

# Predicting Virality with Extreme Gradient Boosting on Online News Popularity Data

---

Ednalyn C. De Dios  
D214  
September 7, 2023  
Western Governors University

---

Data Science Pipeline (PAPEM-DM):

- Planning
- Acquisition of Data
- Preparation of Data
- Exploration of Data
- Modeling
- Delivery
- Maintenance

## PLANNING

Get the data. Prepare the data. Conduct EDA. Split the data. Prepare data for modeling. Create initial xgboost model. Make predictions on train and test data. Evaluate performance. Tune hyperparameters. Create final xgboost model. Make predictions on train and test data. Evaluate performance. Extract feature importance. - *De Dios (2020)*

## ACQUISITION

Before we can get the data, let's first import the packages that we're going to need.

```
In [1]: # setting the random seed for reproducibility
import random
random.seed(493)

# for manipulating dataframes
import pandas as pd
import numpy as np

# for statistical testing
from scipy import stats
```

```

# for modeling
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import KFold
from sklearn import metrics
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer
import xgboost as xgb
from xgboost import XGBClassifier
import shap

# for visualizations
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")

# to print out all the outputs
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# set display options
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)

# print the JS visualization code to the notebook
shap.initjs()

```

Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to force console mode (e.g. in jupyter console)



In [2]: THRESHOLD = 1400  
ALPHA = 0.05

Let's read the raw dataset which can be downloaded from the UCI Machine Learning Repository  
<https://archive.ics.uci.edu/dataset/332/online+news+popularity>

In [3]: # Read a csv file  
df = pd.read\_csv('../data/in/OnlineNewsPopularity.csv')

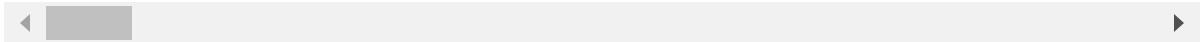
# PREPARATION

Let's take a peek and familiarize ourselves with the structure and contents of the dataset.

```
In [4]: df.head()  
df.info()  
df.shape
```

Out[4]:

		url	timedelta	n_tokens_title	n_tokens_content	n_l
0		http://mashable.com/2013/01/07/amazon-instant-video-browser/	731.0	12.0	219.0	
1		http://mashable.com/2013/01/07/app-samsung-sponsored-tweets/	731.0	9.0	255.0	
2		http://mashable.com/2013/01/07/apple-40-billion-app-downloads/	731.0	9.0	211.0	
3		http://mashable.com/2013/01/07/astronaut-notre-dame-bcs/	731.0	9.0	531.0	
4		http://mashable.com/2013/01/07/att-universe-apps/	731.0	13.0	1072.0	



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39644 entries, 0 to 39643
Data columns (total 61 columns):
 #   Column           Non-Null Count Dtype  
 ---  -- 
 0   url              39644 non-null  object  
 1   timedelta        39644 non-null  float64 
 2   n_tokens_title  39644 non-null  float64 
 3   n_tokens_content 39644 non-null  float64 
 4   n_unique_tokens  39644 non-null  float64 
 5   n_non_stop_words 39644 non-null  float64 
 6   n_non_stop_unique_tokens 39644 non-null  float64 
 7   num_hrefs        39644 non-null  float64 
 8   num_self_hrefs  39644 non-null  float64 
 9   num_imgs          39644 non-null  float64 
 10  num_videos       39644 non-null  float64 
 11  average_token_length 39644 non-null  float64 
 12  num_keywords     39644 non-null  float64 
 13  data_channel_is_lifestyle 39644 non-null  float64 
 14  data_channel_is_entertainment 39644 non-null  float64 
 15  data_channel_is_bus    39644 non-null  float64 
 16  data_channel_is_socmed 39644 non-null  float64 
 17  data_channel_is_tech  39644 non-null  float64 
 18  data_channel_is_world 39644 non-null  float64 
 19  kw_min_min      39644 non-null  float64 
 20  kw_max_min      39644 non-null  float64 
 21  kw_avg_min      39644 non-null  float64 
 22  kw_min_max      39644 non-null  float64 
 23  kw_max_max      39644 non-null  float64 
 24  kw_avg_max      39644 non-null  float64 
 25  kw_min_avg      39644 non-null  float64 
 26  kw_max_avg      39644 non-null  float64 
 27  kw_avg_avg      39644 non-null  float64 
 28  self_reference_min_shares 39644 non-null  float64 
 29  self_reference_max_shares 39644 non-null  float64 
 30  self_reference_avg_shares 39644 non-null  float64 
 31  weekday_is_monday 39644 non-null  float64 
 32  weekday_is_tuesday 39644 non-null  float64 
 33  weekday_is_wednesday 39644 non-null  float64 
 34  weekday_is_thursday 39644 non-null  float64 
 35  weekday_is_friday   39644 non-null  float64 
 36  weekday_is_saturday 39644 non-null  float64 
 37  weekday_is_sunday   39644 non-null  float64 
 38  is_weekend        39644 non-null  float64 
 39  LDA_00            39644 non-null  float64 
 40  LDA_01            39644 non-null  float64 
 41  LDA_02            39644 non-null  float64 
 42  LDA_03            39644 non-null  float64 
 43  LDA_04            39644 non-null  float64 
 44  global_subjectivity 39644 non-null  float64 
 45  global_sentiment_polarity 39644 non-null  float64 
 46  global_rate_positive_words 39644 non-null  float64 
 47  global_rate_negative_words 39644 non-null  float64 
 48  rate_positive_words 39644 non-null  float64 
 49  rate_negative_words 39644 non-null  float64 
 50  avg_positive_polarity 39644 non-null  float64
```

```
51 min_positive_polarity      39644 non-null float64
52 max_positive_polarity     39644 non-null float64
53 avg_negative_polarity    39644 non-null float64
54 min_negative_polarity    39644 non-null float64
55 max_negative_polarity    39644 non-null float64
56 title_subjectivity        39644 non-null float64
57 title_sentiment_polarity  39644 non-null float64
58 abs_title_subjectivity   39644 non-null float64
59 abs_title_sentiment_polarity 39644 non-null float64
60 shares                    39644 non-null int64
dtypes: float64(59), int64(1), object(1)
memory usage: 18.5+ MB
```

Out[4]: (39644, 61)

Let's see if there's any missing values anywhere.

```
In [5]: def show_missing(df):
    """
    Takes a dataframe and returns a dataframe with stats
    on missing and null values with their percentages.
    """
    null_count = df.isnull().sum()
    null_percentage = (null_count / df.shape[0]) * 100
    empty_count = pd.Series(((df == ' ') | (df == '')).sum())
    empty_percentage = (empty_count / df.shape[0]) * 100
    nan_count = pd.Series(((df == 'nan') | (df == 'NaN')).sum())
    nan_percentage = (nan_count / df.shape[0]) * 100
    dfx = pd.DataFrame({'num_missing': null_count, 'missing_percentage': null_perce
                           'num_empty': empty_count, 'empty_percentage': empty_percen
                           'nan_count': nan_count, 'nan_percentage': nan_percentage})
    return dfx

show_missing(df)
```

Out[5]:

	num_missing	missing_percentage	num_empty	empty_perce
<b>url</b>	0	0.0	0	0
<b>timedelta</b>	0	0.0	0	0
<b>n_tokens_title</b>	0	0.0	0	0
<b>n_tokens_content</b>	0	0.0	0	0
<b>n_unique_tokens</b>	0	0.0	0	0
<b>n_non_stop_words</b>	0	0.0	0	0
<b>n_non_stop_unique_tokens</b>	0	0.0	0	0
<b>num_hrefs</b>	0	0.0	0	0
<b>num_self_hrefs</b>	0	0.0	0	0
<b>num_imgs</b>	0	0.0	0	0
<b>num_videos</b>	0	0.0	0	0
<b>average_token_length</b>	0	0.0	0	0
<b>num_keywords</b>	0	0.0	0	0
<b>data_channel_is_lifestyle</b>	0	0.0	0	0
<b>data_channel_is_entertainment</b>	0	0.0	0	0
<b>data_channel_is_bus</b>	0	0.0	0	0
<b>data_channel_is_socmed</b>	0	0.0	0	0
<b>data_channel_is_tech</b>	0	0.0	0	0
<b>data_channel_is_world</b>	0	0.0	0	0
<b>kw_min_min</b>	0	0.0	0	0
<b>kw_max_min</b>	0	0.0	0	0
<b>kw_avg_min</b>	0	0.0	0	0
<b>kw_min_max</b>	0	0.0	0	0
<b>kw_max_max</b>	0	0.0	0	0
<b>kw_avg_max</b>	0	0.0	0	0
<b>kw_min_avg</b>	0	0.0	0	0
<b>kw_max_avg</b>	0	0.0	0	0
<b>kw_avg_avg</b>	0	0.0	0	0
<b>self_reference_min_shares</b>	0	0.0	0	0
<b>self_reference_max_shares</b>	0	0.0	0	0

	num_missing	missing_percentage	num_empty	empty_perce
<b>self_reference_avg_sharess</b>	0	0.0	0	0
<b>weekday_is_monday</b>	0	0.0	0	0
<b>weekday_is_tuesday</b>	0	0.0	0	0
<b>weekday_is_wednesday</b>	0	0.0	0	0
<b>weekday_is_thursday</b>	0	0.0	0	0
<b>weekday_is_friday</b>	0	0.0	0	0
<b>weekday_is_saturday</b>	0	0.0	0	0
<b>weekday_is_sunday</b>	0	0.0	0	0
<b>is_weekend</b>	0	0.0	0	0
<b>LDA_00</b>	0	0.0	0	0
<b>LDA_01</b>	0	0.0	0	0
<b>LDA_02</b>	0	0.0	0	0
<b>LDA_03</b>	0	0.0	0	0
<b>LDA_04</b>	0	0.0	0	0
<b>global_subjectivity</b>	0	0.0	0	0
<b>global_sentiment_polarity</b>	0	0.0	0	0
<b>global_rate_positive_words</b>	0	0.0	0	0
<b>global_rate_negative_words</b>	0	0.0	0	0
<b>rate_positive_words</b>	0	0.0	0	0
<b>rate_negative_words</b>	0	0.0	0	0
<b>avg_positive_polarity</b>	0	0.0	0	0
<b>min_positive_polarity</b>	0	0.0	0	0
<b>max_positive_polarity</b>	0	0.0	0	0
<b>avg_negative_polarity</b>	0	0.0	0	0
<b>min_negative_polarity</b>	0	0.0	0	0
<b>max_negative_polarity</b>	0	0.0	0	0
<b>title_subjectivity</b>	0	0.0	0	0
<b>title_sentiment_polarity</b>	0	0.0	0	0
<b>abs_title_subjectivity</b>	0	0.0	0	0
<b>abs_title_sentiment_polarity</b>	0	0.0	0	0

	num_missing	missing_percentage	num_empty	empty_perce
--	-------------	--------------------	-----------	-------------

shares	0	0.0	0
--------	---	-----	---

There is no missing values. However, one of the columns seems misspelled. Let's take a look at the rest of the columns and see if there's anything else that is odd.

```
In [6]: df.columns
```

```
Out[6]: Index(['url', 'timedelta', 'n_tokens_title', 'n_tokens_content',
       'n_unique_tokens', 'n_non_stop_words', 'n_non_stop_unique_tokens',
       'num_hrefs', 'num_self_hrefs', 'num_imgs', 'num_videos',
       'average_token_length', 'num_keywords', 'data_channel_is_lifestyle',
       'data_channel_is_entertainment', 'data_channel_is_bus',
       'data_channel_is_socmed', 'data_channel_is_tech',
       'data_channel_is_world', 'kw_min_min', 'kw_max_min', 'kw_avg_min',
       'kw_min_max', 'kw_max_max', 'kw_avg_max', 'kw_min_avg',
       'kw_max_avg', 'kw_avg_avg', 'self_reference_min_shares',
       'self_reference_max_shares', 'self_reference_avg_shares',
       'weekday_is_monday', 'weekday_is_tuesday', 'weekday_is_wednesday',
       'weekday_is_thursday', 'weekday_is_friday', 'weekday_is_saturday',
       'weekday_is_sunday', 'is_weekend', 'LDA_00', 'LDA_01', 'LDA_02',
       'LDA_03', 'LDA_04', 'global_subjectivity',
       'global_sentiment_polarity', 'global_rate_positive_words',
       'global_rate_negative_words', 'rate_positive_words',
       'rate_negative_words', 'avg_positive_polarity',
       'min_positive_polarity', 'max_positive_polarity',
       'avg_negative_polarity', 'min_negative_polarity',
       'max_negative_polarity', 'title_subjectivity',
       'title_sentiment_polarity', 'abs_title_subjectivity',
       'abs_title_sentiment_polarity', 'shares'],
      dtype='object')
```

The column names are prepended with a space. Let's deal with that.

```
In [7]: for col in df.columns:
    df = df.rename(columns={col:(col.strip(' '))})

df = df.rename(columns={'self_reference_avg_shares': 'self_reference_avg_shares'})
```

Now, let's drop some duplicates if any.

```
In [8]: df.shape
df = df.drop_duplicates(keep = False)
df.shape
```

```
Out[8]: (39644, 61)
```

```
Out[8]: (39644, 61)
```

There are no duplicates after all. This dataset is already [retty clean!]

Let's now create our target variable using the threshold of 1400 social media shares.

```
In [9]: # creates a new column for the new target variable and non-descriptive column  
df['target'] = np.where(df['shares'] > 1400, int(1), int(0))  
df = df.drop(columns=['url', 'timedelta'])
```

Now, we're ready to export our cleaned and prepared dataset.

```
In [10]: df.to_csv('../data/out/online_news_popularity_clean.csv', index=False)
```

Let's do a little bit of exploration.

## EXPLORATION

Let's take a peek!

```
In [11]: df.head()
```

```
Out[11]:   n_tokens_title  n_tokens_content  n_unique_tokens  n_non_stop_words  n_non_stop_unique  
0           12.0          219.0        0.663594            1.0                 C  
1            9.0          255.0        0.604743            1.0                 C  
2            9.0          211.0        0.575130            1.0                 C  
3            9.0          531.0        0.503788            1.0                 C  
4           13.0          1072.0       0.415646            1.0                 C
```

◀ | ▶

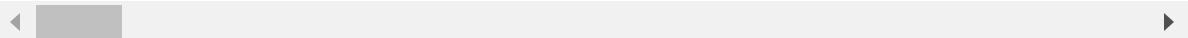
```
In [12]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39644 entries, 0 to 39643
Data columns (total 60 columns):
 #   Column           Non-Null Count Dtype  
 ---  -- 
 0   n_tokens_title    39644 non-null  float64 
 1   n_tokens_content  39644 non-null  float64 
 2   n_unique_tokens   39644 non-null  float64 
 3   n_non_stop_words  39644 non-null  float64 
 4   n_non_stop_unique_tokens 39644 non-null  float64 
 5   num_hrefs         39644 non-null  float64 
 6   num_self_hrefs   39644 non-null  float64 
 7   num_imgs          39644 non-null  float64 
 8   num_videos        39644 non-null  float64 
 9   average_token_length 39644 non-null  float64 
 10  num_keywords      39644 non-null  float64 
 11  data_channel_is_lifestyle 39644 non-null  float64 
 12  data_channel_is_entertainment 39644 non-null  float64 
 13  data_channel_is_bus          39644 non-null  float64 
 14  data_channel_is_socmed       39644 non-null  float64 
 15  data_channel_is_tech        39644 non-null  float64 
 16  data_channel_is_world       39644 non-null  float64 
 17  kw_min_min            39644 non-null  float64 
 18  kw_max_min            39644 non-null  float64 
 19  kw_avg_min             39644 non-null  float64 
 20  kw_min_max             39644 non-null  float64 
 21  kw_max_max             39644 non-null  float64 
 22  kw_avg_max             39644 non-null  float64 
 23  kw_min_avg             39644 non-null  float64 
 24  kw_max_avg             39644 non-null  float64 
 25  kw_avg_avg             39644 non-null  float64 
 26  self_reference_min_shares 39644 non-null  float64 
 27  self_reference_max_shares 39644 non-null  float64 
 28  self_reference_avg_shares 39644 non-null  float64 
 29  weekday_is_monday       39644 non-null  float64 
 30  weekday_is_tuesday      39644 non-null  float64 
 31  weekday_is_wednesday    39644 non-null  float64 
 32  weekday_is_thursday     39644 non-null  float64 
 33  weekday_is_friday       39644 non-null  float64 
 34  weekday_is_saturday     39644 non-null  float64 
 35  weekday_is_sunday       39644 non-null  float64 
 36  is_weekend              39644 non-null  float64 
 37  LDA_00                  39644 non-null  float64 
 38  LDA_01                  39644 non-null  float64 
 39  LDA_02                  39644 non-null  float64 
 40  LDA_03                  39644 non-null  float64 
 41  LDA_04                  39644 non-null  float64 
 42  global_subjectivity     39644 non-null  float64 
 43  global_sentiment_polarity 39644 non-null  float64 
 44  global_rate_positive_words 39644 non-null  float64 
 45  global_rate_negative_words 39644 non-null  float64 
 46  rate_positive_words     39644 non-null  float64 
 47  rate_negative_words     39644 non-null  float64 
 48  avg_positive_polarity   39644 non-null  float64 
 49  min_positive_polarity   39644 non-null  float64 
 50  max_positive_polarity   39644 non-null  float64
```

```
51 avg_negative_polarity           39644 non-null float64
52 min_negative_polarity          39644 non-null float64
53 max_negative_polarity          39644 non-null float64
54 title_subjectivity              39644 non-null float64
55 title_sentiment_polarity        39644 non-null float64
56 abs_title_subjectivity          39644 non-null float64
57 abs_title_sentiment_polarity    39644 non-null float64
58 shares                          39644 non-null int64
59 target                           39644 non-null int32
dtypes: float64(58), int32(1), int64(1)
memory usage: 18.0 MB
```

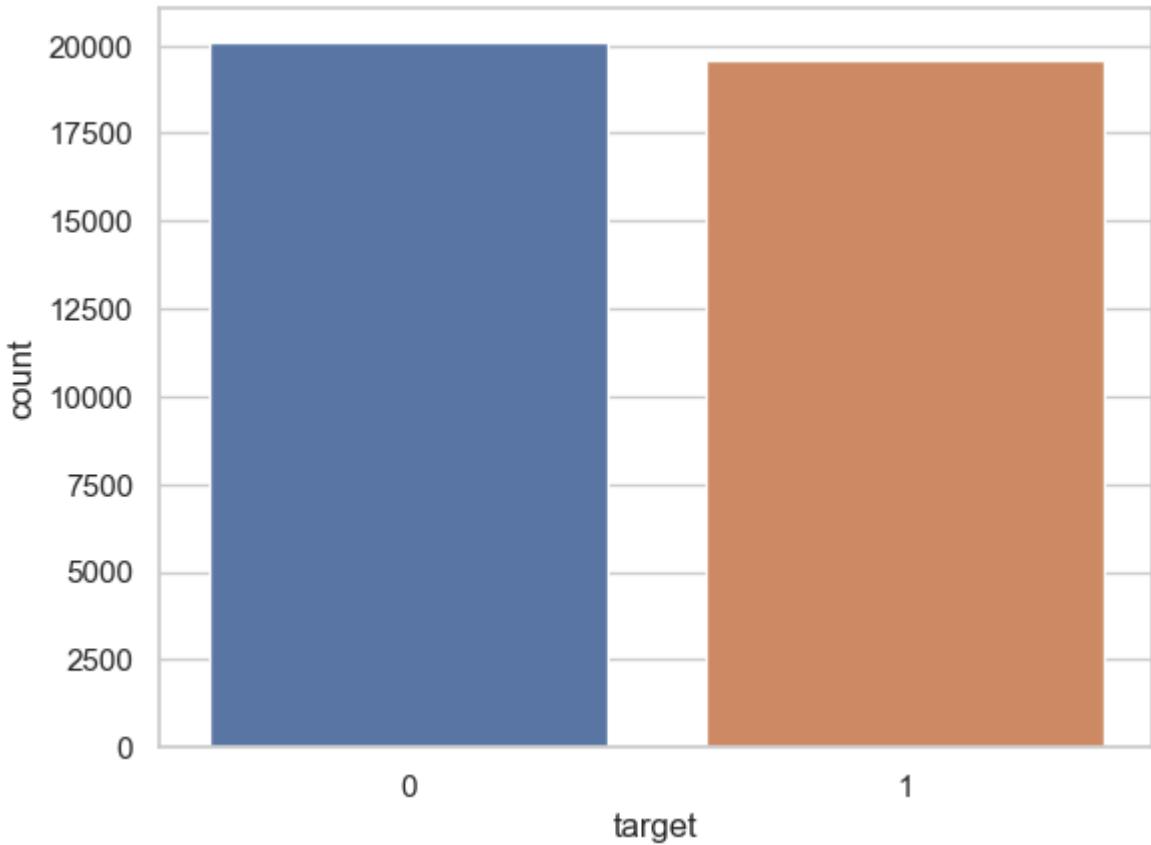
```
In [13]: df.describe()
```

	n_tokens_title	n_tokens_content	n_unique_tokens	n_non_stop_words	n_non_stop_un
<b>count</b>	39644.000000	39644.000000	39644.000000	39644.000000	3
<b>mean</b>	10.398749	546.514731	0.548216	0.996469	
<b>std</b>	2.114037	471.107508	3.520708	5.231231	
<b>min</b>	2.000000	0.000000	0.000000	0.000000	
<b>25%</b>	9.000000	246.000000	0.470870	1.000000	
<b>50%</b>	10.000000	409.000000	0.539226	1.000000	
<b>75%</b>	12.000000	716.000000	0.608696	1.000000	
<b>max</b>	23.000000	8474.000000	701.000000	1042.000000	



```
In [14]: sns.countplot(x='target', data=df)
```

```
Out[14]: <Axes: xlabel='target', ylabel='count'>
```



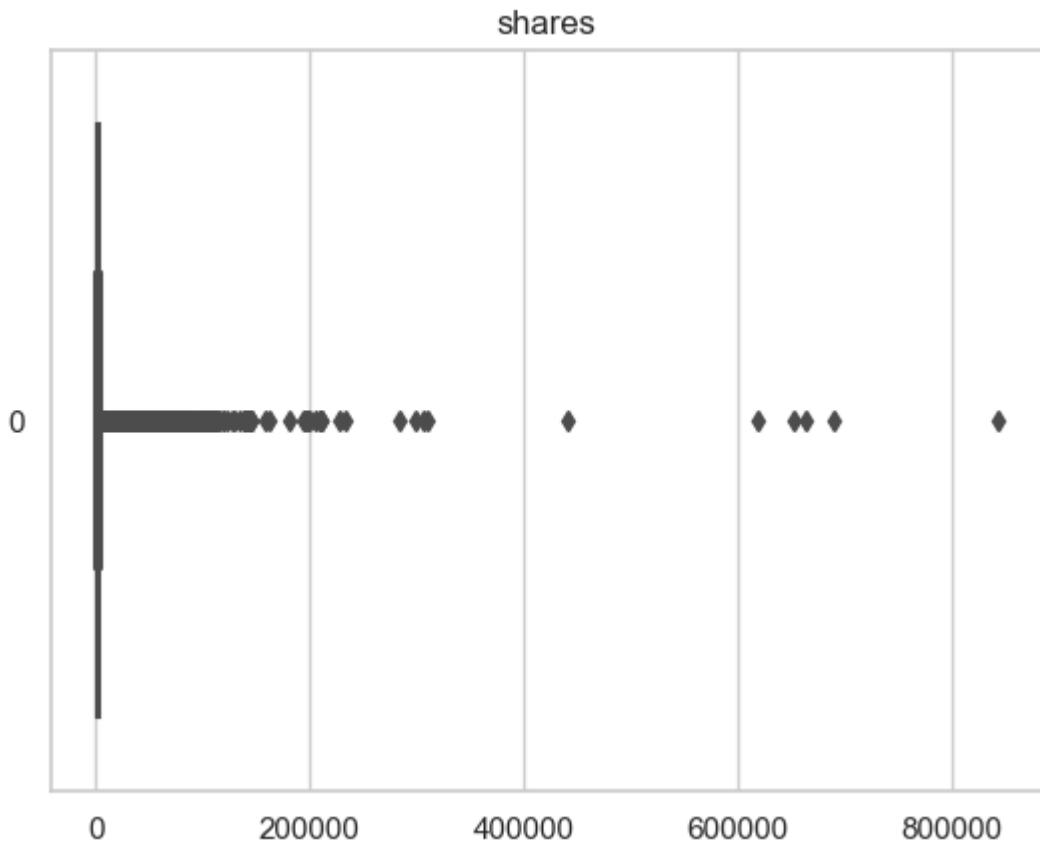
We have do not have an imbalanced dataset.

