.181546  ,  11.605755  ,
           20.072994  ,   1.9509575 ,   1.0104284 ,   2.7451582 ,
           -6.7651024 , -17.938444  ,  -0.29718518,  -4.189915  ,
            0.49740887,  -1.7963977 ,   1.3374608 ,   3.804048  ,
            0.82973343,  -4.5802813 ,  -3.792004  ,   0.59529376,
           -4.4479127 ,   3.0961576 ,   0.09678364,   6.9876165 ,
            3.4458823 ,   5.249668  ,  -5.503358  ,   2.4459732 ],
        [  4.1763725 ,   6.6053157 ,  -6.26212   ,  -1.8468281 ,
            5.8947444 , -11.055052  ,  16.409658  ,   4.4481072 ,
           10.801826  ,   2.6705813 ,  19.742989  , -10.982428  ,
           -7.0191803 ,   1.378798  ,  -2.3377242 ,   3.6979463 ,
            0.8893782 ,  -0.24522638,  -2.0405407 ,   7.9179482 ,
           -7.7598577 ,   5.657601  ,   8.710294  ,   7.591489  ,
           -7.517646  ,  -9.373699  ,  12.201945  ,   1.6120387 ,
           -4.4405456 ,   2.064506  ,  -4.025571  ,  -1.2569398 ,
            9.275347  ,   3.774525  ,  -2.4166684 ,   5.925064  ,
           -4.9106    ,   6.250013  ,   3.1558099 ,   5.1071405 ,
            0.76229167,  12.146876  ,  -2.4513674 ,  -0.679394  ,
           -1.4179453 ,   2.5416732 ,  -4.7745566 ,  -8.512492  ,
            4.8051677 ,  -6.053112  , -11.54631   ,   0.36491406,
            0.8190367 ,  -6.9885855 ,  -1.56673   ,  -4.1197243 ,
            2.7763567 ,   5.4589043 ,  -1.6369283 ,  -1.204077  ,
           -4.955944  ,  -0.5450032 , -10.904157  ,  -0.17724037,
           -5.962854  ,   4.716673  ,  -0.22160816,  -7.6610374 ,
           -6.0942597 ,   5.3322835 ,  -3.6568193 ,  -4.372014  ,
            9.467594  ,   1.2784152 ,  -5.0787563 ,   1.5037313 ,
           -7.411691  ,  -7.2856708 ,   3.2065206 ,  -6.4576864 ,
           -3.1081047 ,  -3.3992252 ,  -0.32932377,   0.11005902,
           -2.4886897 ,   9.941364  ,   6.774614  ,  -4.425606  ,
           -0.5213309 ,   3.925714  ,   4.3876085 ,   0.7395493 ,
            1.7669291 ,   6.883208  ,  -5.6511483 ,  -1.0476801 ]],
      dtype=float32)
```

Part 2: Binary classification

```
In [9]: # split y to above and under 50 posts (regression -> classification)

        y = (standard_data_copy['n_posts'].to_numpy() > 50).astype(int)
        y = np.column_stack(standard_data_copy['n_posts'].to_numpy() > 50).astype(int).T

        print(y.shape)
        print(X.shape)

        (182107, 1)
        (182107, 96)
```

```
In [10]: # train-test split

         from sklearn.model_selection import train_test_split


         X_train, X_test, Y_train, Y_test = train_test_split(X, y,
                                                             test_size=0.2,
                                                             random_state=42069)
```

```
In [11]: import numpy as np
         import torch
         import torch.nn as nn
         import torch.optim as optim
```

```
In [12]: X_train = torch.tensor(X_train, dtype=torch.float32)
         X_test = torch.tensor(X_test, dtype=torch.float32)
```

```
Y_train = torch.tensor(Y_train, dtype=torch.float32).reshape(-1, 1)
Y_test = torch.tensor(Y_test, dtype=torch.float32).reshape(-1, 1)
```

In [13]:
```
# two hidden layers, each with 12 nodes

class BinaryClassifier(nn.Module):
    def __init__(self, n_inputs = 96):
        super().__init__()
        self.hidden1 = nn.Linear(n_inputs, 12)
        self.act1 = nn.ReLU()
        self.hidden2 = nn.Linear(12, 12)
        self.act2 = nn.ReLU()
        self.output = nn.Linear(12, 1)
        self.act_output = nn.Sigmoid()

    def forward(self, x):
        x = self.act1(self.hidden1(x))
        x = self.act2(self.hidden2(x))
        x = self.act_output(self.output(x))
        return x

loss_fn = nn.L1Loss(size_average=None, reduce=None, reduction='mean')
```

In [14]:
```
# training the model

import random

def train_binary_model(X_train, Y_train, n_inputs = 96, n_epochs = 5000):

    model = BinaryClassifier(n_inputs)
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    random.seed(42069)
    batch_size = int(len(X_train) / 50)
    print(batch_size)
    for epoch in range(n_epochs):
        for i in range(0, len(X_train), batch_size):
            Xbatch = X_train[i:i+batch_size]
            y_pred = model(Xbatch)
            ybatch = Y_train[i:i+batch_size]
            loss = loss_fn(y_pred, ybatch)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        if epoch % 100 == 0:
            baseline = sum(ybatch) / len(ybatch)
            accuracy = sum(ybatch == y_pred.round())/len(ybatch)
            print(f'Finished epoch {epoch}, latest loss {loss}, accuracy {accuracy} vs baseline {baseline}')

    return model
```

In [15]:
```
# testing the model

def test_binary_model(model, X_test):
    Y_hat = model(X_test).round()
    print("Accuracy: ")
    print((sum(Y_test == Y_hat)/len(Y_test)).item())
    print("Baseline accuracy: ")
    print((sum(Y_test)/len(Y_test)).item())
```

In [16]:
```
model = train_binary_model(X_train, Y_train, n_epochs=1000)
```

```
2913
Finished epoch 0, latest loss 0.28019288182258606, accuracy tensor([0.7143]) vs baseline tensor([0.6000])
Finished epoch 100, latest loss 0.11489197611808777, accuracy tensor([0.8857]) vs baseline tensor([0.6000])
Finished epoch 200, latest loss 0.08579690009355545, accuracy tensor([0.9143]) vs baseline tensor([0.6000])
Finished epoch 300, latest loss 0.08585256338119507, accuracy tensor([0.9143]) vs baseline tensor([0.6000])
Finished epoch 400, latest loss 0.08591507375240326, accuracy tensor([0.9143]) vs baseline tensor([0.6000])
Finished epoch 500, latest loss 0.08572710305452347, accuracy tensor([0.9143]) vs baseline tensor([0.6000])
Finished epoch 600, latest loss 0.08575011789798737, accuracy tensor([0.9143]) vs baseline tensor([0.6000])
Finished epoch 700, latest loss 0.08571437746286392, accuracy tensor([0.9143]) vs baseline tensor([0.6000])
Finished epoch 800, latest loss 0.08571577072143555, accuracy tensor([0.9143]) vs baseline tensor([0.6000])
Finished epoch 900, latest loss 0.08582895994186401, accuracy tensor([0.9143]) vs baseline tensor([0.6000])
```

In [17]:
```
test_binary_model(model, X_test)
```

```
Accuracy: 
0.6365109086036682
Baseline accuracy: 
0.5655098557472229
```

Can we get better accuracy using subtitles?

Here, we try using subtitle instead of article title as the predictor

In [18]:
```
for i in range(len(standard_data_copy)):

    if standard_data_copy.loc[i, 'subtitle_tokens'] == '':
        standard_data_copy.at[i, 'subtitle_vectors'] = np.zeros(96)
```

In [19]:
```
X = np.vstack(standard_data_copy['subtitle_vectors'].to_numpy())
X

X_train, X_test, Y_train, Y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42069)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)

Y_train = torch.tensor(Y_train, dtype=torch.float32).reshape(-1, 1)
Y_test = torch.tensor(Y_test, dtype=torch.float32).reshape(-1, 1)
```

In [20]:
```
model = train_binary_model(X_train, Y_train, n_epochs = 1000)
```