```
In [15]: token_cols = ['n_tokens_title', 'n_tokens_content', 'n_unique_tokens', 'n_non_stop_links_cols = ['num_hrefs', 'num_self_hrefs']
media_cols = ['num_imgs', 'num_videos']
channel_cols = ['data_channel_is_lifestyle', 'data_channel_is_entertainment', 'data_kw_cols = ['kw_min_min', 'kw_max_min', 'kw_avg_min', 'kw_min_max', 'kw_max_max', 'k
self_ref_cols = ['self_reference_min_shares', 'self_reference_max_shares', 'self_re
week_cols = ['weekday_is_monday', 'weekday_is_tuesday', 'weekday_is_wednesday', 'we
topic_cols = ['LDA_00', 'LDA_01', 'LDA_02', 'LDA_03', 'LDA_04']
global_cols = ['global_subjectivity', 'global_sentiment_polarity', 'global_rate_pos
local_cols = ['rate_positive_words', 'rate_negative_words', 'avg_positive_polarity'
title_cols = ['title_subjectivity', 'title_sentiment_polarity', 'abs_title_subjecti

all_columns = ['token_cols', 'links_cols', 'media_cols', 'channel_cols', 'kw_cols',
               'weekday_cols', 'weekend_cols', 'topic_cols', 'global_cols', 'local_
```

```
In [16]: def viz_box(df, col):
    sns.boxplot(df[col], orient="h")
    plt.title(str(col))
    plt.show()
```

```
In [17]: viz_box(df, 'shares')
```



We see some outliers. Let's remove them.

```
In [18]: percentile25 = df['shares'].quantile(0.25)
percentile75 = df['shares'].quantile(0.75)

print("75th quartile: ", percentile75)
print("25th quartile: ", percentile25)

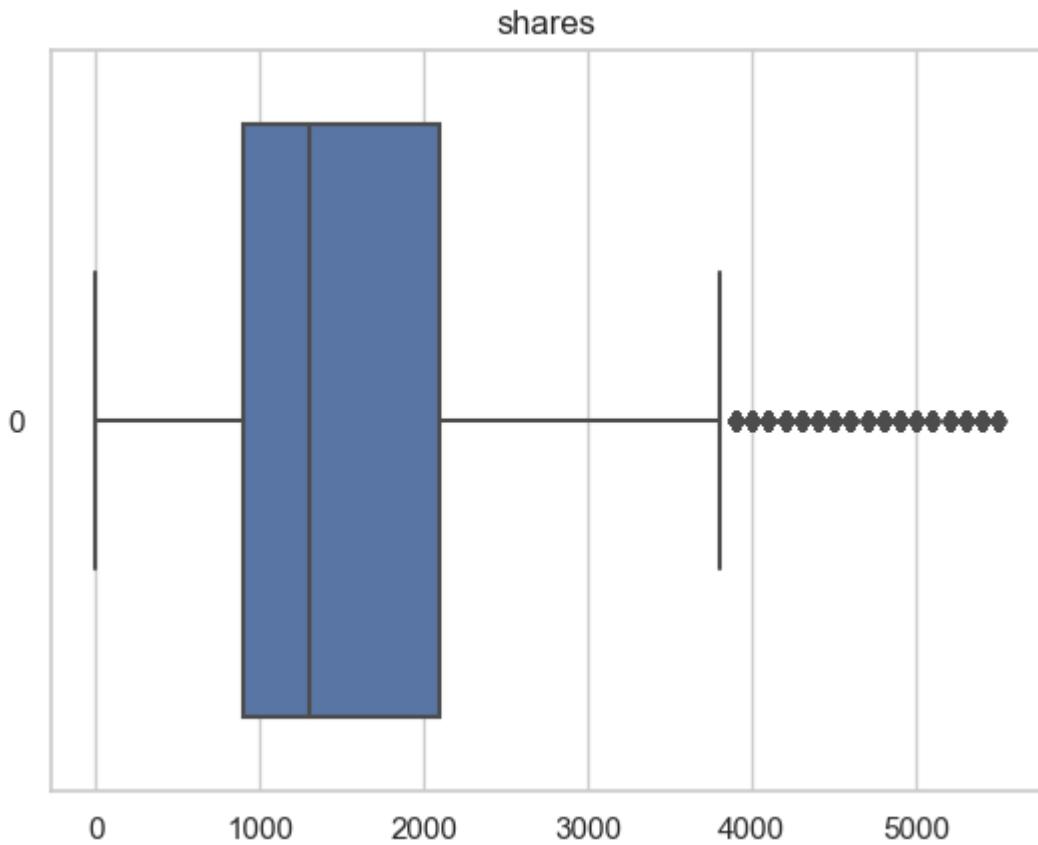
iqr = percentile75 - percentile25

upper_bound = percentile75 + 1.5 * iqr
lower_bound = percentile25 - 1.5 * iqr

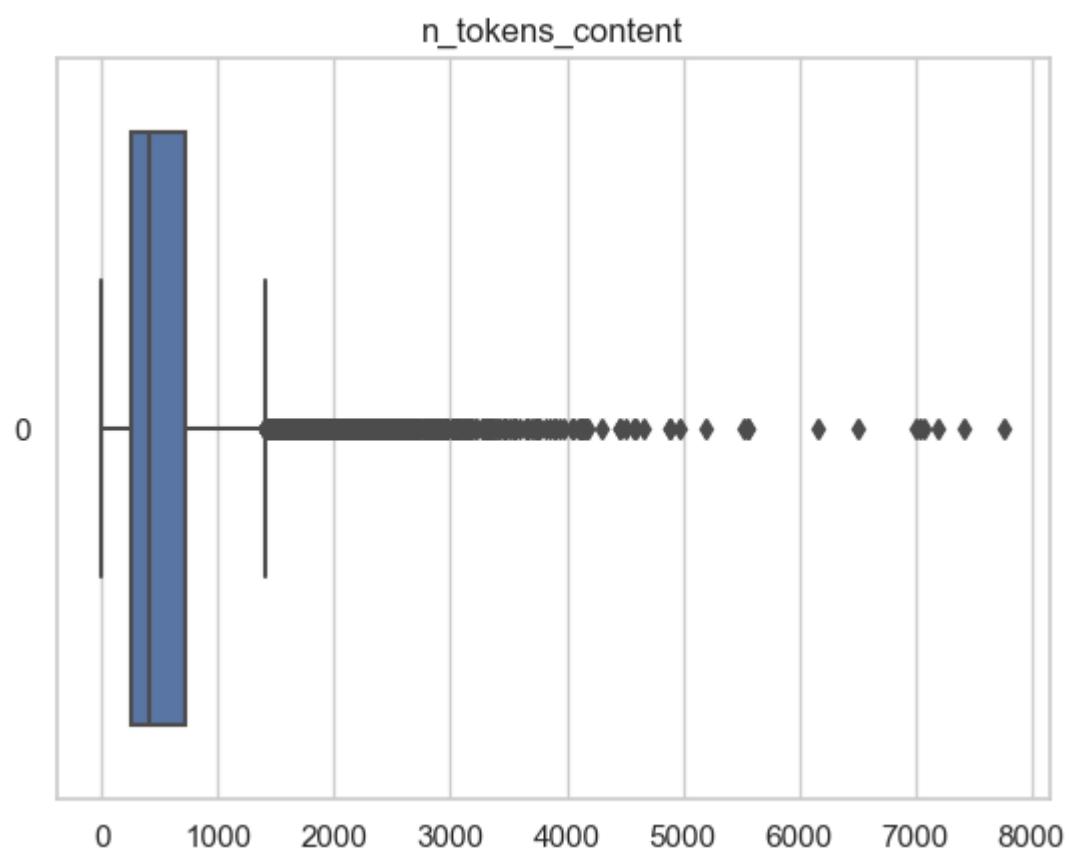
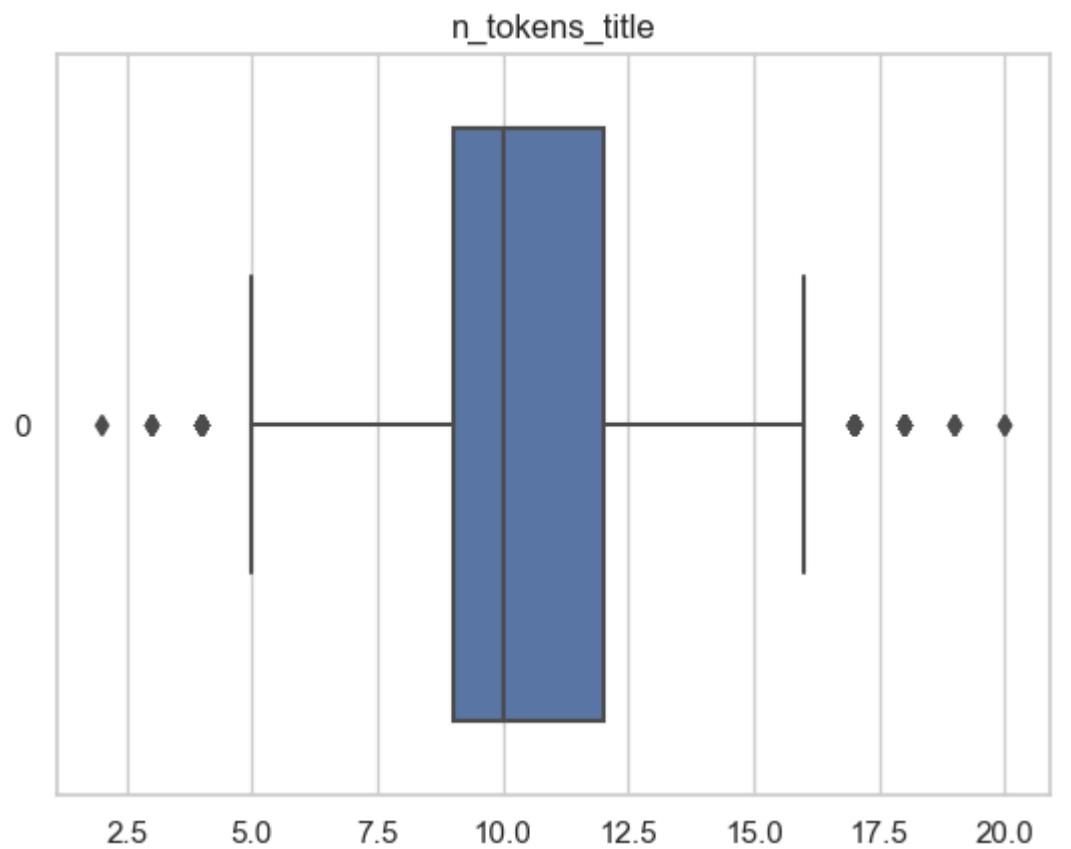
df = df[df['shares'] < upper_bound]
df = df[df['shares'] > lower_bound]
print(len(df))
```

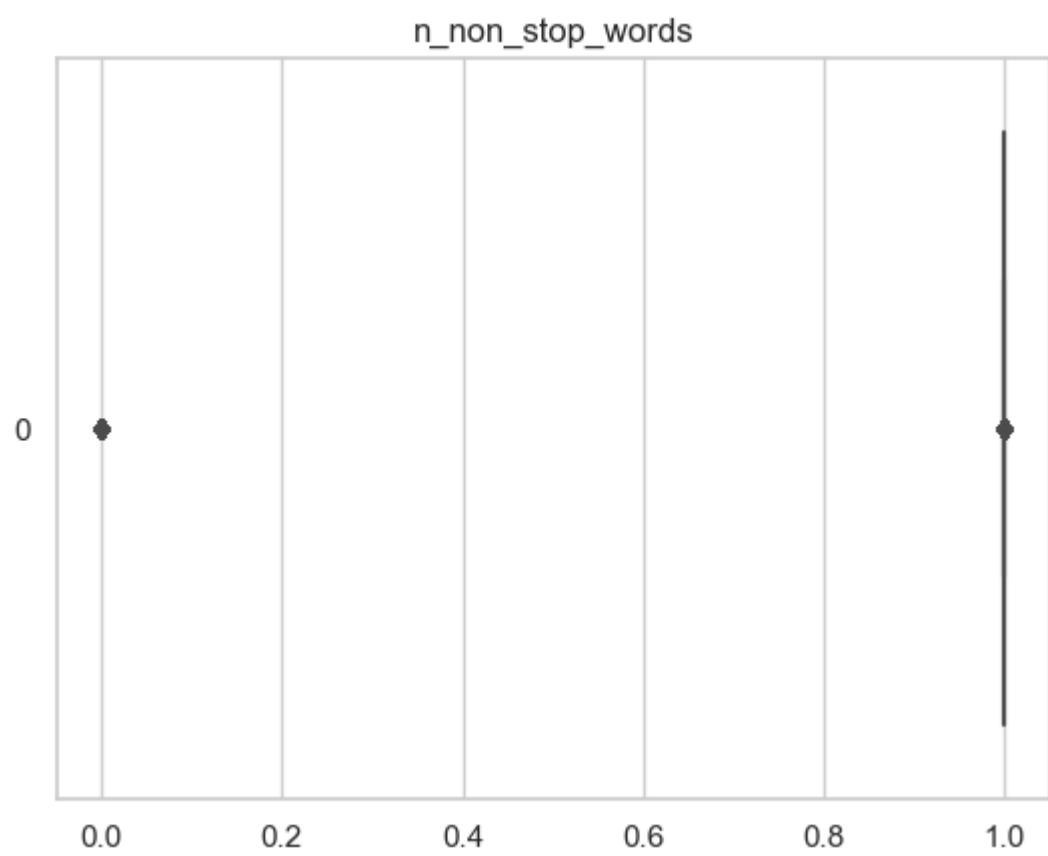
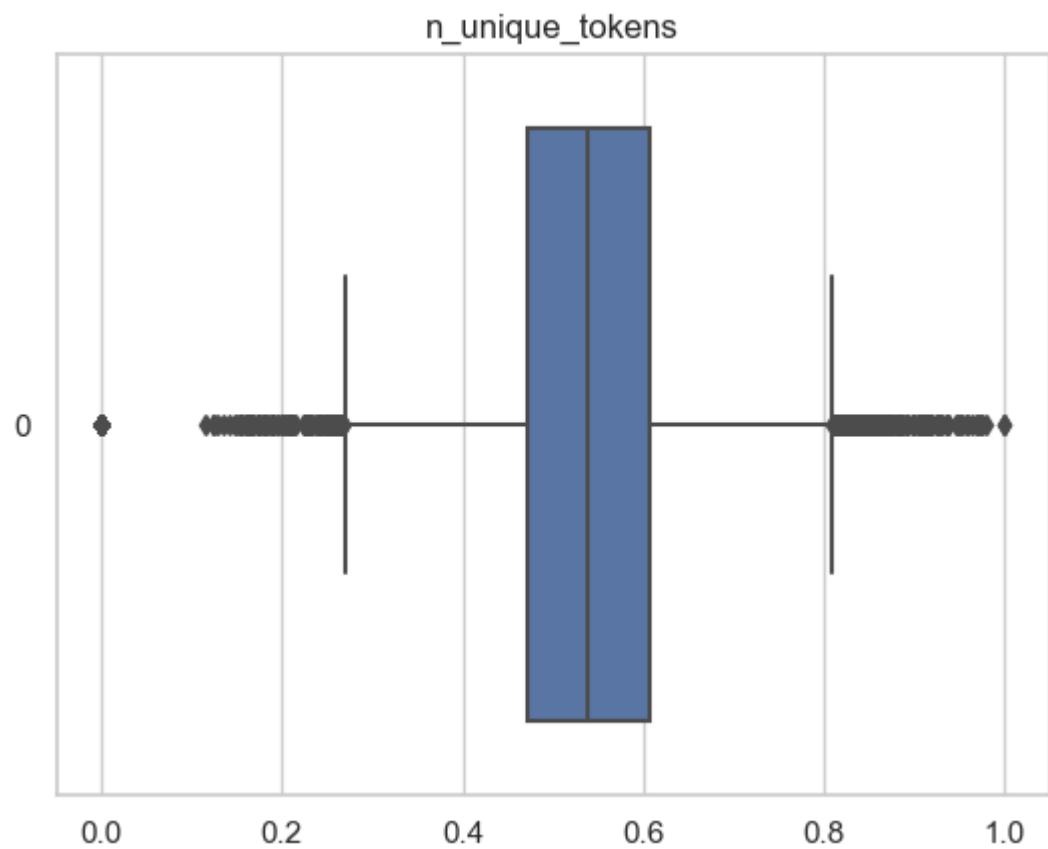
```
75th quartile:  2800.0
25th quartile:  946.0
35103
```