```
2913
Finished epoch 0, latest loss 0.368115097284317, accuracy tensor([0.6571]) vs baseline tensor([0.6000])
Finished epoch 100, latest loss 0.20157773792743683, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 200, latest loss 0.2004449963569641, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 300, latest loss 0.20048309862613678, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 400, latest loss 0.20007279515266418, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 500, latest loss 0.20005889236927032, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 600, latest loss 0.20005641877651215, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 700, latest loss 0.2000395506620407, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 800, latest loss 0.20004263520240784, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 900, latest loss 0.20002888143062592, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
```

In [21]:
```
test_binary_model(model, X_test)
```

```
Accuracy:
0.6548514366149902
Baseline accuracy:
0.5655098557472229
```

We can see, using article's subtitle yields worse imporvement in accuracy. Next, we try using both

In [22]:
```
X = np.concatenate((np.vstack(standard_data_copy['title_vectors'].to_numpy()), np.vstack(standard_data_copy['subtitle_vec
print(X.shape)

X_train, X_test, Y_train, Y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42069)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)

Y_train = torch.tensor(Y_train, dtype=torch.float32).reshape(-1, 1)
Y_test = torch.tensor(Y_test, dtype=torch.float32).reshape(-1, 1)
```

```
(182107, 192)
```

In [23]:
```
model = train_binary_model(X_train, Y_train, n_inputs = 192, n_epochs = 1000)
```

```
2913
Finished epoch 0, latest loss 0.29562562704086304, accuracy tensor([0.7143]) vs baseline tensor([0.6000])
Finished epoch 100, latest loss 0.20029419660568237, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 200, latest loss 0.20000003278255463, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 300, latest loss 0.200010460615158, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 400, latest loss 0.2000003308057785, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 500, latest loss 0.20000004768371582, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 600, latest loss 0.20000001788139343, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 700, latest loss 0.20000068843364716, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 800, latest loss 0.20000000298023224, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
Finished epoch 900, latest loss 0.20000000298023224, accuracy tensor([0.8000]) vs baseline tensor([0.6000])
```

In [24]:
```
test_binary_model(model, X_test)
```

```
Accuracy:
0.6726703643798828
Baseline accuracy:
0.5655098557472229
```

Part 3: Multi-class classifier

We also want to try out a classifier for variables with more than two categories. The predictors are going to be the same, for dependent variables we will choose columns *kicker* and *storylabels*

3.1: Kickers

In [25]:
```
# 20 most common kickers

standard_data_copy['kicker'].value_counts()[0:20]
```

Out[25]:
```
kicker
Fußball                     3162
Nachrichtenüberblick        3101
Netzpolitik                 2674
Sudoku                      2414
Bundesliga                  1775
Sport                       1684
USA                         1518
IT-Business                 1464
Coronavirus                 1464
Games                       1356
Tennis                      1252
Switchlist                  1208
Krieg in der Ukraine        1203
Deutsche Bundesliga         1180
Etat-Überblick              1161
Hans Rauscher               1153
Wintersport                 1134
TV-Tagebuch                 1080
Eishockey                   1058
Thema des Tages             1032
Name: count, dtype: int64
```

In [26]:
```
# filtering for the most common ones

kickers_20 = standard_data_copy['kicker'].value_counts().nlargest(20).index.tolist()

standard_kickers = standard_data_copy.copy()
standard_kickers = standard_kickers[standard_kickers['kicker'].isin(kickers_20)].reset_index()[['kicker', 'title_vectors'

standard_kickers.head(3)
```

Out[26]:

	kicker	title_vectors	subtitle_vectors
0	Eishockey	[16.587378, -5.880515, -10.134394, -2.7442758,...	[17.787502, -9.519611, 6.9725924, -22.601917, ...
1	Deutsche Bundesliga	[5.977871, -8.565644, -8.99353, 8.525143, 13.7...	[30.480652, -4.33595, -10.554278, -1.181915, 1...
2	Sport	[6.727174, -2.1885931, -6.3502436, -9.411136, ...	[4.372181, -6.303278, -5.9711304, 3.729673, 8....

In [27]:
```
# dependent variable: bianry vector of length 20

y = np.zeros((len(standard_kickers), 20))
y.shape

for row in range(y.shape[0]):
    y_row = [1 if k == standard_kickers['kicker'][row] else 0 for k in kickers_20]
    y[row,] = y_row

y[0:2]
```

Out[27]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
        0., 0., 0., 0.]])
```