```
In [19]: viz_box(df, 'shares')
```

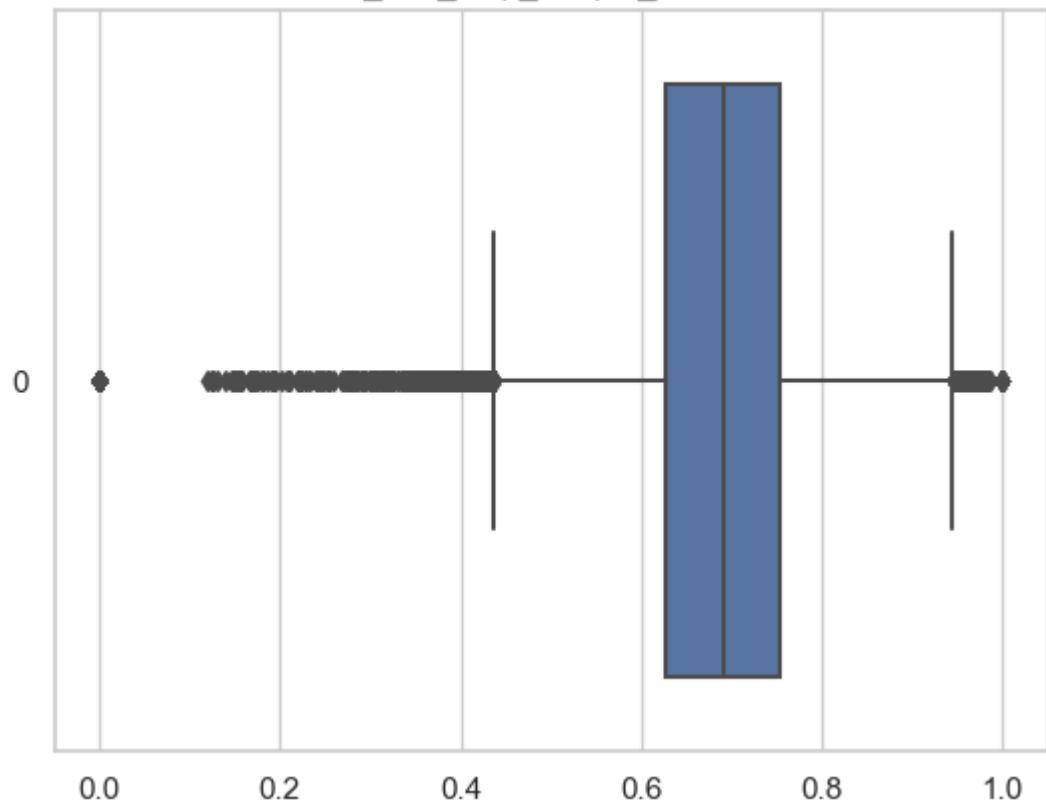


```
In [20]: continuous_cols = token_cols + links_cols + kw_cols + self_ref_cols + global_cols +  
In [21]: for col in continuous_cols:  
         viz_box(df, col)
```

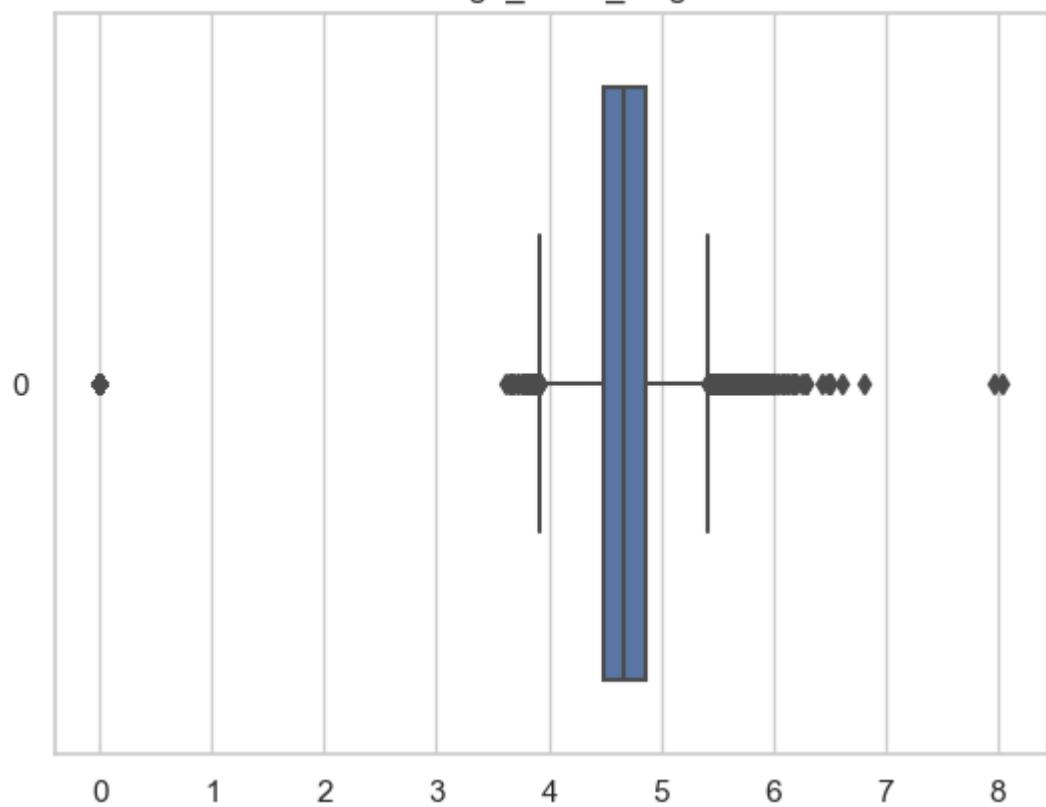




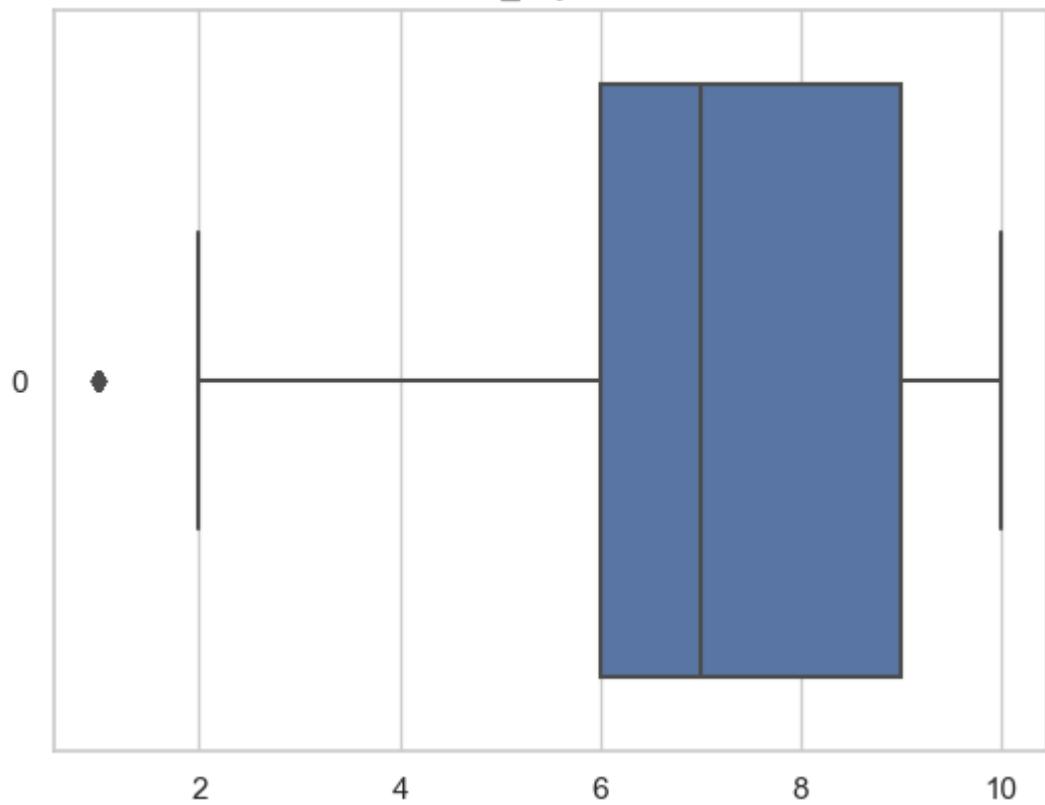
n\_non\_stop\_unique\_tokens



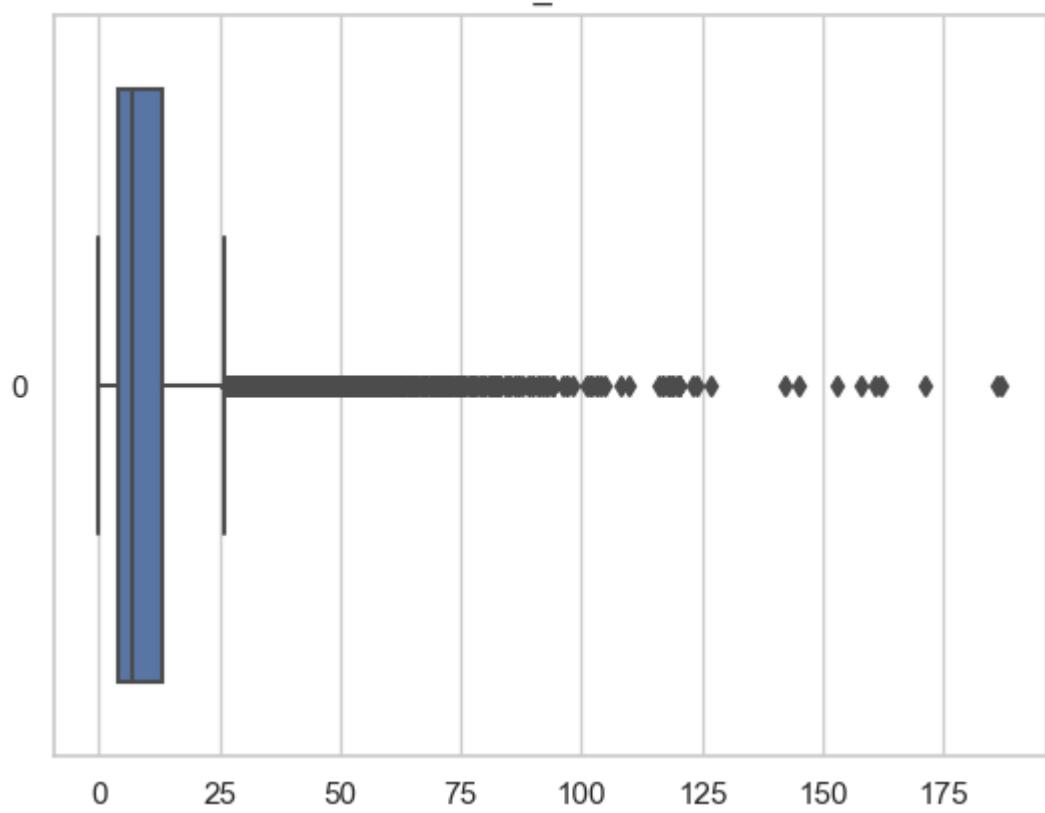
average\_token\_length

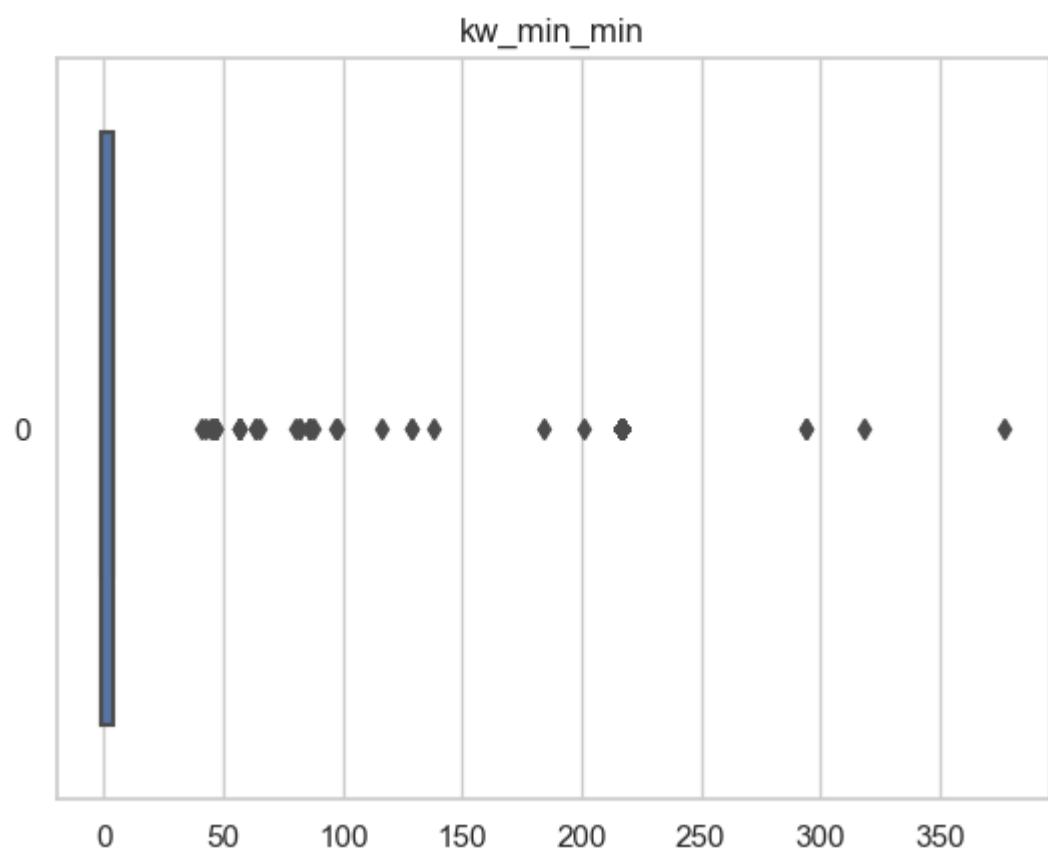
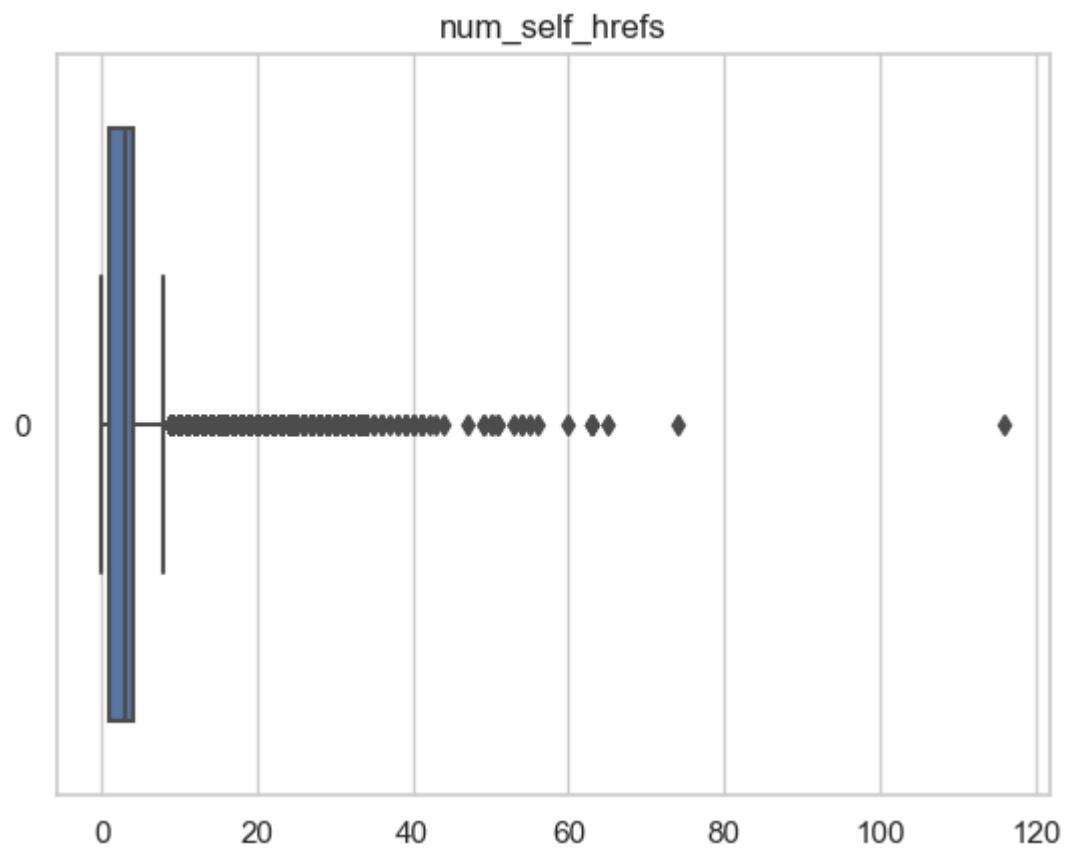


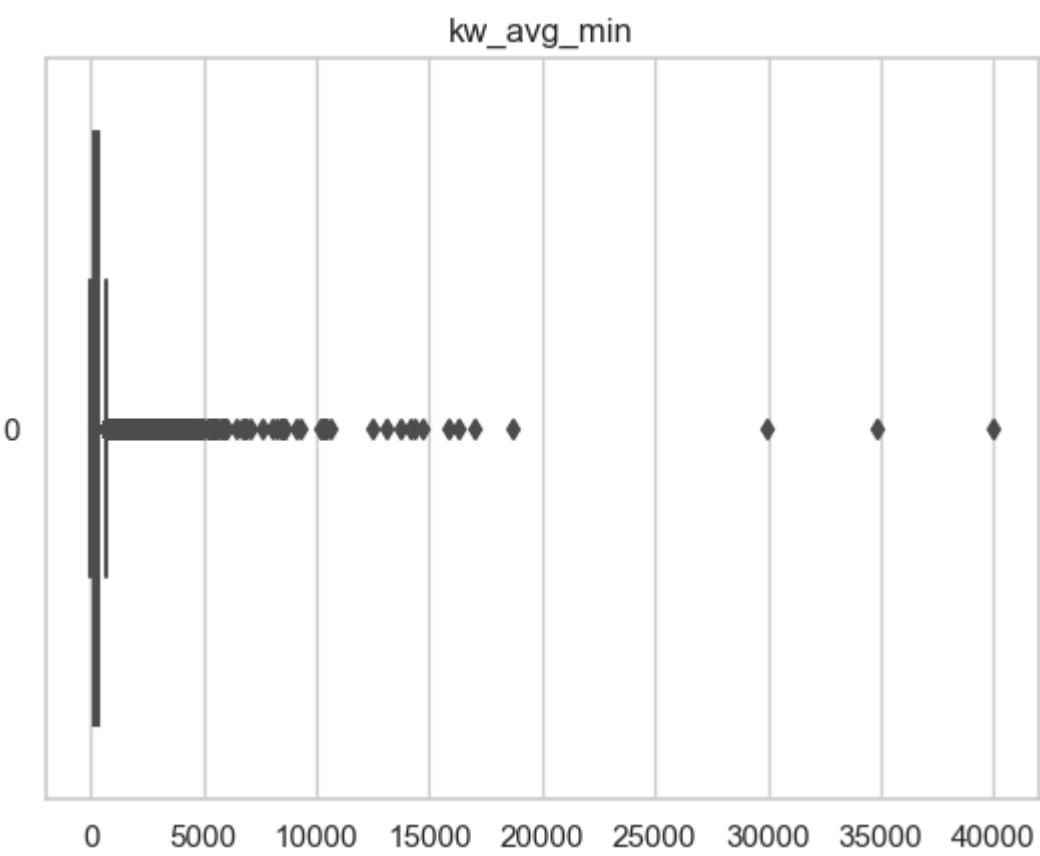
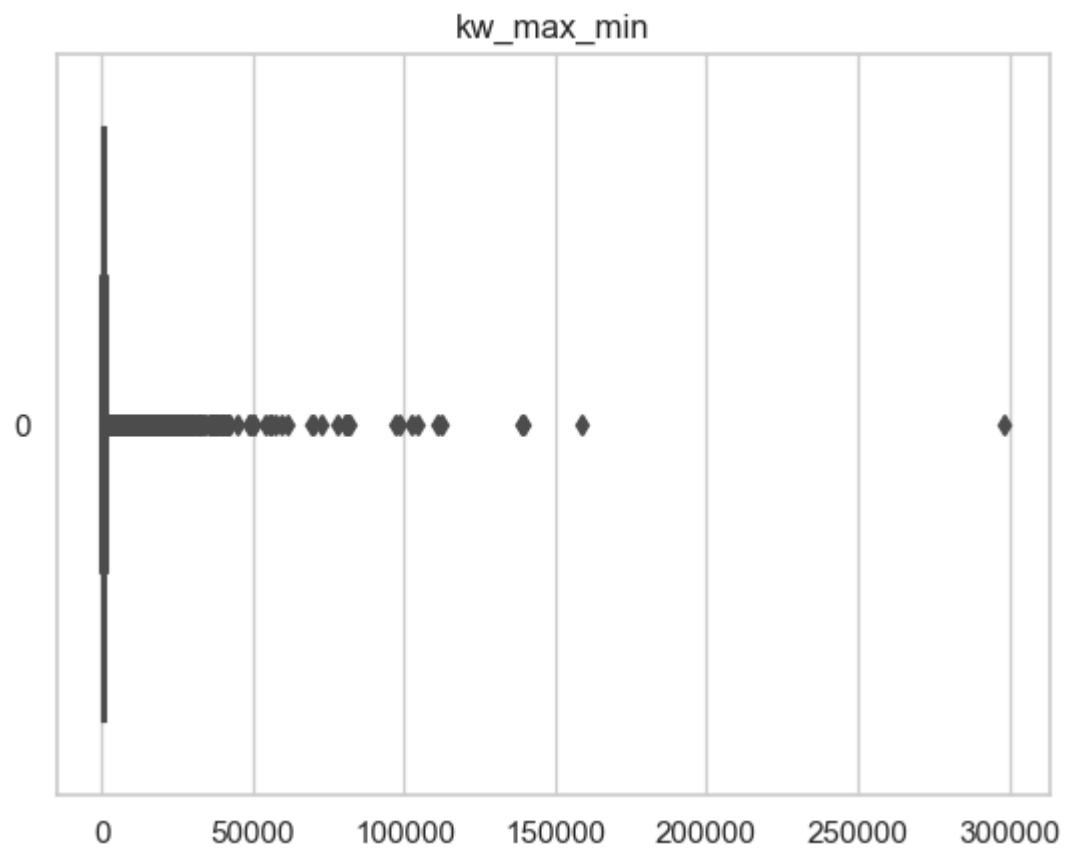
num\_keywords



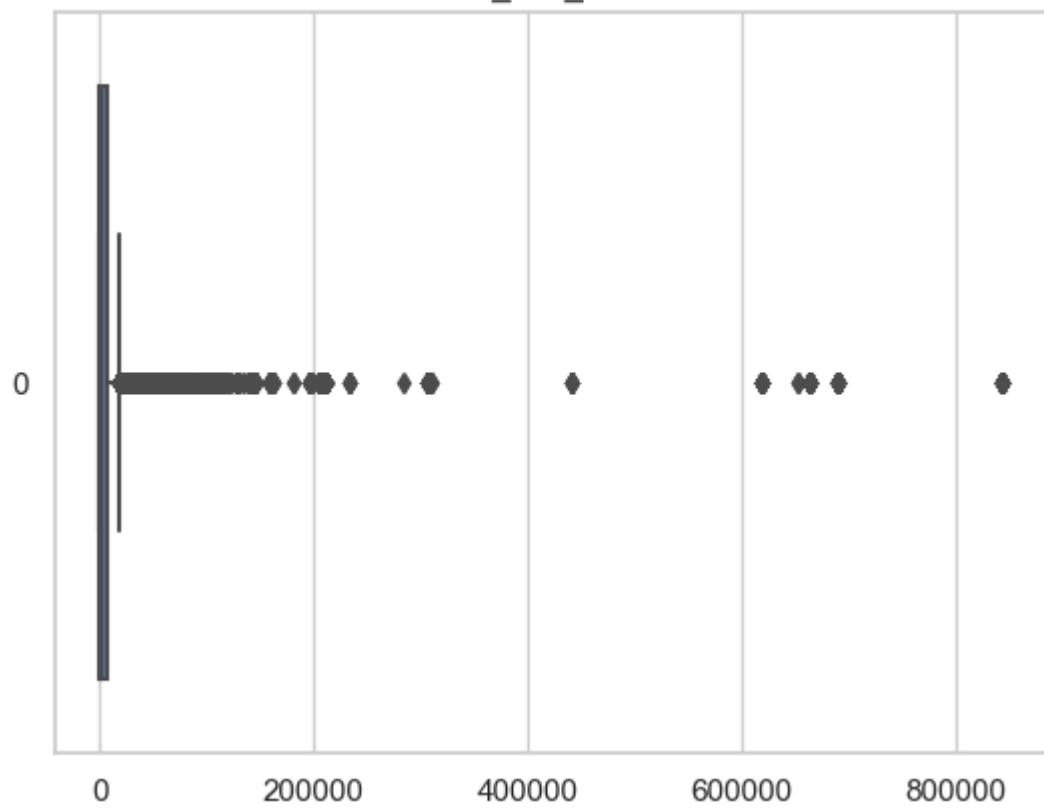
num\_hrefs



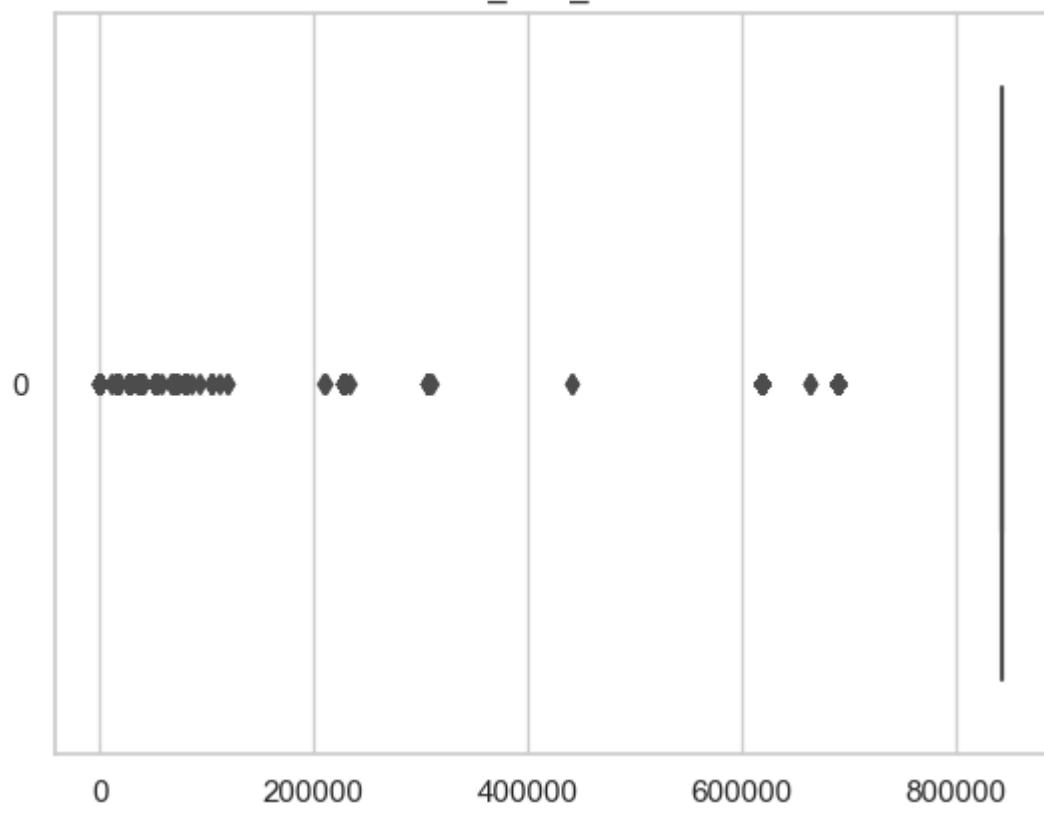




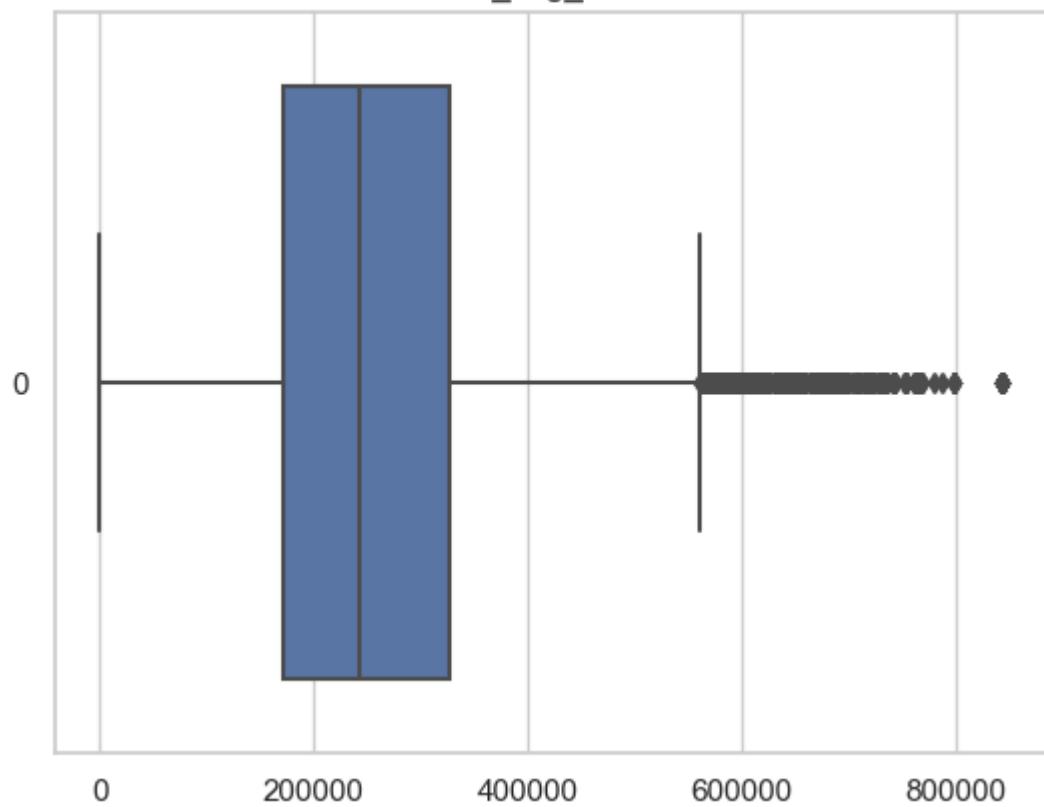
kw\_min\_max



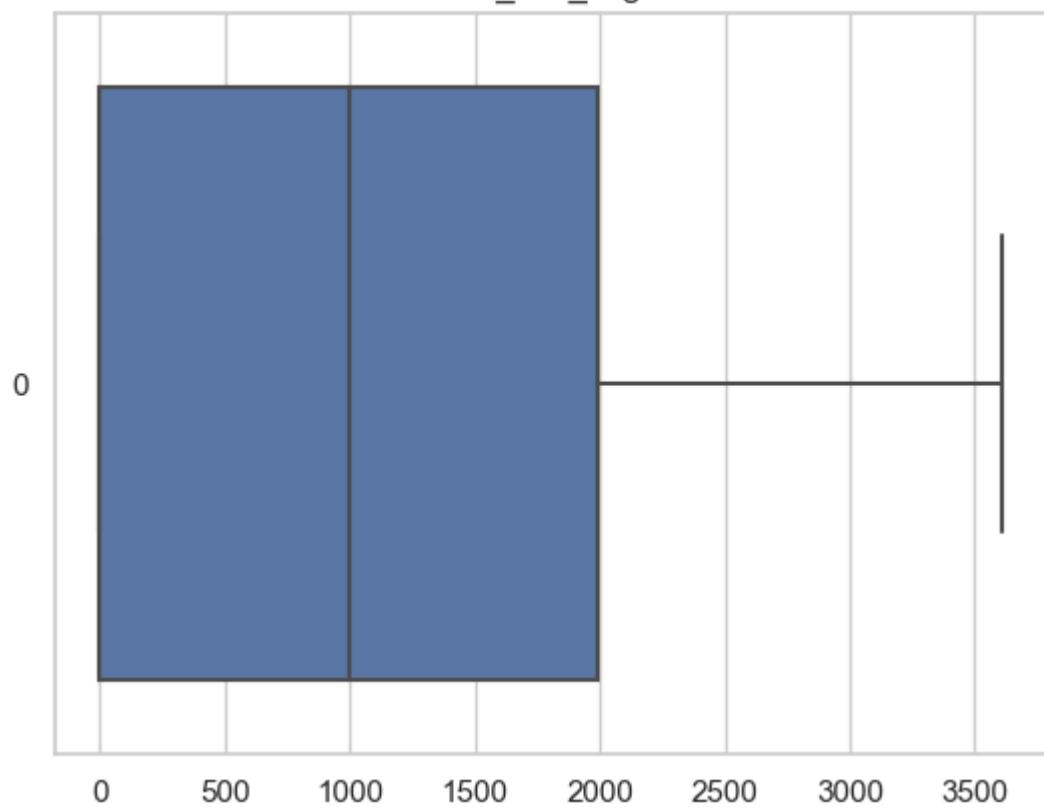
kw\_max\_max

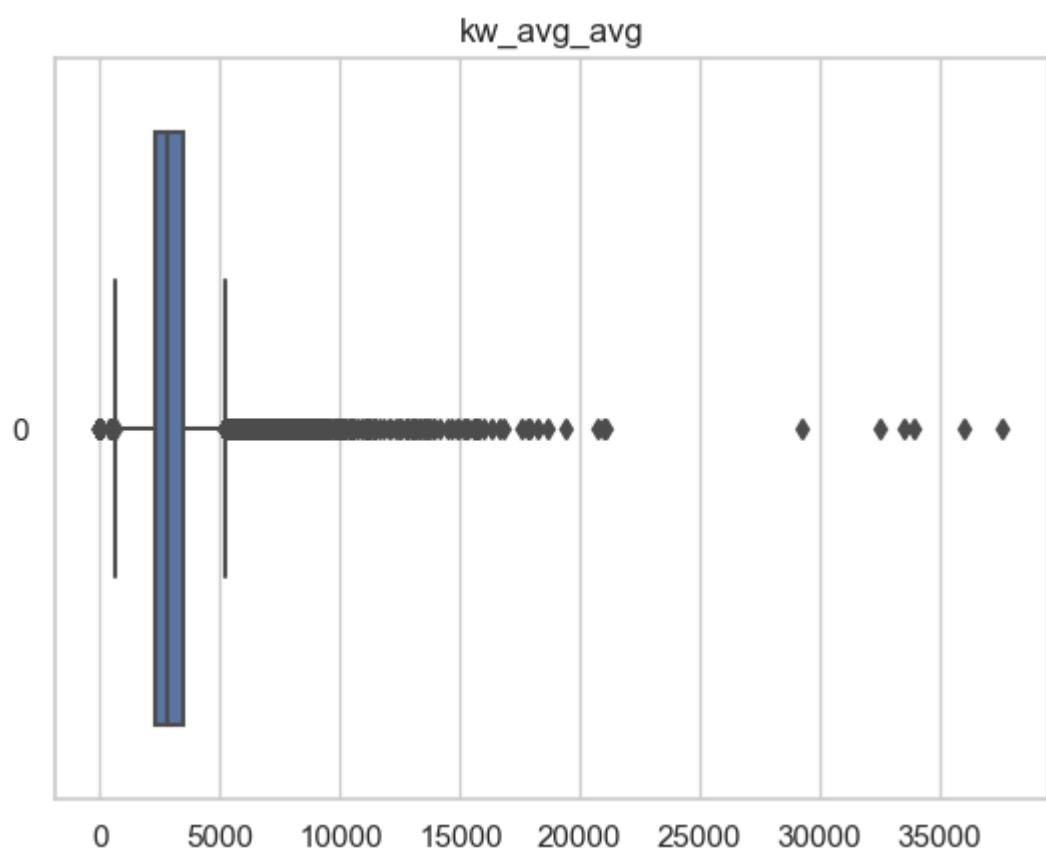
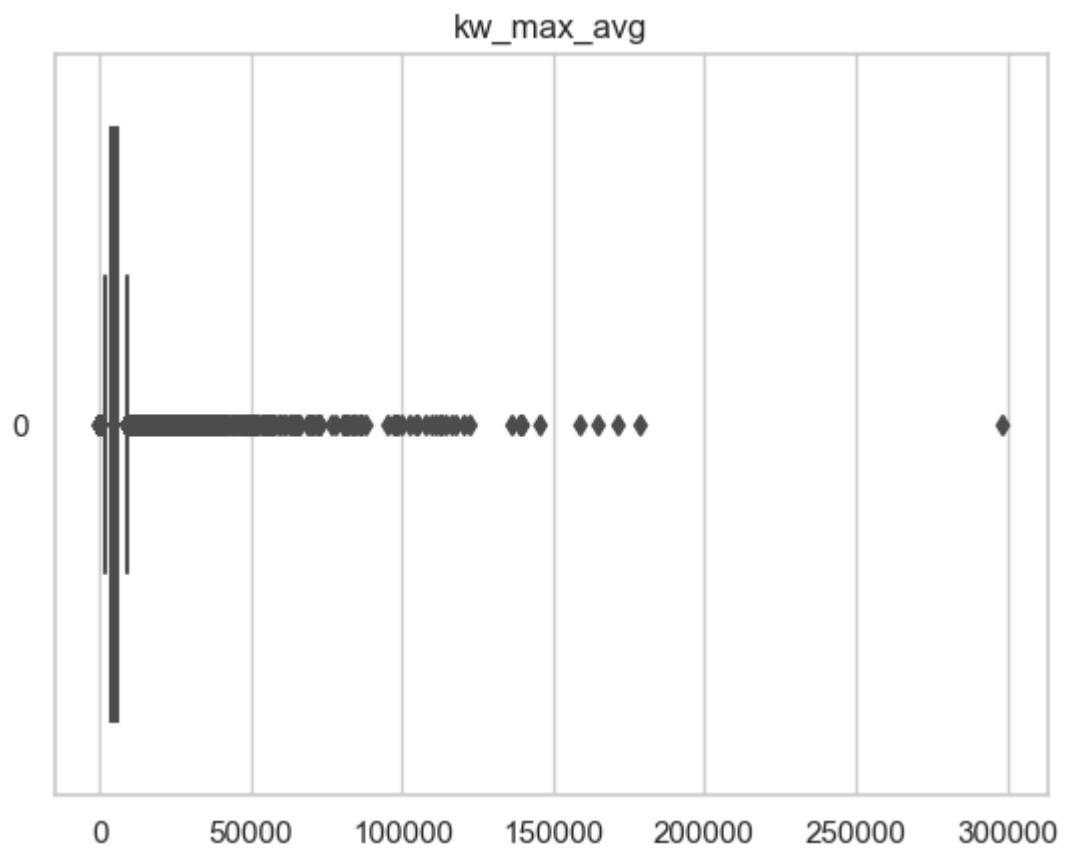


kw\_avg\_max

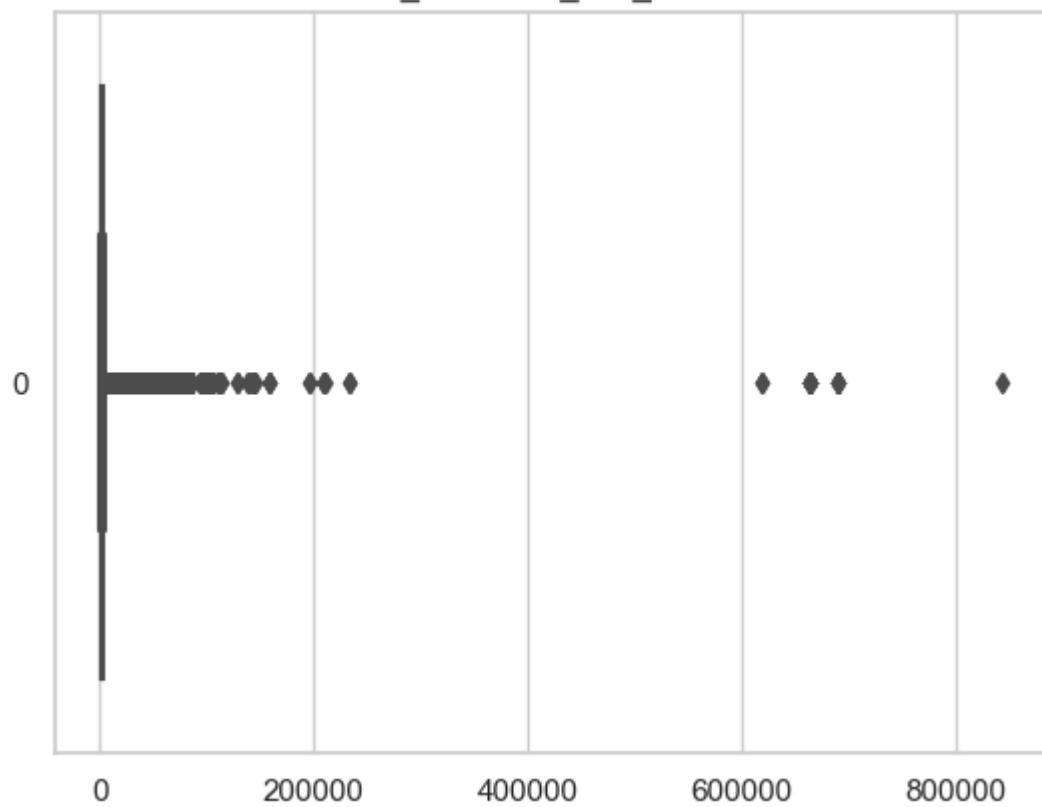


kw\_min\_avg

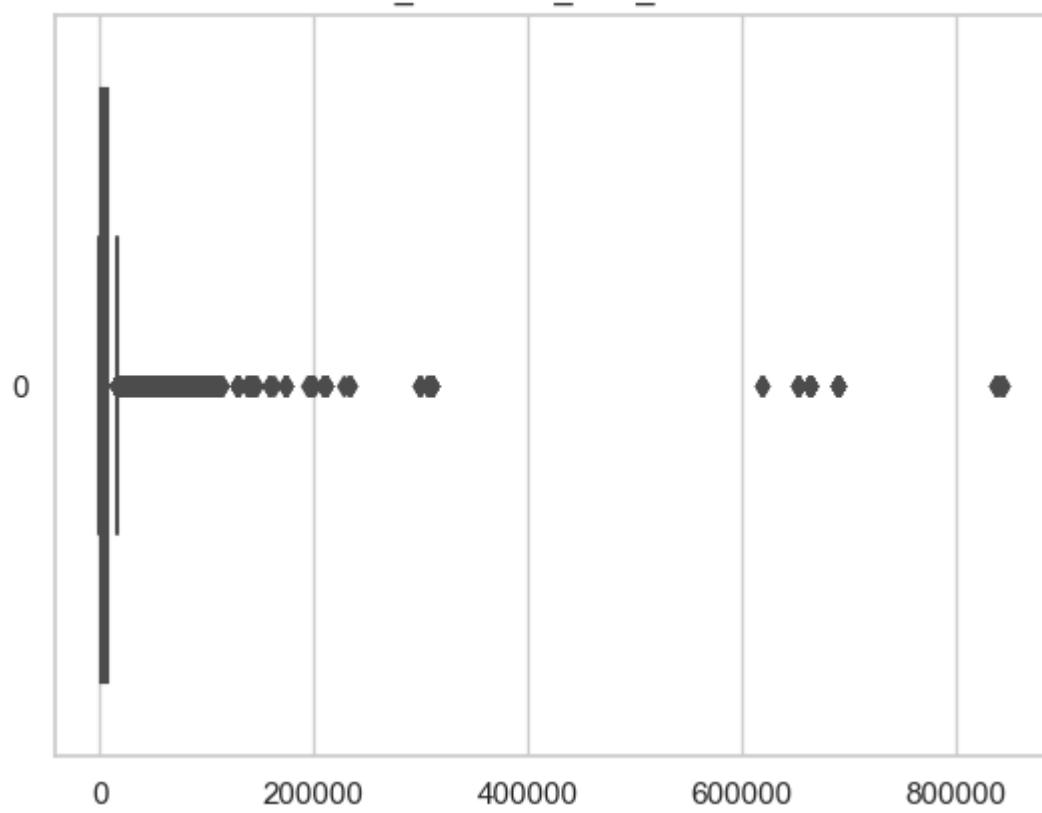




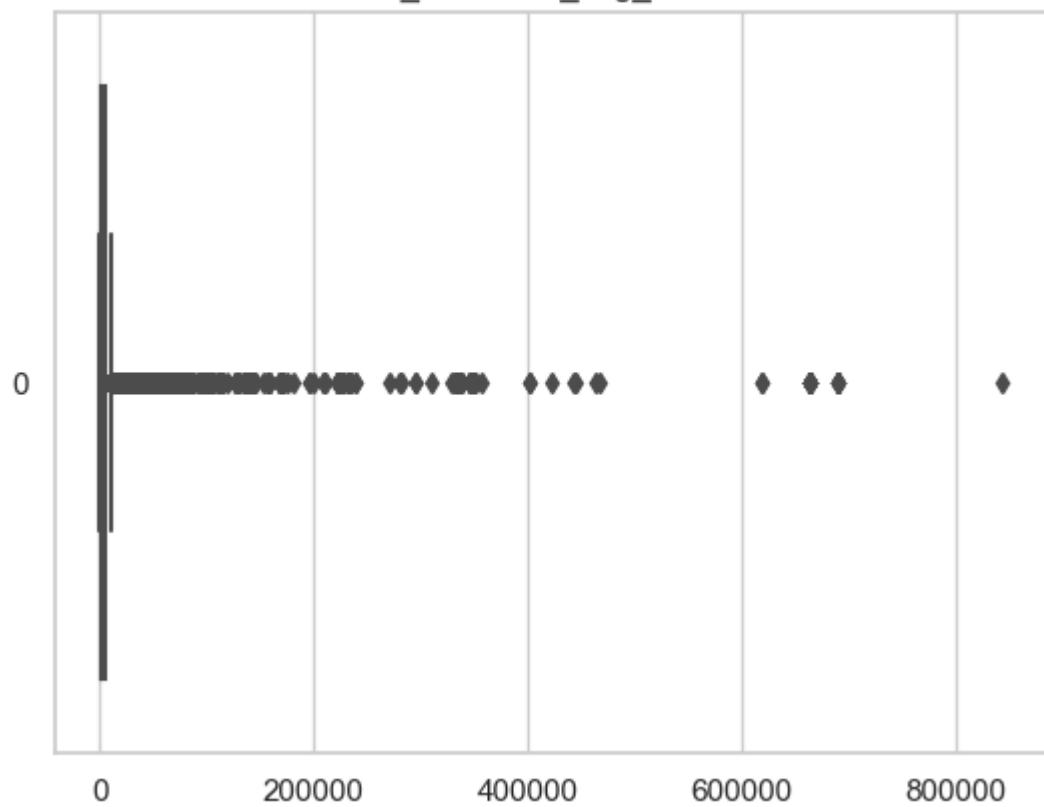
self\_reference\_min\_shares



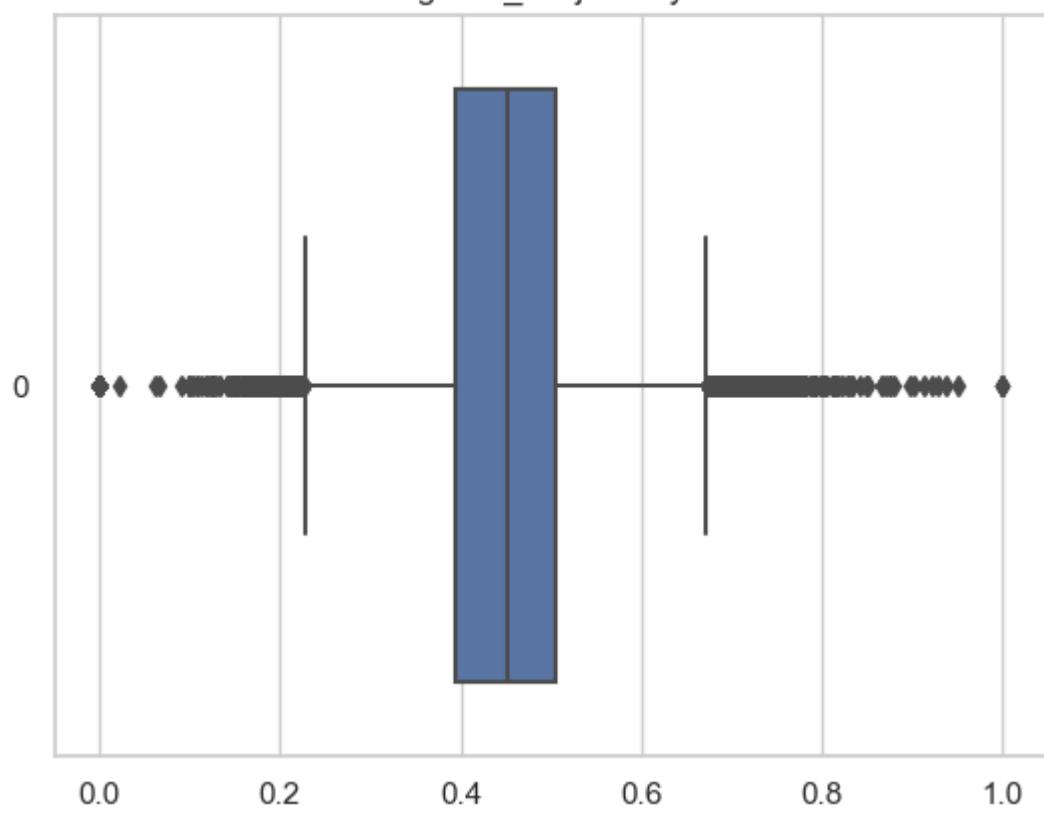
self\_reference\_max\_shares