In [28]:
```
X_title = np.vstack(standard_kickers['title_vectors'].to_numpy())
X_title[0:2]

X_subtitle = np.vstack(standard_kickers['subtitle_vectors'].to_numpy())
X_subtitle[0:2]
```

```
Out[28]: array([[ 17.78750229,  -9.51961136,   6.97259235, -22.60191727,
                  23.74225616,   9.31873894,   8.5572834 ,  23.8451519 ,
                  10.47311592,   0.22695971,  34.2782402 , -30.66071892,
                 -22.11200523,  -2.05303741, -20.84233093,  24.89317513,
                  20.07475853,  42.06415558, -25.77451897,  -8.23214912,
                  23.6224556 ,  20.40205765,  24.19214058,   2.76441216,
                 -18.69178772, -16.48721504,  19.79255867,  -3.66987586,
                  40.11144257,  12.80308914, -32.58789825,  -8.74606419,
                   4.88409948,   1.47553456,  16.15632629, -12.9812727 ,
                   1.87224245,  -2.28196573,  20.60498428,   2.65613198,
                 -31.96606445,  53.01169968,  10.38889885,  -8.28358364,
                  42.46792603,   4.36738205, -20.69274902, -27.77854729,
                 -12.88485432,  -8.18120575, -25.95684433,  11.9279747 ,
                  -2.15959144,   5.14111853,  18.90647316, -12.25464344,
                  20.79118919,  24.50359917, -17.3789978 , -15.80044937,
                 -12.89952755, -20.38439178,  -5.98169279,   1.39122534,
                 -20.21481514,   1.05093873,  16.25543022, -30.86683083,
                   2.52105093,  18.34233665, -33.49952698,  25.06282616,
                  47.10434723, -23.10667419, -39.30410767, -18.42495155,
                  -8.50583649,  -4.53000689,  13.55102825, -21.09018326,
                 -14.89650345,  -8.21998405, -23.88335419,  19.44838142,
                  -3.64663601,  14.58350945, -11.82317543,  13.01065826,
                   1.91321766,  23.72675323,  21.42988396,  35.87226486,
                  12.88493824,   7.5420599 , -20.29470253,  20.51945877],
                [ 30.48065186,  -4.3359499 , -10.55427837,  -1.18191504,
                  18.83841705,  17.63163757,  30.53222275,  20.03815842,
                   3.39736319,   2.30487394,  25.3445015 ,  -7.26597357,
                  -4.58202076, -14.59606552, -19.73396683,   9.31867218,
                  -9.59283638,  21.59188461,   1.12145662,   1.50997066,
                  23.1015892 ,  33.62531281,  17.11453629,  19.28170586,
                 -13.00364304, -14.07565594,   2.0197711 , -16.49940872,
                  11.45446682,   6.29826355, -22.30833817,  -0.72283781,
                  15.60744572,  19.16539383,  16.84324837,  -3.12985754,
                   1.29834294,   8.66239357,  35.42202377,  -7.05598974,
                 -29.72544479,  30.3167038 ,   4.89661646,   9.35298061,
                  19.87916946,  16.34698486, -11.17651176, -14.47218132,
                  16.27173615,  12.55965996, -38.4762001 ,  18.57327461,
                  11.58078003, -18.65499115,  15.47526932, -11.61715317,
                  13.75895691,  15.42659092,   5.01880741, -11.45995712,
                 -21.76966667, -25.96429634, -25.58132172, -41.12456894,
                  -3.47233582,   7.15661573,  18.72220612, -31.13856506,
                   6.04983616,  -4.88566685,  -3.87553883,  18.30246544,
                  28.63316536,  -4.63203001,  -3.52348185,  -9.22648239,
                 -33.90135574, -24.38999176,   7.14297867, -12.18540764,
                 -31.93712425,  -0.31258076,  -1.26990783,   0.40502286,
                   4.4830904 ,  10.65467453,  -2.97914696,  -4.40925884,
                 -21.99129105,   7.50469494,  13.47981453,  -5.81780672,
                  -1.40452564,   1.18488801, -13.77711582,  -2.79524326]])
```

```
In [29]: X_title_train, X_title_test, Y_train, Y_test = train_test_split(X_title, y,
                                                                  test_size=0.25,
                                                                  random_state=42069)
```

```
In [30]: X_subtitle_train, X_subtitle_test, Y_train, Y_test = train_test_split(X_subtitle, y,
                                                                  test_size=0.25,
                                                                  random_state=42069)
```

```
In [31]: X_title_train = torch.tensor(X_title_train, dtype=torch.float32)
         X_title_test = torch.tensor(X_title_test, dtype=torch.float32)

         X_subtitle_train = torch.tensor(X_subtitle_train, dtype=torch.float32)
         X_subtitle_test = torch.tensor(X_subtitle_test, dtype=torch.float32)

         Y_train = torch.tensor(Y_train, dtype=torch.float32)
         Y_test = torch.tensor(Y_test, dtype=torch.float32)
```

```
In [32]: class ClassifierTwentyCategories(nn.Module):
             def __init__(self, n_inputs):
                 super().__init__()
                 self.hidden1 = nn.Linear(n_inputs, 48)
                 self.act1 = nn.ReLU()
                 self.hidden2 = nn.Linear(48, 24)
                 self.act2 = nn.ReLU()
                 self.output = nn.Linear(24, 20)
                 self.act_output = nn.Sigmoid()

             def forward(self, x):
                 x = self.act1(self.hidden1(x))
                 x = self.act2(self.hidden2(x))
                 x = self.act_output(self.output(x))
                 return x
```

```
loss_fn = nn.CrossEntropyLoss()
```