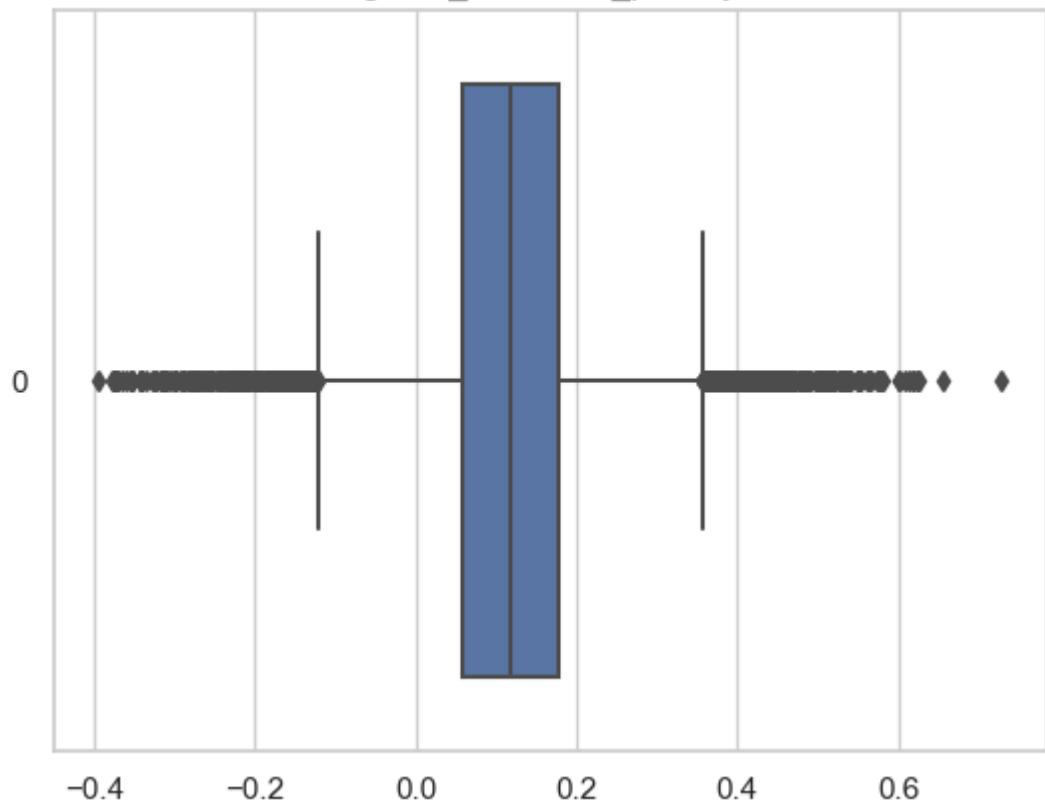
self\_reference\_avg\_shares



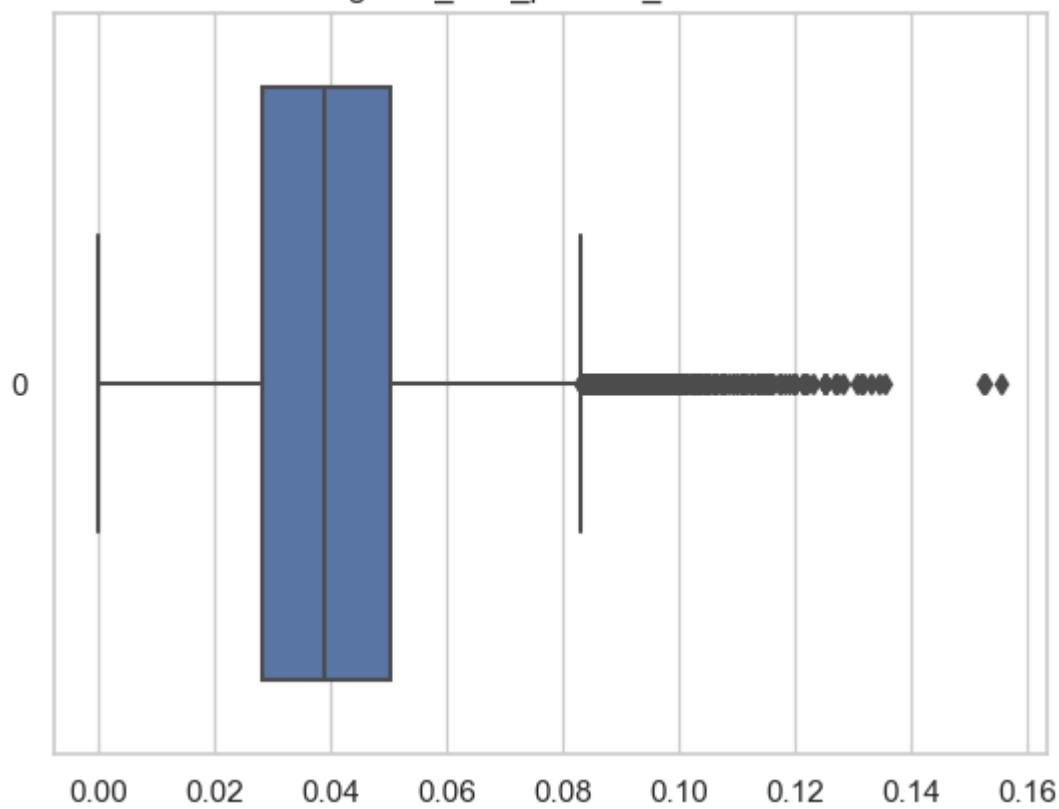
global\_subjectivity



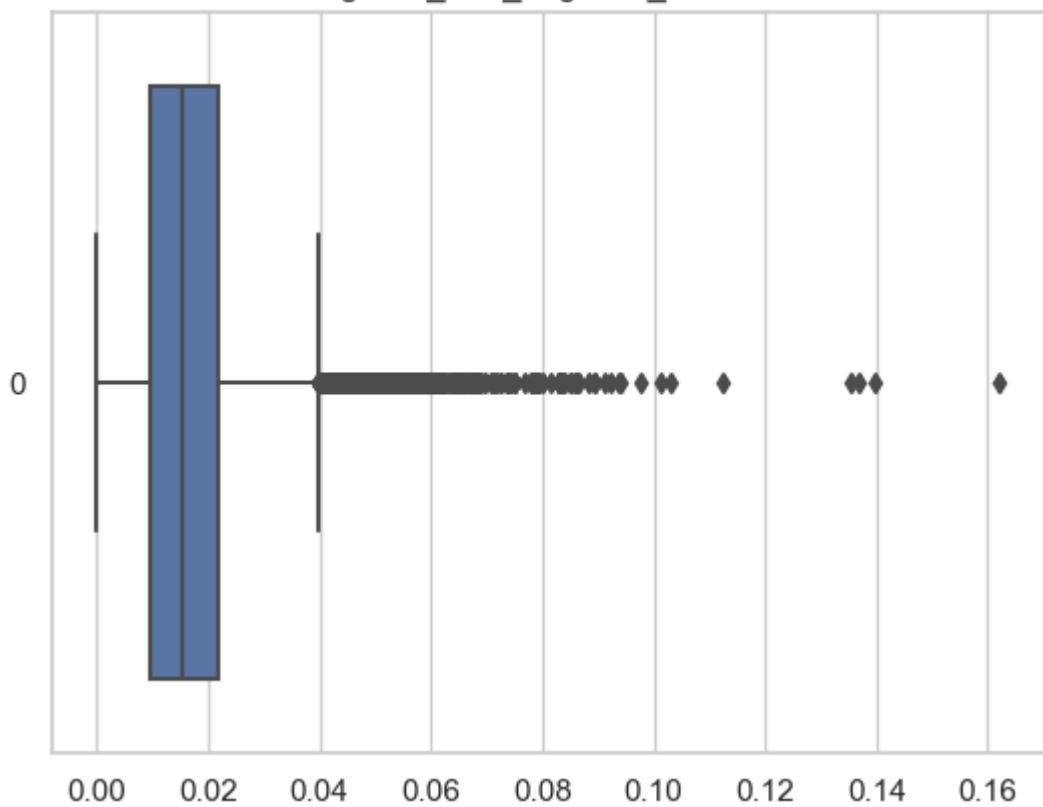
global\_sentiment\_polarity



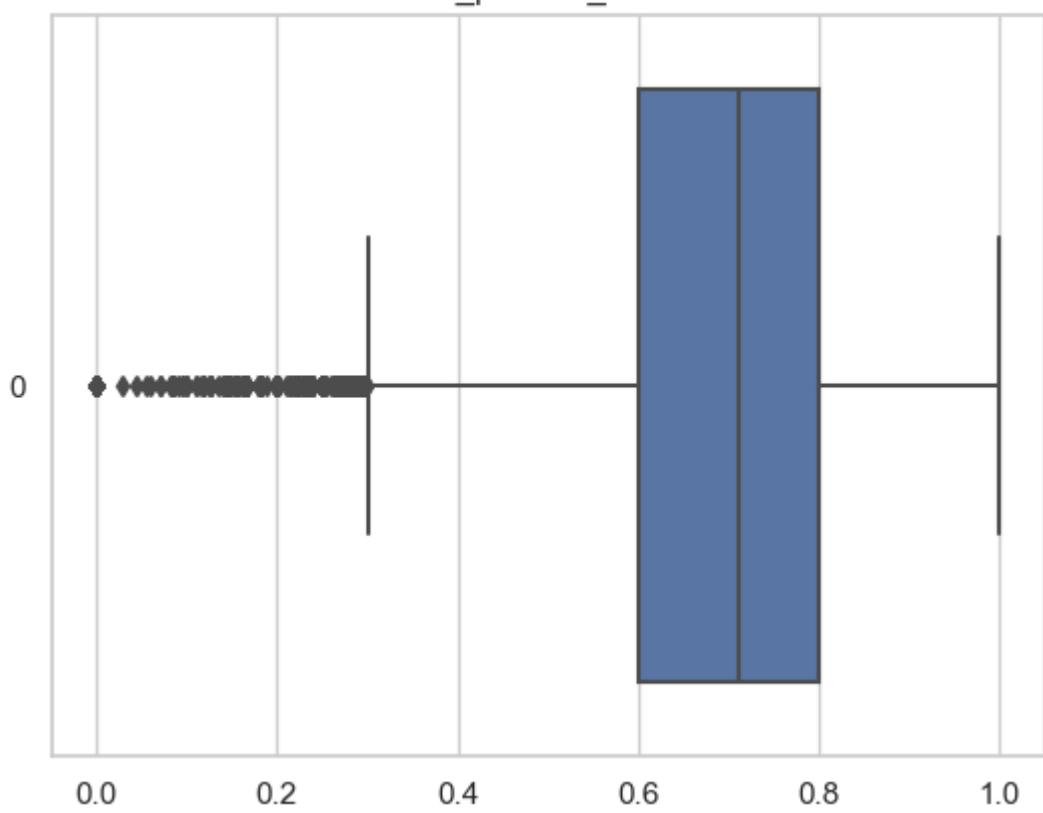
global\_rate\_positive\_words



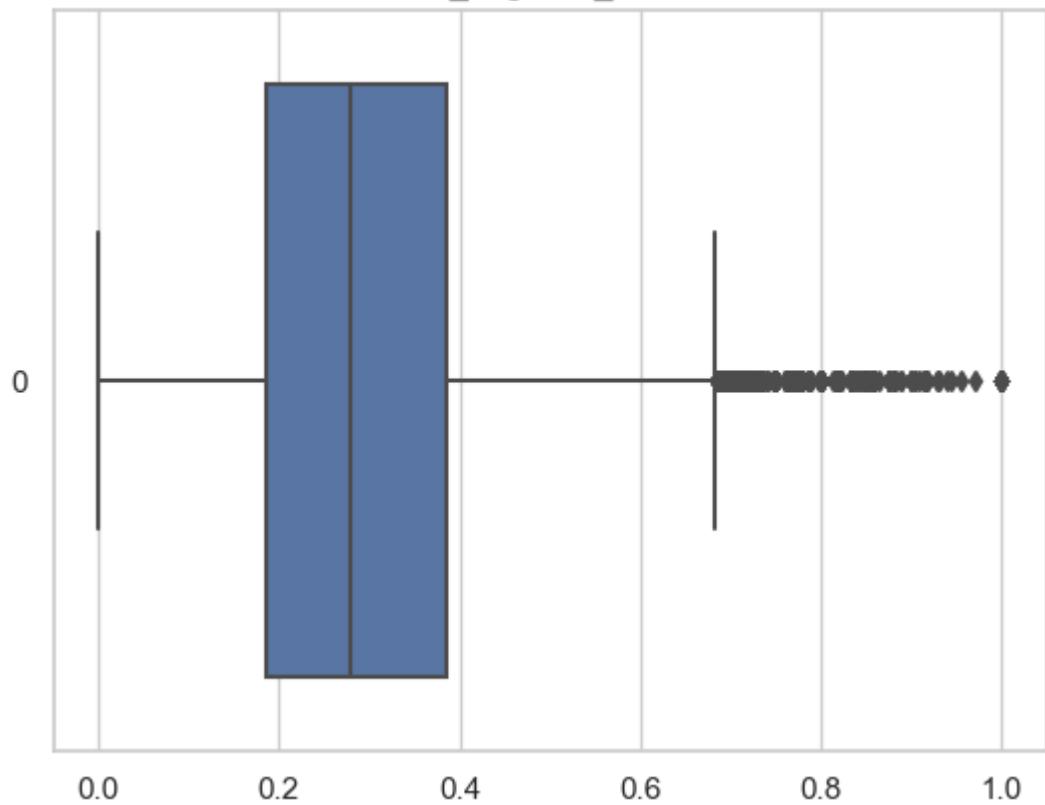
global\_rate\_negative\_words



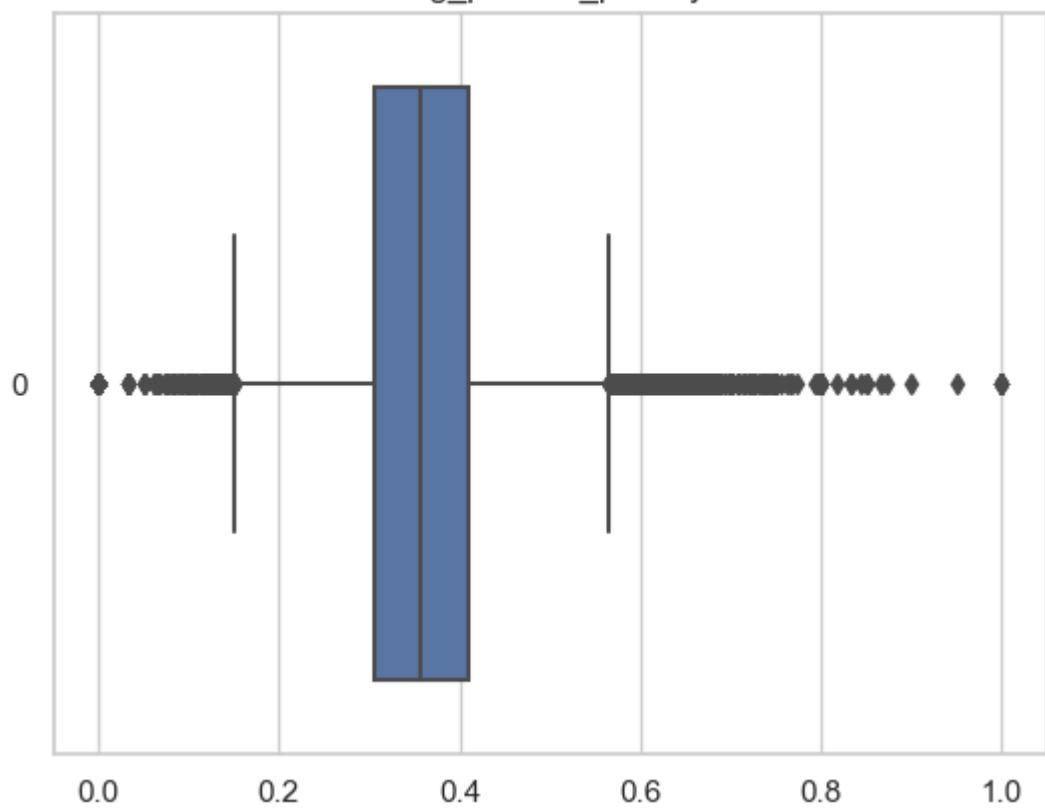
rate\_positive\_words



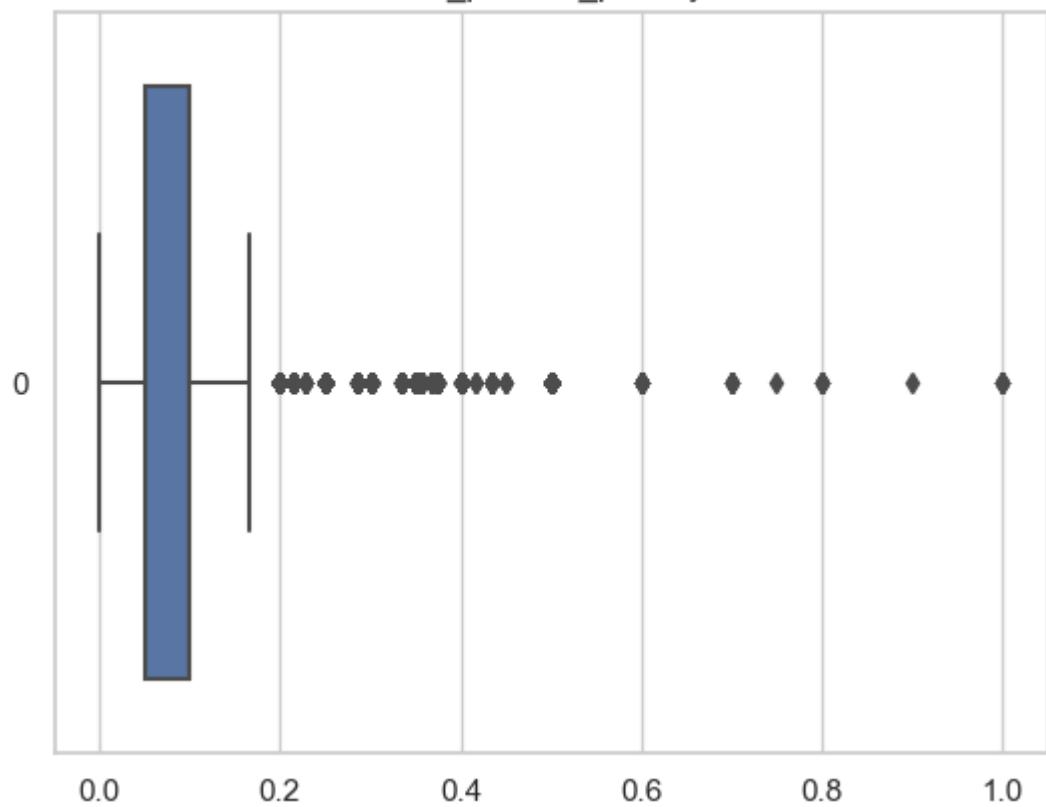
rate\_negative\_words



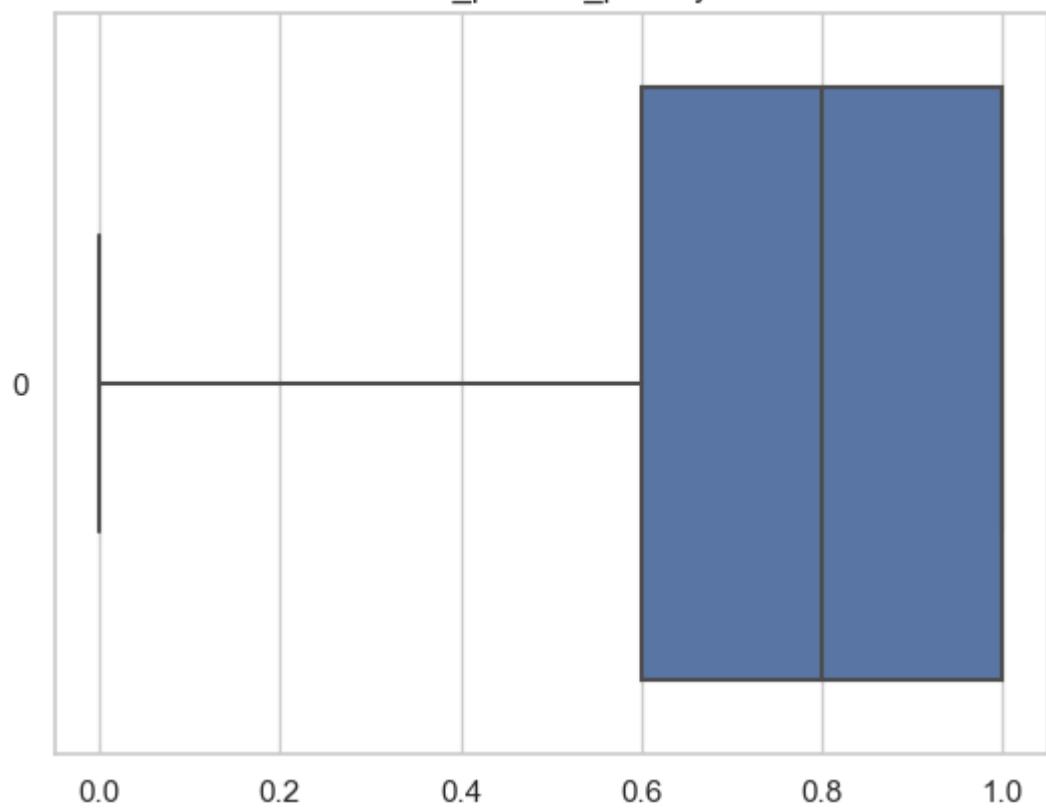
avg\_positive\_polarity



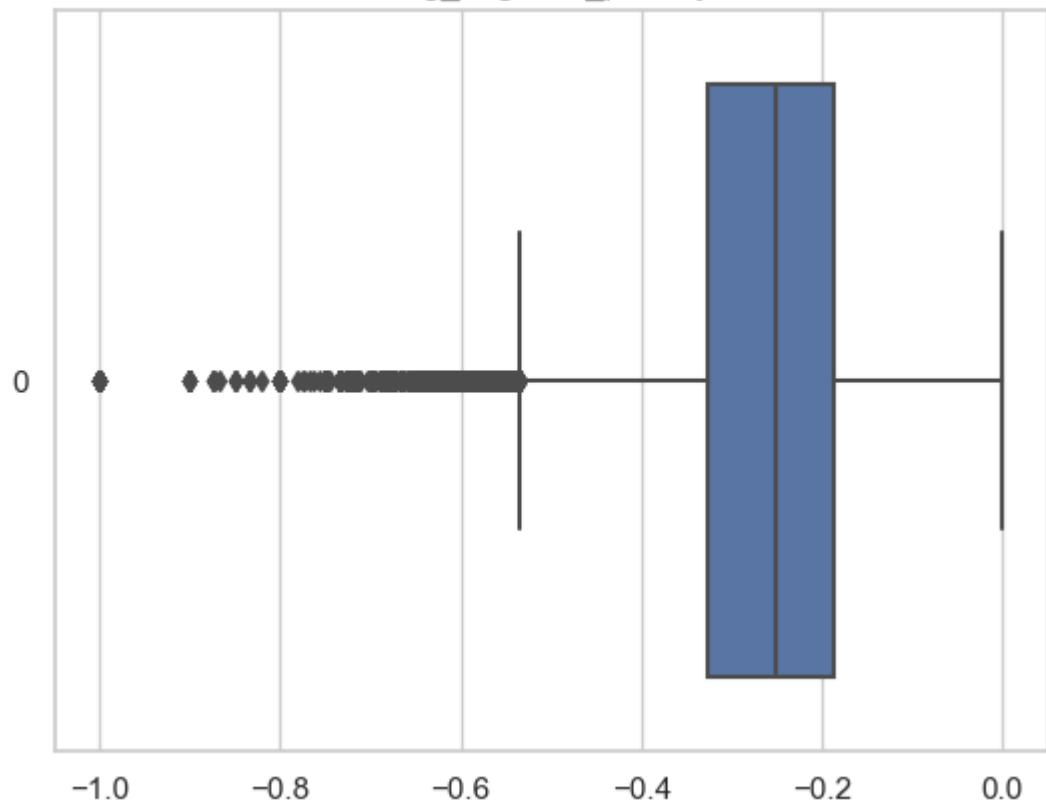
min\_positive\_polarity



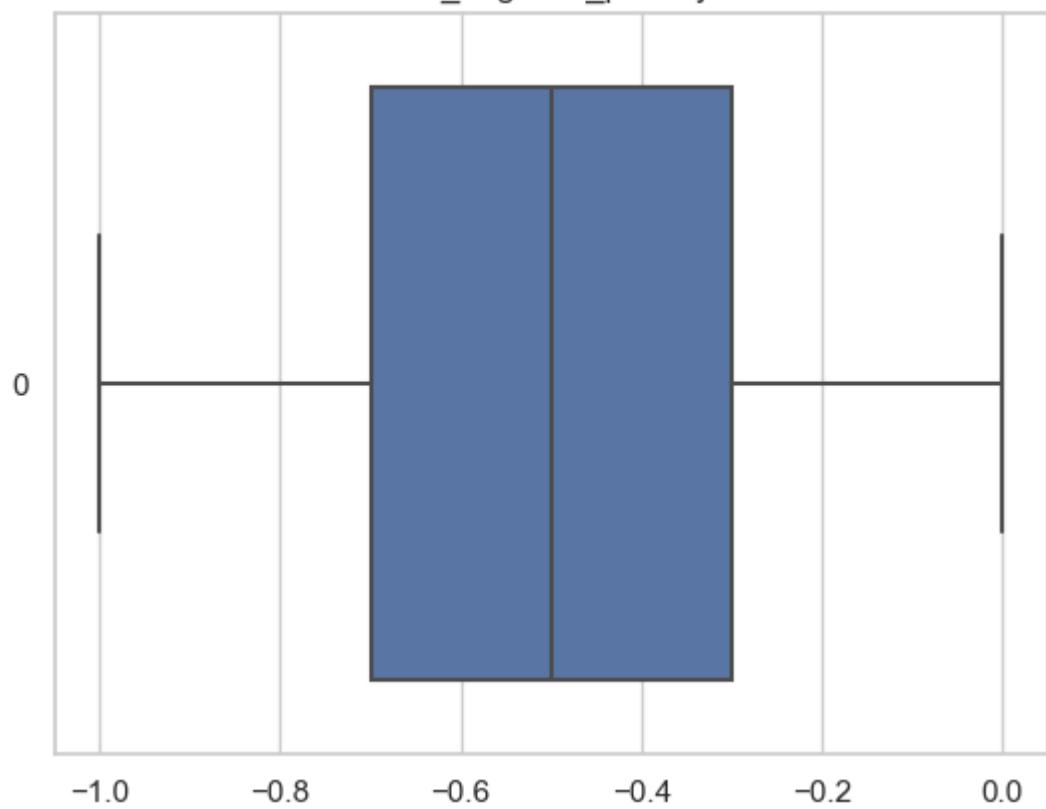
max\_positive\_polarity



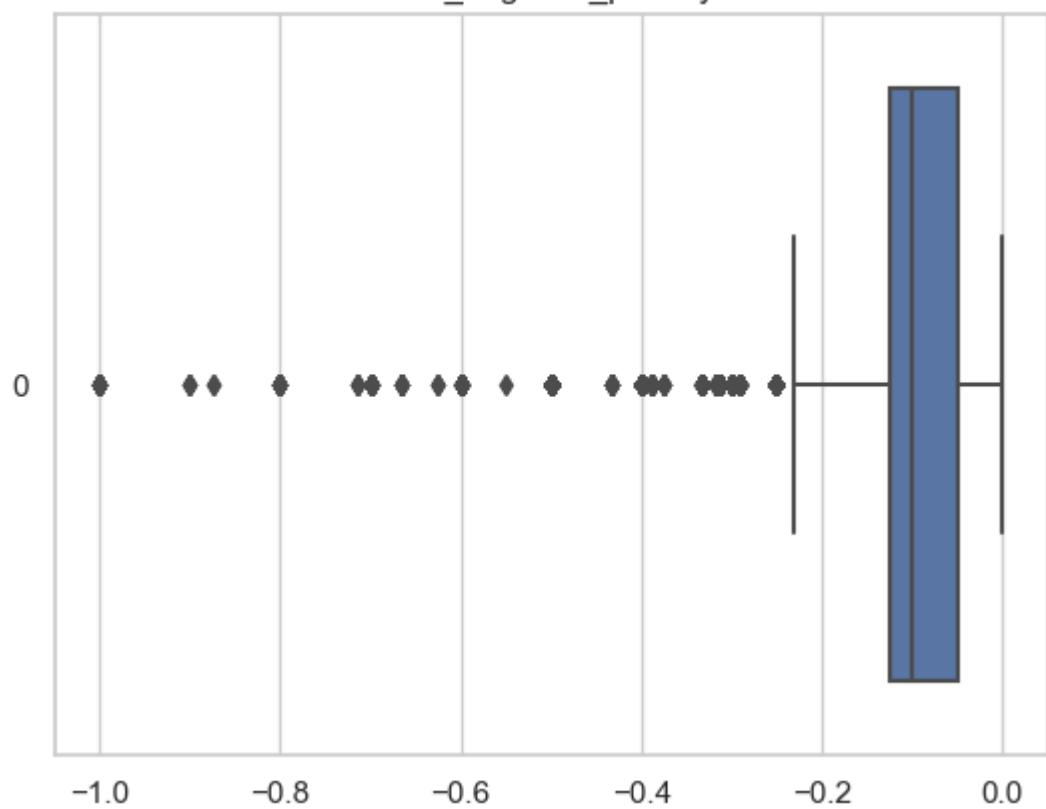
avg\_negative\_polarity



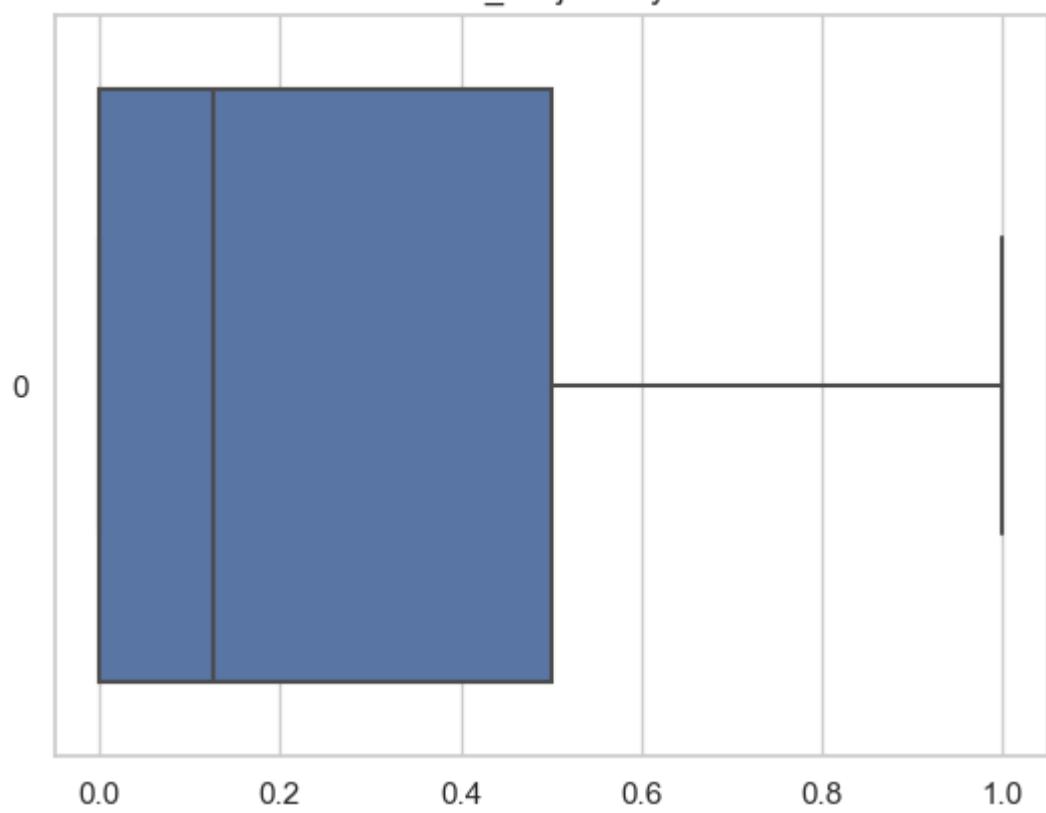
min\_negative\_polarity

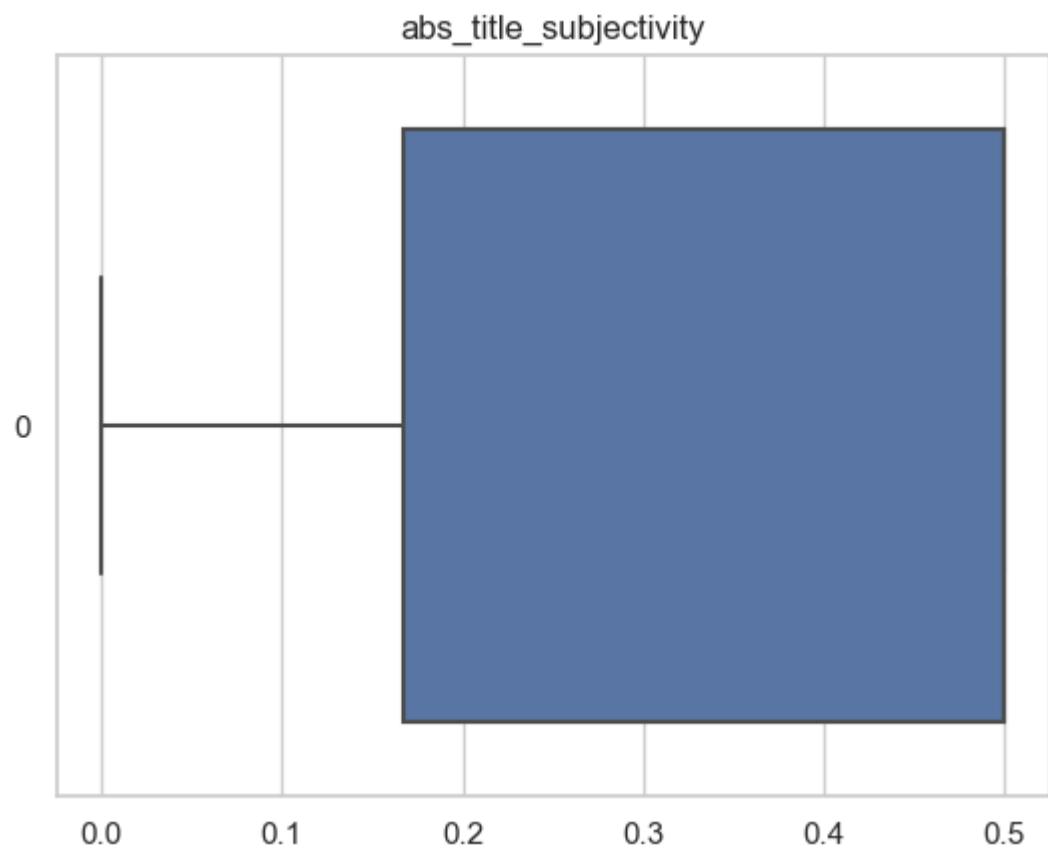
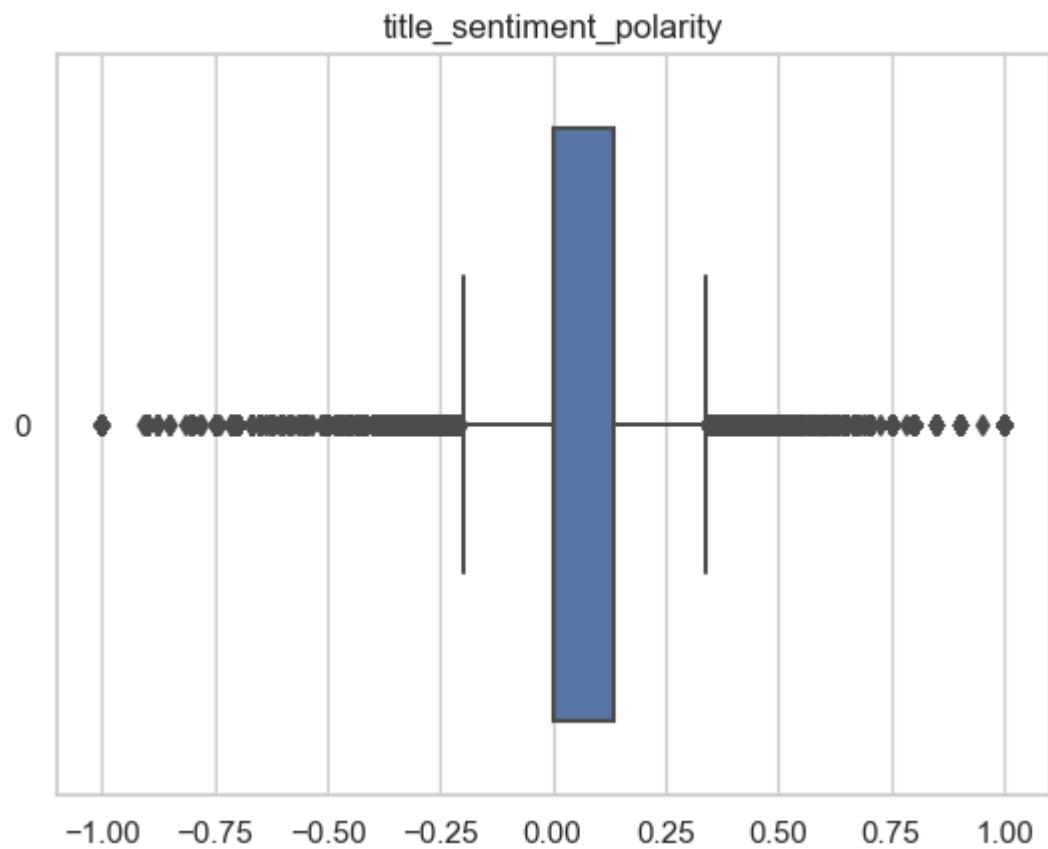


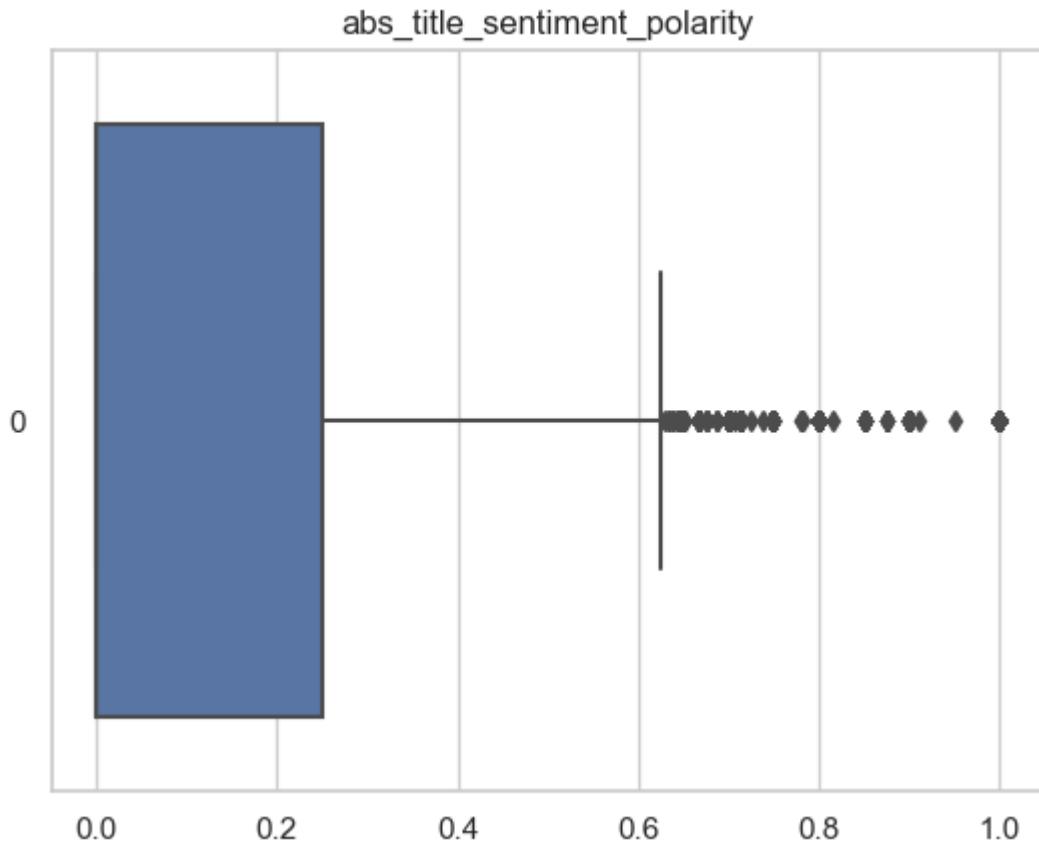
max\_negative\_polarity



title\_subjectivity







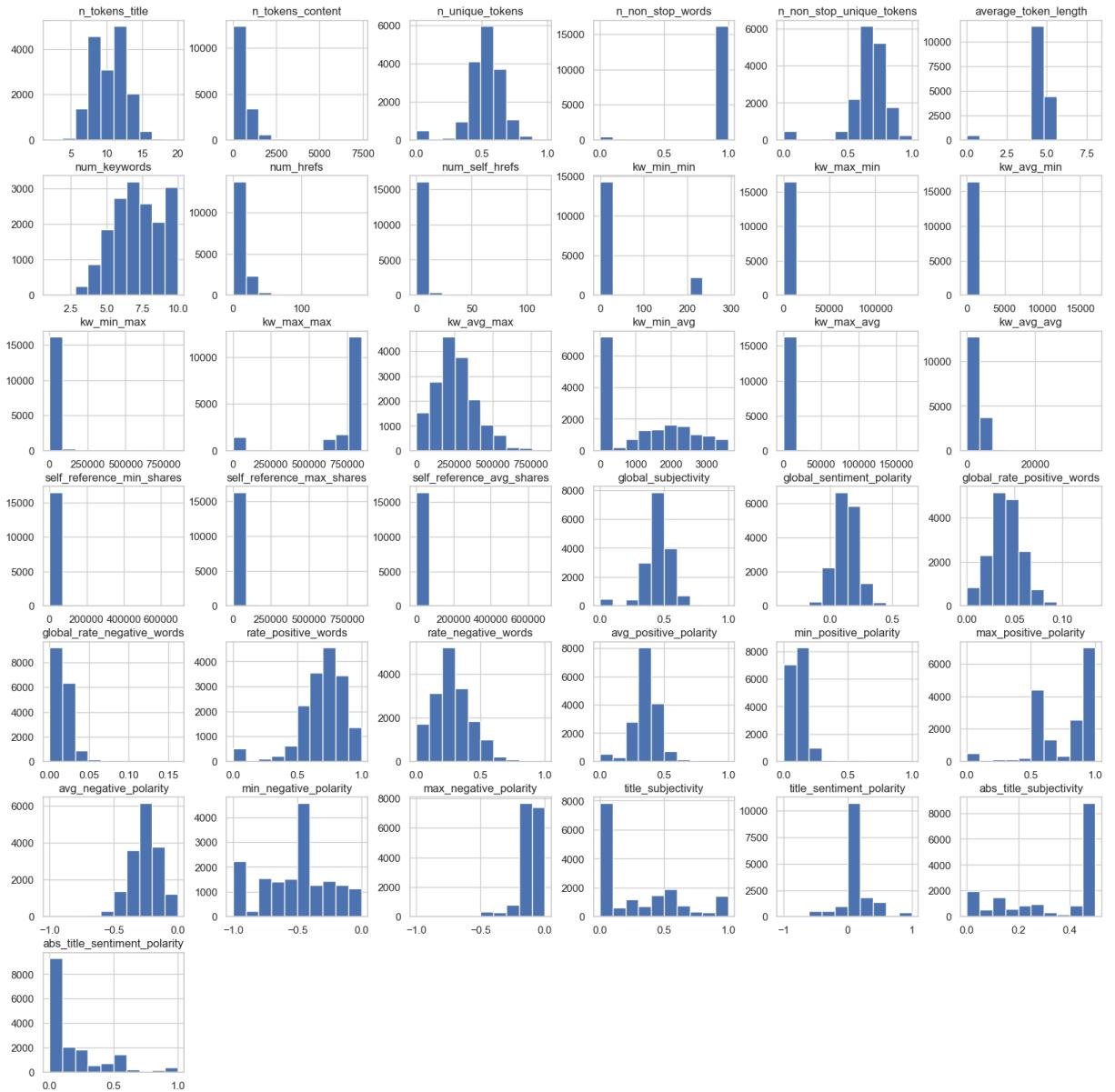
There's a lot of outliers. Thankfully, Xgboost can handle them so we're not going to spend a any more time on outliers.

```
In [22]: unpopular_df = df[df['shares'] < THRESHOLD ]
popular_df = df[df['shares'] >= THRESHOLD ]
```

Let's see histograms for our continuous variables.

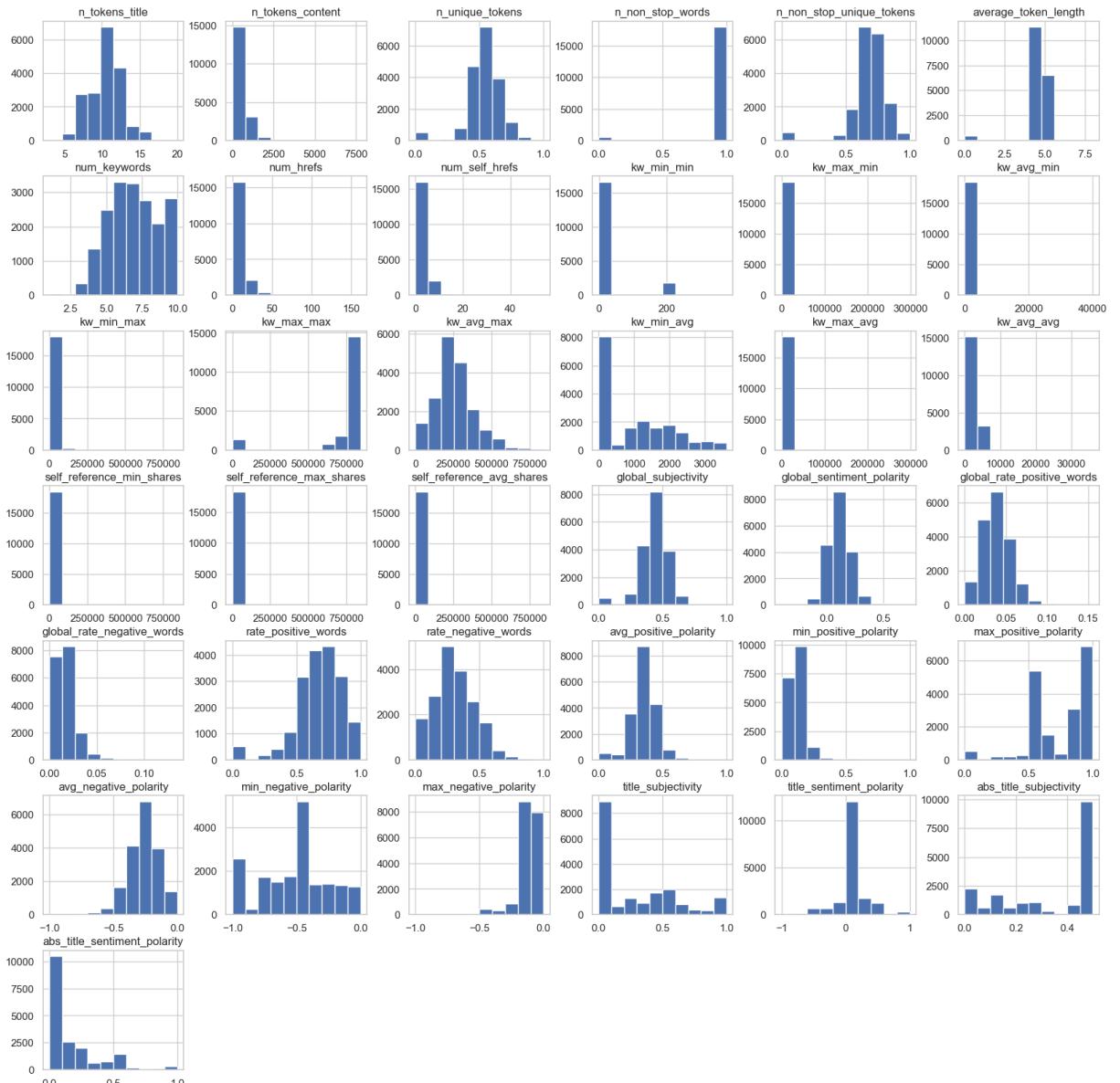
```
In [23]: popular_df[continuous_cols].hist(figsize=(20,20))
plt.show()
```