In [33]:
```
def train_multiclass_model(X_train, Y_train, n_inputs = 96, n_epochs = 1000):

    random.seed(42069)

    model = ClassifierTwentyCategories(n_inputs)
    optimizer = optim.Adam(model.parameters(), lr=0.001)


    batch_size = int(len(X_train) / 19)
    print(batch_size)

    for epoch in range(n_epochs):
        for i in range(0, len(X_train), batch_size):
            Xbatch = X_train[i:i+batch_size]
            y_pred = model(Xbatch)
            ybatch = Y_train[i:i+batch_size]
            loss = loss_fn(y_pred, ybatch)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        if epoch % 100 == 0:
            y_pred_category = [torch.argmax(r).item() for r in y_pred]
            ybatch_category = [torch.argmax(r).item() for r in ybatch]
            accuracy = sum([1 if x == y else 0 for x, y in zip(y_pred_category, ybatch_category)]) / len(ybatch_category)
            print(f'Finished epoch {epoch}, latest loss {loss}, accuracy {accuracy}')

    return model
```

In [34]:
```
def test_multiclass_model(model, X_test, Y_test, categories = kickers_20):

    p = [torch.argmax(r).item() for r in model(X_test)]
    Y_hat = [categories[i] for i in p]

    p = [torch.argmax(r).item() for r in Y_test]

    Y_test_values = [categories[i] for i in p]

    acc = sum([1 if x == y else 0 for x, y in zip(Y_hat, Y_test_values)]) / len(Y_test_values)
    print("Accuracy: " + str(acc))
```

In [35]:
```
model_titles = train_multiclass_model(X_title_train, Y_train, n_epochs = 1000)
```

```
1266
Finished epoch 0, latest loss 2.870898723602295, accuracy 0.23617693522906794
Finished epoch 100, latest loss 2.3970587253570557, accuracy 0.4794628751974723
Finished epoch 200, latest loss 2.3890132904052734, accuracy 0.466824644549763
Finished epoch 300, latest loss 2.3703079223632812, accuracy 0.4565560821484992
Finished epoch 400, latest loss 2.3584561347961426, accuracy 0.4565560821484992
Finished epoch 500, latest loss 2.347475528717041, accuracy 0.45734597156398105
Finished epoch 600, latest loss 2.3382813930511475, accuracy 0.4510268562401264
Finished epoch 700, latest loss 2.338529348373413, accuracy 0.4518167456556082
Finished epoch 800, latest loss 2.335376024246216, accuracy 0.44944707740916273
Finished epoch 900, latest loss 2.3318440914154053, accuracy 0.4565560821484992
```