```
Out[23]: array([['<Axes: title={"center": "n_tokens_title"}>,<Axes: title={"center": "n_tokens_content"}>,<Axes: title={"center": "n_unique_tokens"}>,<Axes: title={"center": "n_non_stop_words"}>,<Axes: title={"center": "n_non_stop_unique_tokens"}>,<Axes: title={"center": "average_token_length"}>],['<Axes: title={"center": "num_keywords"}>,<Axes: title={"center": "num_hrefs"}>,<Axes: title={"center": "num_self_hrefs"}>,<Axes: title={"center": "kw_min_min"}>,<Axes: title={"center": "kw_max_min"}>,<Axes: title={"center": "kw_avg_min"}>],['<Axes: title={"center": "kw_min_max"}>,<Axes: title={"center": "kw_max_max"}>,<Axes: title={"center": "kw_avg_max"}>,<Axes: title={"center": "kw_min_avg"}>,<Axes: title={"center": "kw_max_avg"}>,<Axes: title={"center": "kw_avg_avg"}>],['<Axes: title={"center": "self_reference_min_shares"}>,<Axes: title={"center": "self_reference_max_shares"}>,<Axes: title={"center": "self_reference_avg_shares"}>,<Axes: title={"center": "global_subjectivity"}>,<Axes: title={"center": "global_sentiment_polarity"}>,<Axes: title={"center": "global_rate_positive_words"}>],['<Axes: title={"center": "global_rate_negative_words"}>,<Axes: title={"center": "rate_positive_words"}>,<Axes: title={"center": "rate_negative_words"}>,<Axes: title={"center": "avg_positive_polarity"}>,<Axes: title={"center": "min_positive_polarity"}>,<Axes: title={"center": "max_positive_polarity"}>],['<Axes: title={"center": "avg_negative_polarity"}>,<Axes: title={"center": "min_negative_polarity"}>,<Axes: title={"center": "max_negative_polarity"}>,<Axes: title={"center": "title_subjectivity"}>,<Axes: title={"center": "title_sentiment_polarity"}>,<Axes: title={"center": "abs_title_subjectivity"}>],['<Axes: title={"center": "abs_title_sentiment_polarity"}>,<Axes: >, <Axes: >, <Axes: >, <Axes: >], dtype=object)
```