In [36]:
```
test_multiclass_model(model_titles, X_title_test, Y_test)
```

```
Accuracy: 0.409652076318743
```

In [37]:
```
model_subtitles = train_multiclass_model(X_subtitle_train, Y_train, n_epochs = 1000)
```

```
1266
Finished epoch 0, latest loss 2.843013048171997, accuracy 0.06398104265402843
Finished epoch 100, latest loss 2.4316508769989014, accuracy 0.4146919431279621
Finished epoch 200, latest loss 2.4045395851135254, accuracy 0.4028436018957346
Finished epoch 300, latest loss 2.3934378623962402, accuracy 0.39257503949447076
Finished epoch 400, latest loss 2.39837384223938, accuracy 0.39968404423380727
Finished epoch 500, latest loss 2.4003560543060303, accuracy 0.38704581358609796
Finished epoch 600, latest loss 2.383061170578003, accuracy 0.3941548183254344
Finished epoch 700, latest loss 2.377591609954834, accuracy 0.3902053712480253
Finished epoch 800, latest loss 2.364919662475586, accuracy 0.4028436018957346
Finished epoch 900, latest loss 2.372812271118164, accuracy 0.3933649289099526
```

In [38]:
```
test_multiclass_model(model_subtitles, X_subtitle_test, Y_test)
```

```
Accuracy: 0.3559047262750967
```

In [39]:
```
X = np.concatenate((np.vstack(standard_kickers['title_vectors'].to_numpy()), np.vstack(standard_kickers['subtitle_vectors
print(X.shape)
print(y.shape)
X_train, X_test, Y_train, Y_test = train_test_split(X, y,
                                                    test_size=0.25,
                                                    random_state=42069)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)

Y_train = torch.tensor(Y_train, dtype=torch.float32)
Y_test = torch.tensor(Y_test, dtype=torch.float32)
```

```
(32073, 192)
(32073, 20)
```

In [40]:
```
combined_model = train_multiclass_model(X_train, Y_train, n_inputs=192, n_epochs=1000)
```

```
1266
Finished epoch 0, latest loss 2.7735064029693604, accuracy 0.2401263823064771
Finished epoch 100, latest loss 2.3024654388427734, accuracy 0.5387045813586098
Finished epoch 200, latest loss 2.3085975646972656, accuracy 0.5039494470774092
Finished epoch 300, latest loss 2.280428886413574, accuracy 0.5521327014218009
Finished epoch 400, latest loss 2.2779200077056885, accuracy 0.5284360189573459
Finished epoch 500, latest loss 2.256657361984253, accuracy 0.5387045813586098
Finished epoch 600, latest loss 2.2502005100250244, accuracy 0.5402843601895735
Finished epoch 700, latest loss 2.2486824989318848, accuracy 0.5521327014218009
Finished epoch 800, latest loss 2.2432501316070557, accuracy 0.5481832543443917
Finished epoch 900, latest loss 2.238225221633911, accuracy 0.556872037914692
```

In [41]:
```
test_multiclass_model(combined_model, X_test, Y_test)
```

```
Accuracy: 0.4732510288065844
```

Part 3.2 : Storylabels

In [42]:
```
# filtering for the most common ones

storylabels_20 = standard_data_copy['storylabels'].value_counts().nlargest(20).index.tolist()

standard_storylabels = standard_data_copy.copy()
standard_storylabels = standard_storylabels[standard_storylabels['storylabels'].isin(storylabels_20)].reset_index()[['sto

standard_storylabels.head(3)
```

Out[42]:

	storylabels	title_vectors	subtitle_vectors
0	Kopf des Tages	[20.26102, -2.1764362, 12.056327, -8.353496, 2...	[12.673783, -13.09248, 8.577187, -2.7328134, 9...
1	Spiel	[-3.334474, -4.604315, -2.6970067, -2.6541345,...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
2	Spiel	[-0.60618734, 1.1090474, 3.5244317, -7.726671,...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...

In [43]:
```
# dependent variable: bianry vector of length 20

y = np.zeros((len(standard_storylabels), 20))
print(y.shape)

for row in range(y.shape[0]):
    y_row = [1 if s == standard_storylabels['storylabels'][row] else 0 for s in storylabels_20]
    y[row,] = y_row

y[0:2]
```

```
(36407, 20)
```

Out[43]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
        0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0.]])
```

In [44]:
```
X_title = np.vstack(standard_storylabels['title_vectors'].to_numpy())
X_title[0:2]

X_subtitle = np.vstack(standard_storylabels['subtitle_vectors'].to_numpy())
X_subtitle[0:2]
```

```
Out[44]: array([[ 12.6737833 , -13.09247971,   8.57718658,  -2.73281336,
           9.55943489,   1.19442225,  13.35463333,  18.54079056,
          -1.24317193, -16.44563866,  26.11481094, -23.8042469 ,
         -19.78061295,  -1.20370197, -14.3854847 ,  21.38131714,
          11.98632336,  11.43702793,  -8.80769539,   5.8185854 ,
           9.41383076,  19.68308258,  12.65031338,   6.34458733,
         -14.87839794, -12.42210007,  14.90303516,   5.76547098,
           8.85460186,  13.4050703 , -14.50532913, -11.64571476,
           6.06785011,   7.40806866,  18.20200348,   0.66248214,
          -7.48345232,  -6.37721109,  30.55085945,  -2.74134064,
         -18.50937462,  46.69207764,  14.90998268,  -1.3300662 ,
          21.63993645,   9.69990635, -10.60776997, -12.7449646 ,
          -7.96562004,  -2.69942284, -28.52653694,  -3.66213107,
           3.77998972,   4.08382893,  10.05634022,  -5.16197395,
          14.41419888,   7.13106537, -17.49291039,  -9.09768009,
         -14.40098476,  -5.38680267, -15.8886013 , -10.017313  ,
          -6.79507494,  12.58858109,  -5.66467857, -18.49414444,
           1.37763619,  21.84117126, -10.74147129,   8.79939842,
          23.5548439 ,  -6.04022217, -20.06015778,  -2.71913767,
         -12.75486469,  -4.30656385,  -2.91261292, -17.43518829,
          -6.46806765,   1.19207692, -11.48882484,  -0.17394084,
          10.08629131,  16.67332077,  -6.58500385,   7.27330875,
         -13.24323082,  31.27229691,  18.21424294,   0.77637053,
           3.49404359,  14.81214142, -18.95225525,  -3.14274883],
        [  0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ,
           0.        ,   0.        ,   0.        ,   0.        ]])
```

```
In [45]: X_title_train, X_title_test, Y_train, Y_test = train_test_split(X_title, y,
                                                                         test_size=0.25,
                                                                         random_state=42069)
```

```
In [46]: X_subtitle_train, X_subtitle_test, Y_train, Y_test = train_test_split(X_subtitle, y,
                                                                               test_size=0.25,
                                                                               random_state=42069)
```

```
In [47]: X_title_train = torch.tensor(X_title_train, dtype=torch.float32)
         X_title_test = torch.tensor(X_title_test, dtype=torch.float32)

         X_subtitle_train = torch.tensor(X_subtitle_train, dtype=torch.float32)
         X_subtitle_test = torch.tensor(X_subtitle_test, dtype=torch.float32)

         Y_train = torch.tensor(Y_train, dtype=torch.float32)
         Y_test = torch.tensor(Y_test, dtype=torch.float32)
```

```
In [48]: model_titles = train_multiclass_model(X_title_train, Y_train, n_epochs = 1000)
```