```
In [24]: unpopular_df[continuous_cols].hist(figsize=(20,20))
plt.show()
```

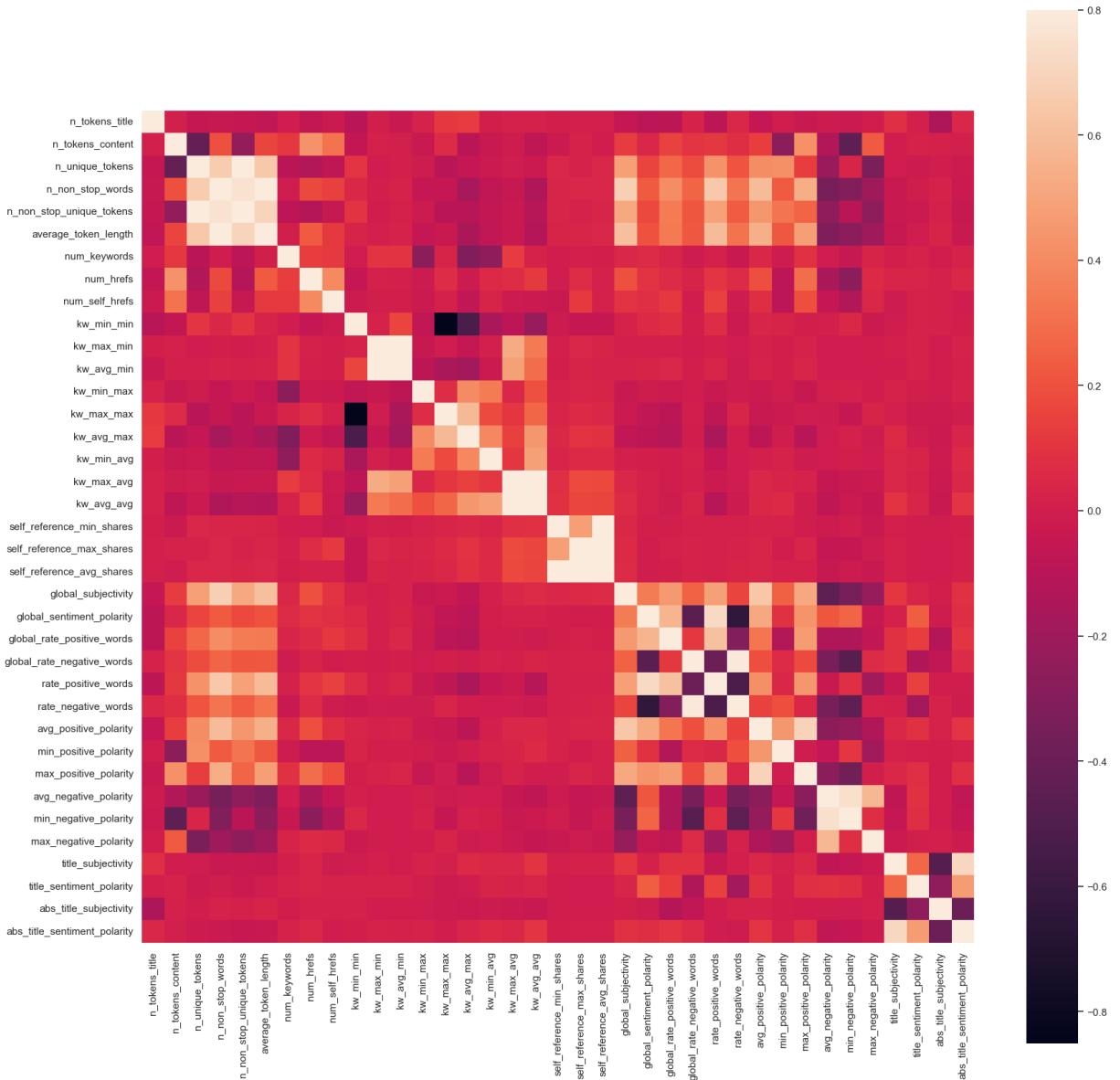
```
Out[24]: array([['<Axes: title={"center": "n_tokens_title"}>,<Axes: title={"center": "n_tokens_content"}>,<Axes: title={"center": "n_unique_tokens"}>,<Axes: title={"center": "n_non_stop_words"}>,<Axes: title={"center": "n_non_stop_unique_tokens"}>,<Axes: title={"center": "average_token_length"}>],['<Axes: title={"center": "num_keywords"}>,<Axes: title={"center": "num_hrefs"}>,<Axes: title={"center": "num_self_hrefs"}>,<Axes: title={"center": "kw_min_min"}>,<Axes: title={"center": "kw_max_min"}>,<Axes: title={"center": "kw_avg_min"}>],['<Axes: title={"center": "kw_min_max"}>,<Axes: title={"center": "kw_max_max"}>,<Axes: title={"center": "kw_avg_max"}>,<Axes: title={"center": "kw_min_avg"}>,<Axes: title={"center": "kw_max_avg"}>,<Axes: title={"center": "kw_avg_avg"}>],['<Axes: title={"center": "self_reference_min_shares"}>,<Axes: title={"center": "self_reference_max_shares"}>,<Axes: title={"center": "self_reference_avg_shares"}>,<Axes: title={"center": "global_subjectivity"}>,<Axes: title={"center": "global_sentiment_polarity"}>,<Axes: title={"center": "global_rate_positive_words"}>],['<Axes: title={"center": "global_rate_negative_words"}>,<Axes: title={"center": "rate_positive_words"}>,<Axes: title={"center": "rate_negative_words"}>,<Axes: title={"center": "avg_positive_polarity"}>,<Axes: title={"center": "min_positive_polarity"}>,<Axes: title={"center": "max_positive_polarity"}>],['<Axes: title={"center": "avg_negative_polarity"}>,<Axes: title={"center": "min_negative_polarity"}>,<Axes: title={"center": "max_negative_polarity"}>,<Axes: title={"center": "title_subjectivity"}>,<Axes: title={"center": "title_sentiment_polarity"}>,<Axes: title={"center": "abs_title_subjectivity"}>],['<Axes: title={"center": "abs_title_sentiment_polarity"}>,<Axes: >, <Axes: >, <Axes: >, <Axes: >], dtype=object)
```



Let's explore correlation between the variables.

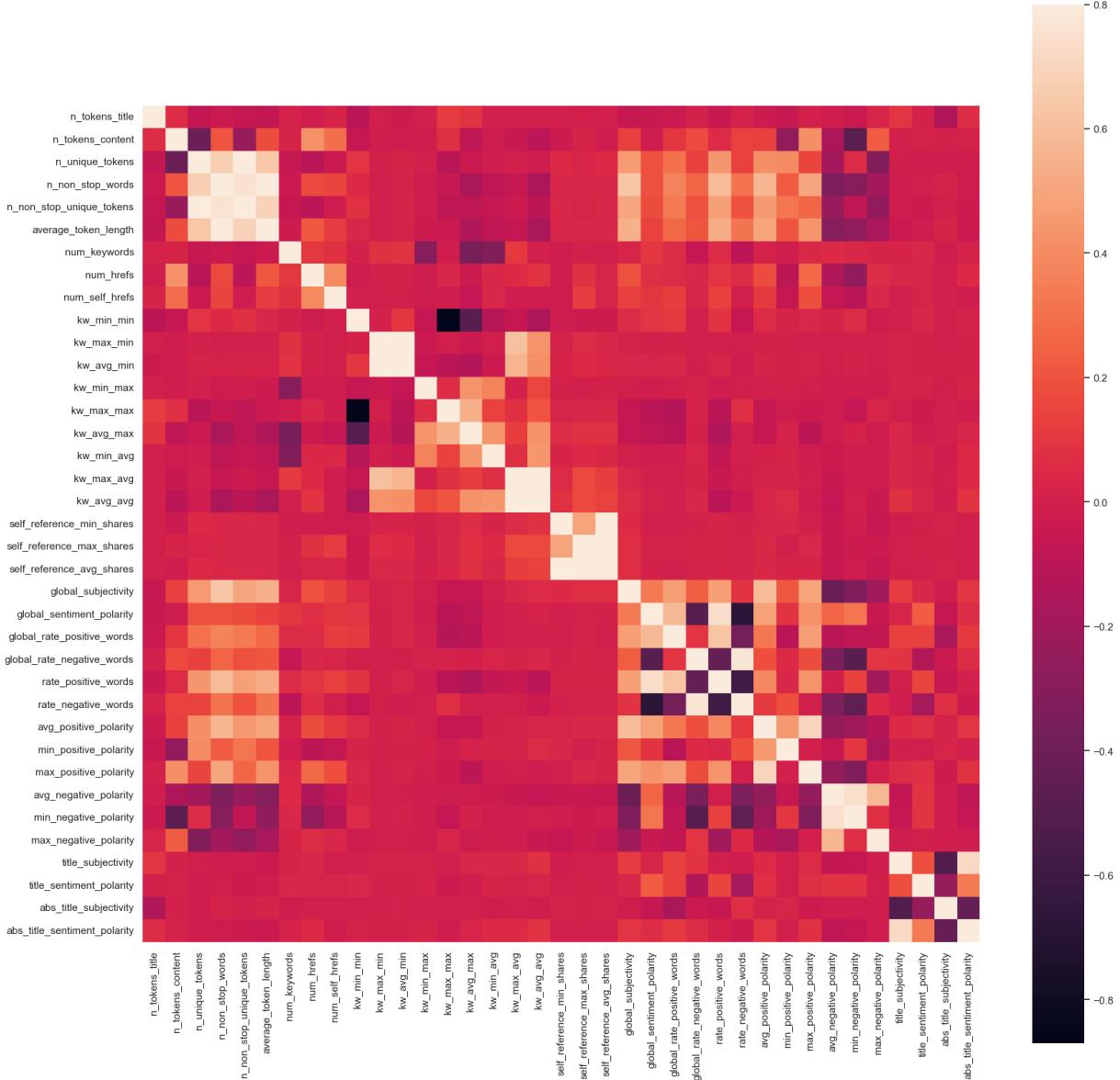
```
In [25]: corr = popular_df[continuous_cols].corr()
fig = plt.figure(figsize = (20,20))
sns.heatmap(corr, vmax = .8, square = True)
plt.show()
```

Out[25]: <Axes: >



```
In [26]: corr = unpopular_df[continuous_cols].corr()
fig = plt.figure(figsize = (20,20))
sns.heatmap(corr, vmax = .8, square = True)
plt.show()
```

Out[26]: <Axes: >



```
In [27]: # Adapted from
# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-c

# Numbers of pairs of bars you want
N = 7

# Data on X-axis

# Specify the values of blue bars (height)
popular_week = popular_df[week_cols].sum().values
# Specify the values of orange bars (height)
unpopular_week = unpopular_df[week_cols].sum().values

# Position of bars on x-axis
ind = np.arange(N)

# Figure size
plt.figure(figsize=(12,5))

# Width of a bar
```

```

width = 0.3

# Plotting
plt.bar(ind, popular_week , width, label='Popular')
plt.bar(ind + width, unpopular_week, width, label='Unpopular')

plt.xlabel('Days of the Week')
plt.ylabel('Count of News Articles')
plt.title('Count of Popular and Unpopular News Articles over Days of the Week')

# xticks()
# First argument - A List of positions at which ticks should be placed
# Second argument - A List of labels to place at the given locations
plt.xticks(ind + width / 2, ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'))

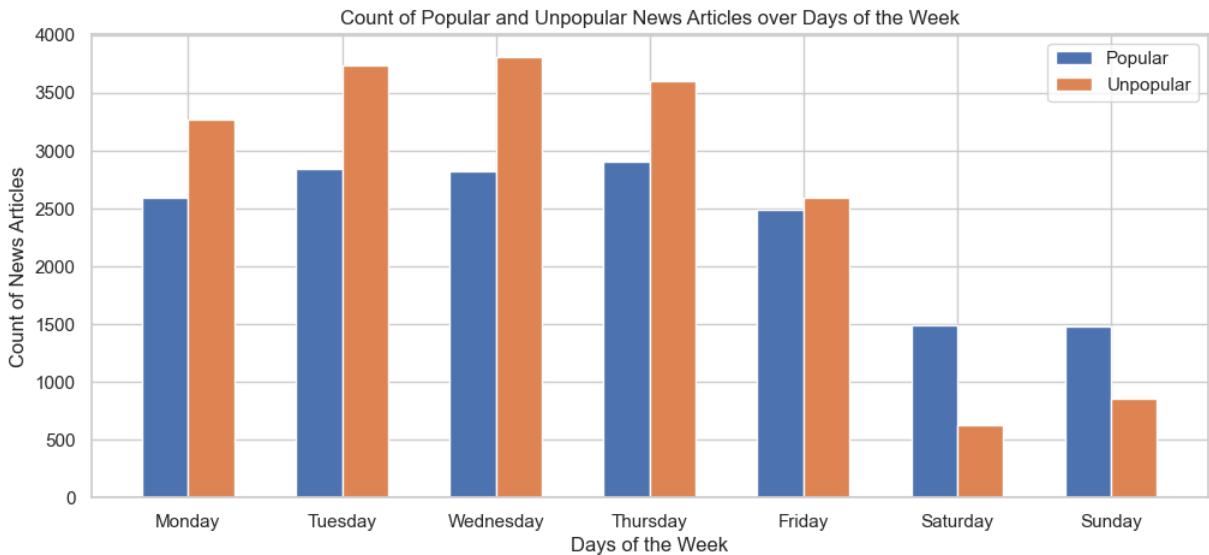
# Finding the best position for legends and putting it
plt.legend(loc='best')
plt.show()

```

```

Out[27]: <Figure size 1200x500 with 0 Axes>
Out[27]: <BarContainer object of 7 artists>
Out[27]: <BarContainer object of 7 artists>
Out[27]: Text(0.5, 0, 'Days of the Week')
Out[27]: Text(0, 0.5, 'Count of News Articles')
Out[27]: Text(0.5, 1.0, 'Count of Popular and Unpopular News Articles over Days of the Wee
k')
Out[27]: ([<matplotlib.axis.XTick at 0x1af571115b0>,
<matplotlib.axis.XTick at 0x1af57111580>,
<matplotlib.axis.XTick at 0x1af56c6d580>,
<matplotlib.axis.XTick at 0x1af5715e430>,
<matplotlib.axis.XTick at 0x1af5713de50>,
<matplotlib.axis.XTick at 0x1af5715eeee0>,
<matplotlib.axis.XTick at 0x1af57165a00>],
[Text(0.15, 0, 'Monday'),
Text(1.15, 0, 'Tuesday'),
Text(2.15, 0, 'Wednesday'),
Text(3.15, 0, 'Thursday'),
Text(4.15, 0, 'Friday'),
Text(5.15, 0, 'Saturday'),
Text(6.15, 0, 'Sunday')])
Out[27]: <matplotlib.legend.Legend at 0x1af5712ff40>

```



```
In [28]: # Adapted from
# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-c

# Numbers of pairs of bars you want
N = 7

# Data on X-axis

# Specify the values of blue bars (height)
popular_week = popular_df[week_cols].sum().values
# Specify the values of orange bars (height)
unpopular_week = unpopular_df[week_cols].sum().values

# Position of bars on x-axis
ind = np.arange(N)

# Figure size
plt.figure(figsize=(12,5))

# Width of a bar
width = 0.3

# Plotting
plt.bar(ind, popular_week , width, label='Popular')
plt.bar(ind + width, unpopular_week, width, label='Unpopular')

plt.xlabel('Days of the Week')
plt.ylabel('Count of News Articles')
plt.title('Count of Popular and Unpopular News Articles over Days of the Week')

# xticks()
# First argument - A list of positions at which ticks should be placed
# Second argument - A list of labels to place at the given locations
plt.xticks(ind + width / 2, ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'))

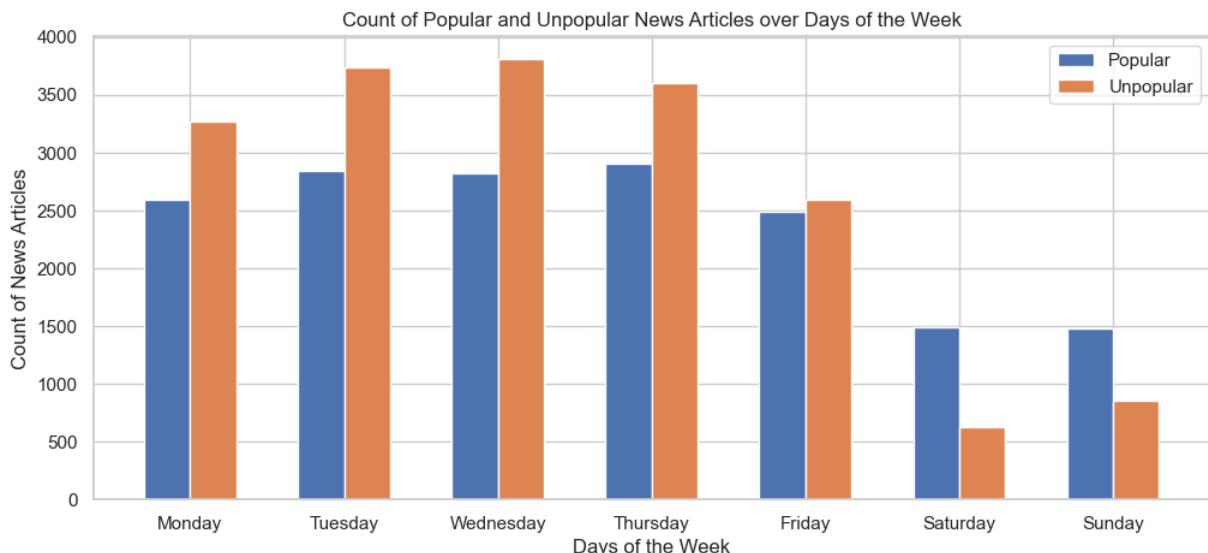
# Finding the best position for legends and putting it
plt.legend(loc='best')
plt.show()
```

```

Out[28]: <Figure size 1200x500 with 0 Axes>
Out[28]: <BarContainer object of 7 artists>
Out[28]: <BarContainer object of 7 artists>
Out[28]: Text(0.5, 0, 'Days of the Week')
Out[28]: Text(0, 0.5, 'Count of News Articles')
Out[28]: Text(0.5, 1.0, 'Count of Popular and Unpopular News Articles over Days of the Week')
Out[28]: ([<matplotlib.axis.XTick at 0x1af57107700>,
           <matplotlib.axis.XTick at 0x1af571078b0>,
           <matplotlib.axis.XTick at 0x1af57107c40>,
           <matplotlib.axis.XTick at 0x1af570e1700>,
           <matplotlib.axis.XTick at 0x1af571bef70>,
           <matplotlib.axis.XTick at 0x1af571c3790>,
           <matplotlib.axis.XTick at 0x1af571c3f70>],
           [Text(0.15, 0, 'Monday'),
            Text(1.15, 0, 'Tuesday'),
            Text(2.15, 0, 'Wednesday'),
            Text(3.15, 0, 'Thursday'),
            Text(4.15, 0, 'Friday'),
            Text(5.15, 0, 'Saturday'),
            Text(6.15, 0, 'Sunday')])

Out[28]: <matplotlib.legend.Legend at 0x1af5742c370>

```



```

In [29]: # Adapted from
          # https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-c

          # Numbers of pairs of bars you want
N = 6

          # Data on X-axis

          # Specify the values of blue bars (height)
popular_week = popular_df[channel_cols].sum().values
          # Specify the values of orange bars (height)

```

```

unpopular_week = unpopular_df[channel_cols].sum().values

# Position of bars on x-axis
ind = np.arange(N)

# Figure size
plt.figure(figsize=(12,5))

# Width of a bar
width = 0.3

# Plotting
plt.bar(ind, popular_week , width, label='Popular')
plt.bar(ind + width, unpopular_week, width, label='Unpopular')

plt.xlabel('Channels')
plt.ylabel('Count of News Articles')
plt.title('Count of Popular and Unpopular News Articles over Days of the Week')

# xticks()
# First argument - A List of positions at which ticks should be placed
# Second argument - A List of labels to place at the given locations
plt.xticks(ind + width / 2, ('Lifestyle', 'Entertainment', 'Business', 'Social Medi

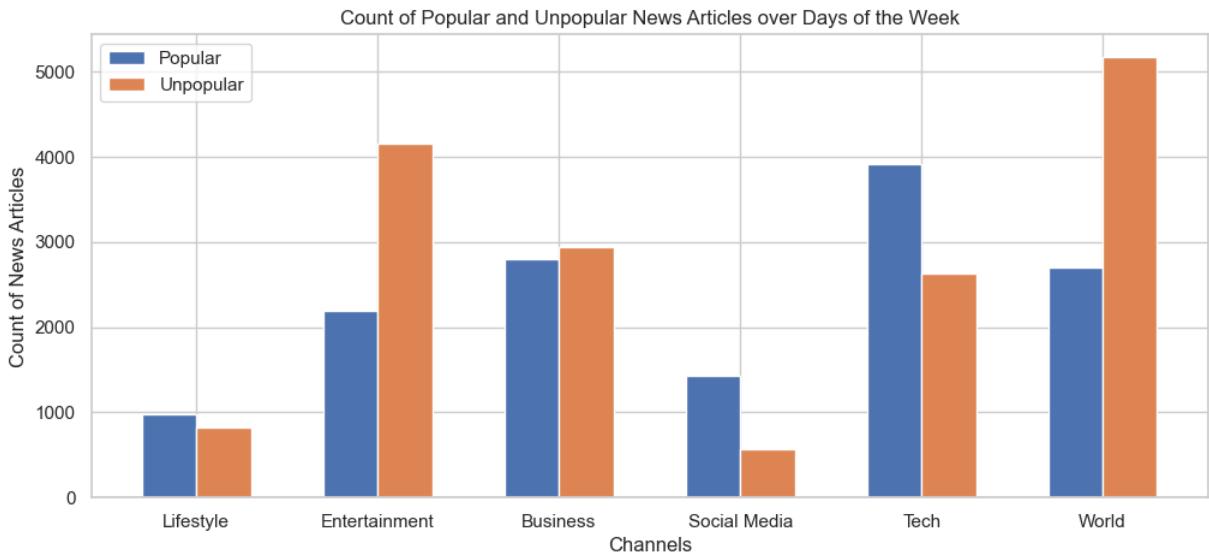
# Finding the best position for legends and putting it
plt.legend(loc='best')
plt.show()

```

```

Out[29]: <Figure size 1200x500 with 0 Axes>
Out[29]: <BarContainer object of 6 artists>
Out[29]: <BarContainer object of 6 artists>
Out[29]: Text(0.5, 0, 'Channels')
Out[29]: Text(0, 0.5, 'Count of News Articles')
Out[29]: Text(0.5, 1.0, 'Count of Popular and Unpopular News Articles over Days of the Wee
k')
Out[29]: ([<matplotlib.axis.XTick at 0x1af5720e100>,
<matplotlib.axis.XTick at 0x1af5720e0d0>,
<matplotlib.axis.XTick at 0x1af573fb310>,
<matplotlib.axis.XTick at 0x1af57251430>,
<matplotlib.axis.XTick at 0x1af57251cd0>,
<matplotlib.axis.XTick at 0x1af57251f10>],
[Text(0.15, 0, 'Lifestyle'),
Text(1.15, 0, 'Entertainment'),
Text(2.15, 0, 'Business'),
Text(3.15, 0, 'Social Media'),
Text(4.15, 0, 'Tech'),
Text(5.15, 0, 'World')])
Out[29]: <matplotlib.legend.Legend at 0x1af5720e1f0>

```



```
In [30]: # Adapted from
# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-c