```
1437
Finished epoch 0, latest loss 2.7086448669433594, accuracy 0.0
Finished epoch 100, latest loss 2.190015544281006, accuracy 0.0
Finished epoch 200, latest loss 2.189687728881836, accuracy 0.0
Finished epoch 300, latest loss 2.189687490463257, accuracy 0.0
Finished epoch 400, latest loss 2.18967342376709, accuracy 0.0
Finished epoch 500, latest loss 2.189673900604248, accuracy 0.0
Finished epoch 600, latest loss 2.189673900604248, accuracy 0.0
Finished epoch 700, latest loss 2.189673900604248, accuracy 0.0
Finished epoch 800, latest loss 2.189673900604248, accuracy 0.0
Finished epoch 900, latest loss 2.189673900604248, accuracy 0.0
```

```
In [49]: test_multiclass_model(model_titles, X_title_test, Y_test)
```

```
Accuracy: 0.346736980883322 33
```

```
In [50]: model_subtitles = train_multiclass_model(X_subtitle_train, Y_train, n_epochs = 1000)
```

```
1437
Finished epoch 0, latest loss 2.792396068572998, accuracy 0.0
Finished epoch 100, latest loss 2.294781446456909, accuracy 0.0
Finished epoch 200, latest loss 2.2581024169921875, accuracy 0.0
Finished epoch 300, latest loss 2.258202075958252, accuracy 0.0
Finished epoch 400, latest loss 2.25809383392334, accuracy 0.0
Finished epoch 500, latest loss 2.258094072341919, accuracy 0.0
Finished epoch 600, latest loss 2.258089065551758, accuracy 0.0
Finished epoch 700, latest loss 2.2580909729003906, accuracy 0.0
Finished epoch 800, latest loss 2.2580976486206055, accuracy 0.0
Finished epoch 900, latest loss 2.258089065551758, accuracy 0.0
```

```
In [51]: test_multiclass_model(model_subtitles, X_subtitle_test, Y_test)
```

```
Accuracy: 0.32179740716326083
```

```
In [52]: X = np.concatenate((np.vstack(standard_storylabels['title_vectors'].to_numpy()), np.vstack(standard_storylabels['subtitle
print(X.shape)
print(y.shape)
X_train, X_test, Y_train, Y_test = train_test_split(X, y,
                                                    test_size=0.25,
                                                    random_state=42069)

X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)

Y_train = torch.tensor(Y_train, dtype=torch.float32)
Y_test = torch.tensor(Y_test, dtype=torch.float32)
```

```
(36407, 192)
(36407, 20)
```

```
In [53]: combined_model = train_multiclass_model(X_train, Y_train, n_inputs=192, n_epochs=1000)
```

```
1437
Finished epoch 0, latest loss 2.785167694091797, accuracy 0.0
Finished epoch 100, latest loss 2.1517343521118164, accuracy 0.5
Finished epoch 200, latest loss 2.1516098976135254, accuracy 0.5
Finished epoch 300, latest loss 2.1516027450561523, accuracy 0.5
Finished epoch 400, latest loss 2.1516025066375732, accuracy 0.5
Finished epoch 500, latest loss 2.151602268218994, accuracy 0.5
Finished epoch 600, latest loss 2.116236448287964, accuracy 0.5
Finished epoch 700, latest loss 2.1162257194519043, accuracy 0.5
Finished epoch 800, latest loss 2.1162257194519043, accuracy 0.5
Finished epoch 900, latest loss 2.1162257194519043, accuracy 0.5
```

```
In [54]: test_multiclass_model(combined_model, X_test, Y_test)
```

```
Accuracy: 0.43331136014062843
```

Summary

We have processed the scraped data and trained classification models. The word2vec encoding of article title and subtitle is able to improve upon baseline accuracy on test data in both binary and multi-class classification tasks. However, there is still a large room for improvement. This could be achieved by "bruteforcing" the models - more layers, more nodes, more epochs etc., by using a more sofisticated network architecture such as RNN, or by using superior encodings - such as larger models included in the spacy module.

References

https://machinelearningmastery.com/develop-your-first-neural-network-with-pytorch-step-by-step/

http://mitloehner.com/lehre/dsai1/

https://pytorch.org/docs/stable/nn.html

https://spacy.io/models/de