# Numbers of pairs of bars you want
N = 5

# Data on X-axis

# Specify the values of blue bars (height)
popular_week = popular_df[topic_cols].sum().values
# Specify the values of orange bars (height)
unpopular_week = unpopular_df[topic_cols].sum().values

# Position of bars on x-axis
ind = np.arange(N)

# Figure size
plt.figure(figsize=(12,5))

# Width of a bar
width = 0.3

# Plotting
plt.bar(ind, popular_week , width, label='Popular')
plt.bar(ind + width, unpopular_week, width, label='Unpopular')

plt.xlabel('LDA Topic')
plt.ylabel('Count of News Articles')
plt.title('Count of Popular and Unpopular News Articles over LDA Topics')

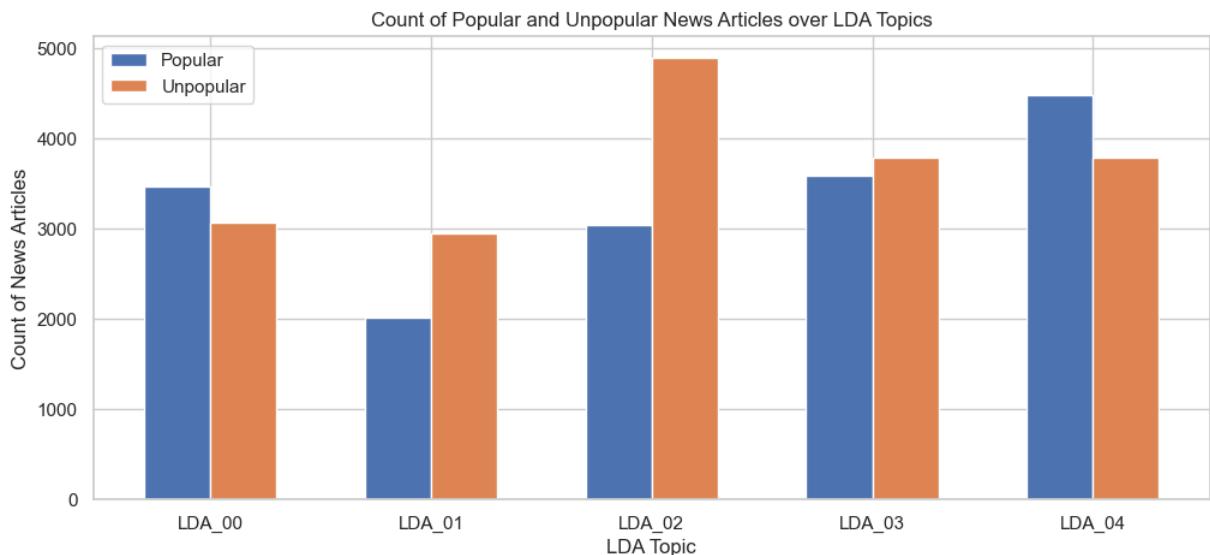
# xticks()
# First argument - A list of positions at which ticks should be placed
# Second argument - A list of labels to place at the given locations
plt.xticks(ind + width / 2, ('LDA_00', 'LDA_01', 'LDA_02', 'LDA_03', 'LDA_04'))

# Finding the best position for legends and putting it
plt.legend(loc='best')
plt.show()
```

```

Out[30]: <Figure size 1200x500 with 0 Axes>
Out[30]: <BarContainer object of 5 artists>
Out[30]: <BarContainer object of 5 artists>
Out[30]: Text(0.5, 0, 'LDA Topic')
Out[30]: Text(0, 0.5, 'Count of News Articles')
Out[30]: Text(0.5, 1.0, 'Count of Popular and Unpopular News Articles over LDA Topics')
Out[30]: ([<matplotlib.axis.XTick at 0x1af5728ce80>,
           <matplotlib.axis.XTick at 0x1af5728ce50>,
           <matplotlib.axis.XTick at 0x1af5728ca60>,
           <matplotlib.axis.XTick at 0x1af572c7f40>,
           <matplotlib.axis.XTick at 0x1af572d8bb0>],
           [Text(0.15, 0, 'LDA_00'),
            Text(1.15, 0, 'LDA_01'),
            Text(2.15, 0, 'LDA_02'),
            Text(3.15, 0, 'LDA_03'),
            Text(4.15, 0, 'LDA_04')])
Out[30]: <matplotlib.legend.Legend at 0x1af57294640>

```



```

In [31]: ttest_same = []
ttest_diff = []

for column in continuous_cols:

    result = stats.ttest_ind(popular_df[column], unpopular_df[column])[1]

    if result > ALPHA:
        interpretation = 'insignificant - SAME'
        ttest_same.append(column)
    else:
        interpretation = 'significant - DIFFERENT'
        ttest_diff.append(column)

    print(result, '-', column, ' - ', interpretation)

```

```
2.2504282720304512e-19 - n_tokens_title - significant - DIFFERENT
1.540713115573688e-19 - n_tokens_content - significant - DIFFERENT
2.9880436047459157e-20 - n_unique_tokens - significant - DIFFERENT
0.1936199104088964 - n_non_stop_words - insignificant - SAME
1.3901924209284012e-20 - n_non_stop_unique_tokens - significant - DIFFERENT
0.0008373135389349559 - average_token_length - significant - DIFFERENT
5.124818509804509e-38 - num_keywords - significant - DIFFERENT
2.864037347216668e-48 - num_hrefs - significant - DIFFERENT
1.8442399645567025e-18 - num_self_hrefs - significant - DIFFERENT
5.950648914047732e-29 - kw_min_min - significant - DIFFERENT
5.100318754407514e-05 - kw_max_min - significant - DIFFERENT
1.344140676190898e-10 - kw_avg_min - significant - DIFFERENT
0.4989833952664203 - kw_min_max - insignificant - SAME
4.375358509213268e-17 - kw_max_max - significant - DIFFERENT
0.5208306960828408 - kw_avg_max - insignificant - SAME
3.159778399488084e-47 - kw_min_avg - significant - DIFFERENT
1.7754195612623336e-20 - kw_max_avg - significant - DIFFERENT
3.146230642300792e-116 - kw_avg_avg - significant - DIFFERENT
4.227667874173408e-14 - self_reference_min_shares - significant - DIFFERENT
1.872477551391861e-21 - self_reference_max_shares - significant - DIFFERENT
2.1416561555131787e-22 - self_reference_avg_shares - significant - DIFFERENT
8.83025051250046e-30 - global_subjectivity - significant - DIFFERENT
4.422571139138585e-50 - global_sentiment_polarity - significant - DIFFERENT
8.64088412791597e-36 - global_rate_positive_words - significant - DIFFERENT
1.1463628301422673e-10 - global_rate_negative_words - significant - DIFFERENT
7.128978169033561e-28 - rate_positive_words - significant - DIFFERENT
1.3342228191351876e-47 - rate_negative_words - significant - DIFFERENT
1.7680195246297564e-05 - avg_positive_polarity - significant - DIFFERENT
2.90326731691146e-10 - min_positive_polarity - significant - DIFFERENT
1.2660915642576633e-14 - max_positive_polarity - significant - DIFFERENT
0.013479436287143189 - avg_negative_polarity - significant - DIFFERENT
0.0838875232392592 - min_negative_polarity - insignificant - SAME
0.19100281659526497 - max_negative_polarity - insignificant - SAME
0.0003501111655936487 - title_subjectivity - significant - DIFFERENT
5.425254469115109e-21 - title_sentiment_polarity - significant - DIFFERENT
0.41820000733762275 - abs_title_subjectivity - insignificant - SAME
3.015299461334996e-07 - abs_title_sentiment_polarity - significant - DIFFERENT
```

```
In [32]: shap_yesn = []
shap_notn = []

for column in continuous_cols:

    stat, result = stats.shapiro(df[column])

    if result > ALPHA:
        interpretation = 'insignificant - NORMAL'
        shap_yesn.append(column)
    else:
        interpretation = 'significant - NOT NORMAL'
        shap_notn.append(column)

    print(result, '-', column, ' - ', interpretation)
```

```
0.0 - n_tokens_title - significant - NOT NORMAL
0.0 - n_tokens_content - significant - NOT NORMAL
0.0 - n_unique_tokens - significant - NOT NORMAL
0.0 - n_non_stop_words - significant - NOT NORMAL
0.0 - n_non_stop_unique_tokens - significant - NOT NORMAL
0.0 - average_token_length - significant - NOT NORMAL
0.0 - num_keywords - significant - NOT NORMAL
0.0 - num_hrefs - significant - NOT NORMAL
0.0 - num_self_hrefs - significant - NOT NORMAL
0.0 - kw_min_min - significant - NOT NORMAL
0.0 - kw_max_min - significant - NOT NORMAL
0.0 - kw_avg_min - significant - NOT NORMAL
0.0 - kw_min_max - significant - NOT NORMAL
0.0 - kw_max_max - significant - NOT NORMAL
0.0 - kw_avg_max - significant - NOT NORMAL
0.0 - kw_min_avg - significant - NOT NORMAL
0.0 - kw_max_avg - significant - NOT NORMAL
0.0 - kw_avg_avg - significant - NOT NORMAL
0.0 - self_reference_min_shares - significant - NOT NORMAL
0.0 - self_reference_max_shares - significant - NOT NORMAL
0.0 - self_reference_avg_shares - significant - NOT NORMAL
0.0 - global_subjectivity - significant - NOT NORMAL
8.407790785948902e-45 - global_sentiment_polarity - significant - NOT NORMAL
1.401298464324817e-45 - global_rate_positive_words - significant - NOT NORMAL
0.0 - global_rate_negative_words - significant - NOT NORMAL
0.0 - rate_positive_words - significant - NOT NORMAL
0.0 - rate_negative_words - significant - NOT NORMAL
0.0 - avg_positive_polarity - significant - NOT NORMAL
0.0 - min_positive_polarity - significant - NOT NORMAL
0.0 - max_positive_polarity - significant - NOT NORMAL
0.0 - avg_negative_polarity - significant - NOT NORMAL
0.0 - min_negative_polarity - significant - NOT NORMAL
0.0 - max_negative_polarity - significant - NOT NORMAL
0.0 - title_subjectivity - significant - NOT NORMAL
0.0 - title_sentiment_polarity - significant - NOT NORMAL
0.0 - abs_title_subjectivity - significant - NOT NORMAL
0.0 - abs_title_sentiment_polarity - significant - NOT NORMAL
```

```
In [33]: final_df = df.drop(columns=['shares'])
```

## MODELING

Let's split the data.

```
In [34]: X = final_df.loc[:, final_df.columns != 'target']
y = final_df.loc[:, final_df.columns == 'target']
```

```
In [35]: # Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

## Logistics Regression Models

```
In [36]: logreg0 = LogisticRegression()
logreg0.fit(X_train, y_train)
```

```
Out[36]: ▾ LogisticRegression
          LogisticRegression()
```

```
In [37]: print('Accuracy of logistic regression classifier on train set: {:.2f}'.format(logreg0.score(X_train, y_train)))
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg0.score(X_test, y_test)))
```

Accuracy of logistic regression classifier on train set: 0.59  
Accuracy of logistic regression classifier on test set: 0.59

```
In [38]: y_pred = logreg0.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.60	0.84	0.70	5952
1	0.56	0.26	0.35	4579
accuracy			0.59	10531
macro avg	0.58	0.55	0.52	10531
weighted avg	0.58	0.59	0.55	10531

```
In [39]: # parameter grid
parameters = {
    'penalty' : ['l1','l2'],
    'C'       : np.logspace(-3,3,7),
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
}
```

```
In [40]: logreg1 = LogisticRegression()
clf = GridSearchCV(logreg1,                      # model
                    param_grid = parameters,   # hyperparameters
                    scoring='accuracy',      # metric for scoring
                    cv=10)                   # number of folds

clf.fit(X_train,y_train)
```

```
Out[40]: ▾ GridSearchCV
  ▾ estimator: LogisticRegression
    ▾ LogisticRegression
```

```
In [41]: print("Tuned Hyperparameters :", clf.best_params_)
print("Logistic Regression) Accuracy :",clf.best_score_)
```

Tuned Hyperparameters : {'C': 1.0, 'penalty': 'l1', 'solver': 'liblinear'}  
Logistic Regression) Accuracy : 0.6560726017194691

```
In [42]: logreg2 = LogisticRegression(C = 10,
                                    penalty = 'l1',
                                    solver = 'liblinear')
logreg2.fit(X_train,y_train)

print('Accuracy of logistic regression classifier on train set: {:.2f}'.format(logreg2.score(X_train, y_train)))
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg2.score(X_test, y_test)))
```

```
Out[42]: LogisticRegression
LogisticRegression(C=10, penalty='l1', solver='liblinear')
```

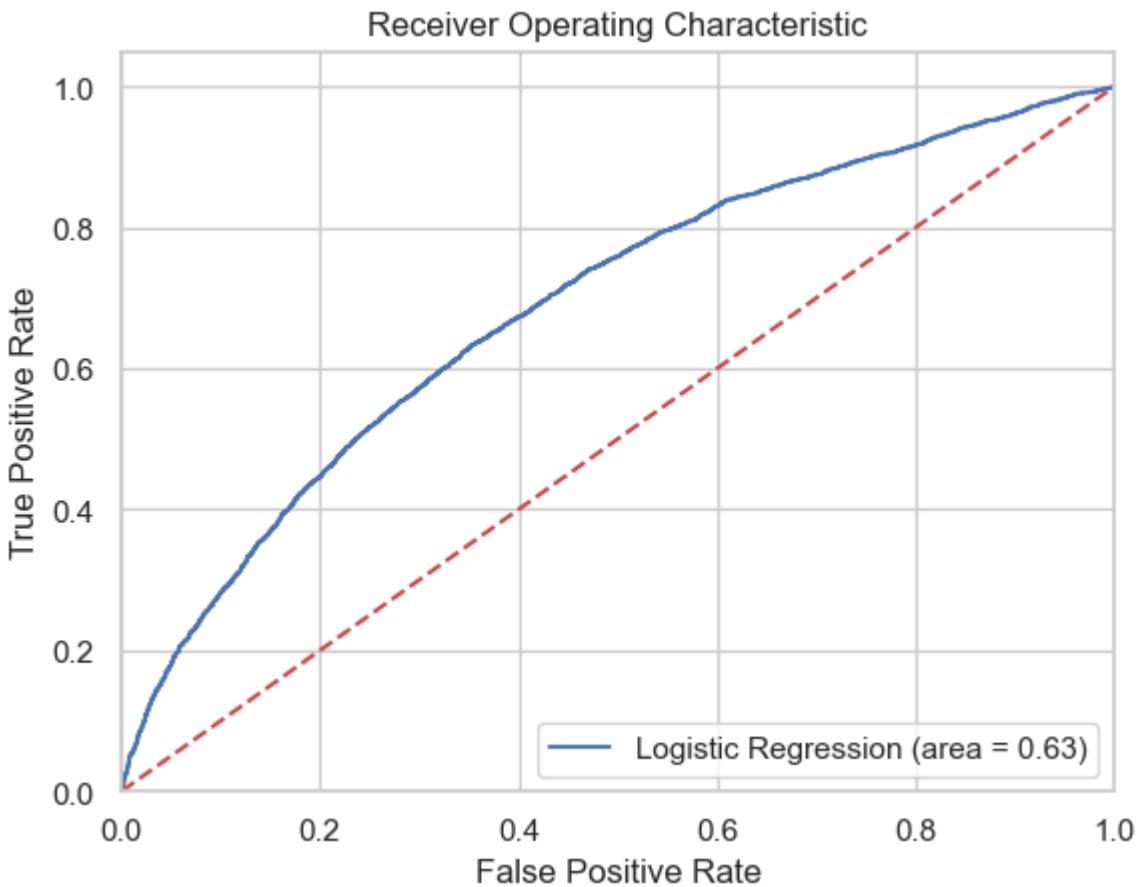
Accuracy of logistic regression classifier on train set: 0.66  
Accuracy of logistic regression classifier on test set: 0.65

```
In [43]: y_pred = logreg2.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.65	0.80	0.72	5952
1	0.63	0.46	0.53	4579
accuracy			0.65	10531
macro avg	0.64	0.63	0.62	10531
weighted avg	0.64	0.65	0.64	10531

```
In [44]: logreg_roc_auc = roc_auc_score(y_test, logreg2.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logreg2.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logreg_roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
```

```
Out[44]: <Figure size 640x480 with 0 Axes>
Out[44]: []
Out[44]: []
Out[44]: (0.0, 1.0)
Out[44]: (0.0, 1.05)
Out[44]: Text(0.5, 0, 'False Positive Rate')
Out[44]: Text(0, 0.5, 'True Positive Rate')
Out[44]: Text(0.5, 1.0, 'Receiver Operating Characteristic')
Out[44]: <matplotlib.legend.Legend at 0x1af5734b760>
```



## Xgboost Models

```
In [45]: # initial XGBOOST model
xgb0 = XGBClassifier(tree_method = 'gpu_hist')
xgb0.fit(X_train, y_train)
```

Out[45]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types
= None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type
= None,
              interaction_constraints=None, learning_rate=None, max_bin
= None,
```

```
In [46]: print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb0.score(X_trai
print('Accuracy of xgboost regression classifier on test set: {:.2f}'.format(xgb0.s
```

Accuracy of xgboost classifier on train set: 0.90  
Accuracy of xgboost regression classifier on test set: 0.64

```
In [47]: y_pred = xgb0.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.66	0.75	0.70	5952
1	0.61	0.50	0.55	4579
accuracy			0.64	10531
macro avg	0.64	0.63	0.63	10531
weighted avg	0.64	0.64	0.64	10531

```
In [48]: pipe = Pipeline([
    ('fs', SelectKBest()),
    ('clf', xgb.XGBClassifier(objective='binary:logistic'))
])
```

```
In [49]: # Define our search space for grid search
search_space = [
    {
        'clf__n_estimators': [100, 200],
        'clf__learning_rate': [0.1, 0.01],
        'clf__max_depth': [3, 4, 5],
        'clf__colsample_bytree': [0.1, 0.2],
        'clf__gamma': [0],
        'clf__tree_method': ['gpu_hist'],
        'fs__score_func': [f_classif],
        'fs__k': [10],
    }
]
```

```
In [50]: # Define cross validation
kfold = KFold(n_splits=10)
# AUC and accuracy as score
scoring = {'AUC':'roc_auc', 'Accuracy':make_scorer(accuracy_score)}
```

```
In [51]: # Define grid search
grid = GridSearchCV(
    pipe,
    param_grid=search_space,
    cv=kfold,
    scoring=scoring,
    refit='AUC',
    verbose=1,
    n_jobs=-1
)
# Fit grid search
xgb1 = grid.fit(X_train, y_train)
```

Fitting 10 folds for each of 24 candidates, totalling 240 fits

```
In [52]: print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb1.score(X_tr
print('Accuracy of xgboost regression classifier on test set: {:.2f}'.format(xgb1.s
```

```
Accuracy of xgboost classifier on train set: 0.69
Accuracy of xgboost regression classifier on test set: 0.67
```

```
In [53]: y_pred = xgb1.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.65	0.80	0.71	5952
1	0.62	0.43	0.51	4579
accuracy			0.64	10531
macro avg	0.64	0.62	0.61	10531
weighted avg	0.64	0.64	0.63	10531

```
In [54]: print(xgb1.best_params_)
```

```
{'clf__colsample_bytree': 0.2, 'clf__gamma': 0, 'clf__learning_rate': 0.1, 'clf__max_depth': 3, 'clf__n_estimators': 200, 'clf__tree_method': 'gpu_hist', 'fs__k': 10, 'fs__score_func': <function f_classif at 0x000001AF47C4C0D0>}
```

```
In [55]: xgb2 = XGBClassifier(colsample_bytree=.2,
                           gamma=0,
                           learning_rate=0.1,
                           max_depth=4,
                           n_estimators=200,
                           tree_method = 'gpu_hist'
                           )
xgb2.fit(X_train, y_train)
```

```
Out[55]:
```

```
▼          XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=0.2, early_stopping_rounds=None,
               enable_categorical=False, eval_metric=None, feature_types
= None,
               gamma=0, gpu_id=None, grow_policy=None, importance_type=N
one,
               interaction_constraints=None, learning_rate=0.1, max_bin=
None,
```

```
In [56]: print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb2.score(X_tr
print('Accuracy of xgboost regression classifier on test set: {:.2f}'.format(xgb2.s
```

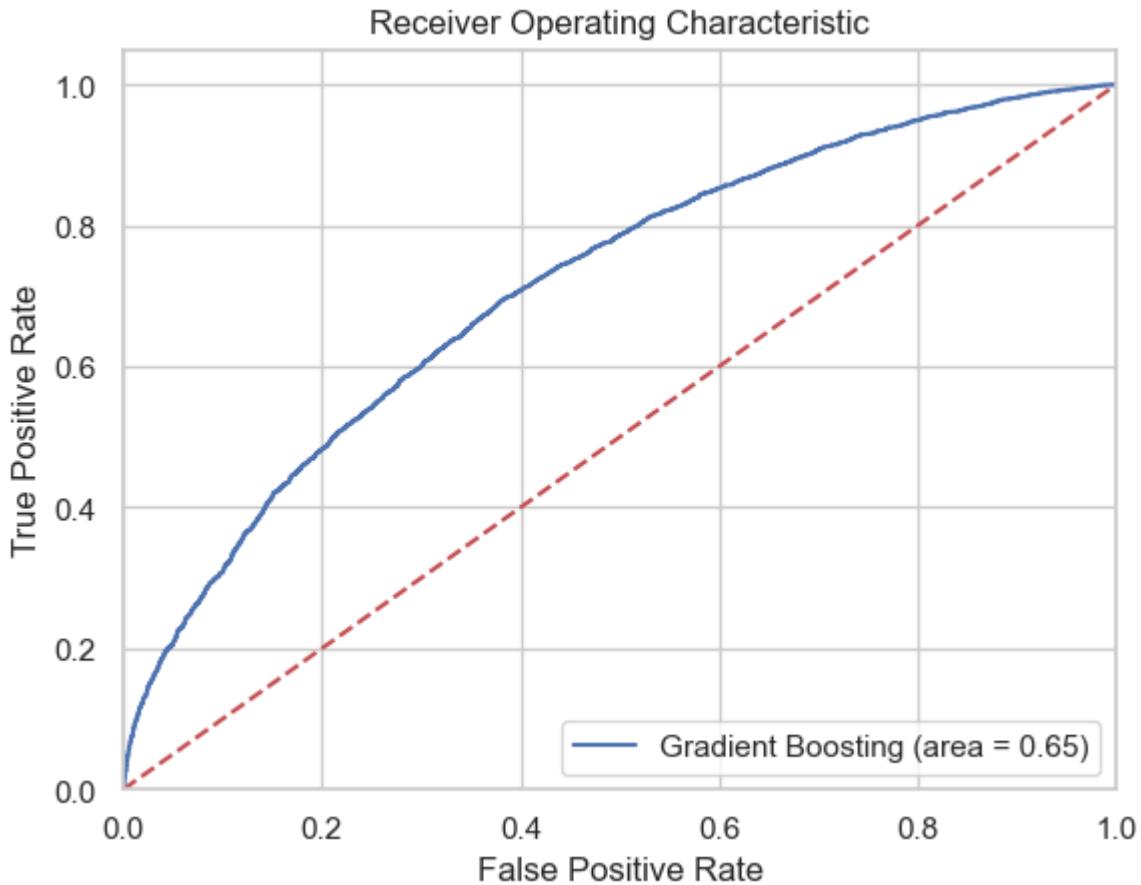
```
Accuracy of xgboost classifier on train set: 0.73
Accuracy of xgboost regression classifier on test set: 0.66
```

```
In [57]: y_pred = xgb2.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.67	0.79	0.72	5952
1	0.64	0.50	0.57	4579
accuracy			0.66	10531
macro avg	0.66	0.65	0.65	10531
weighted avg	0.66	0.66	0.66	10531

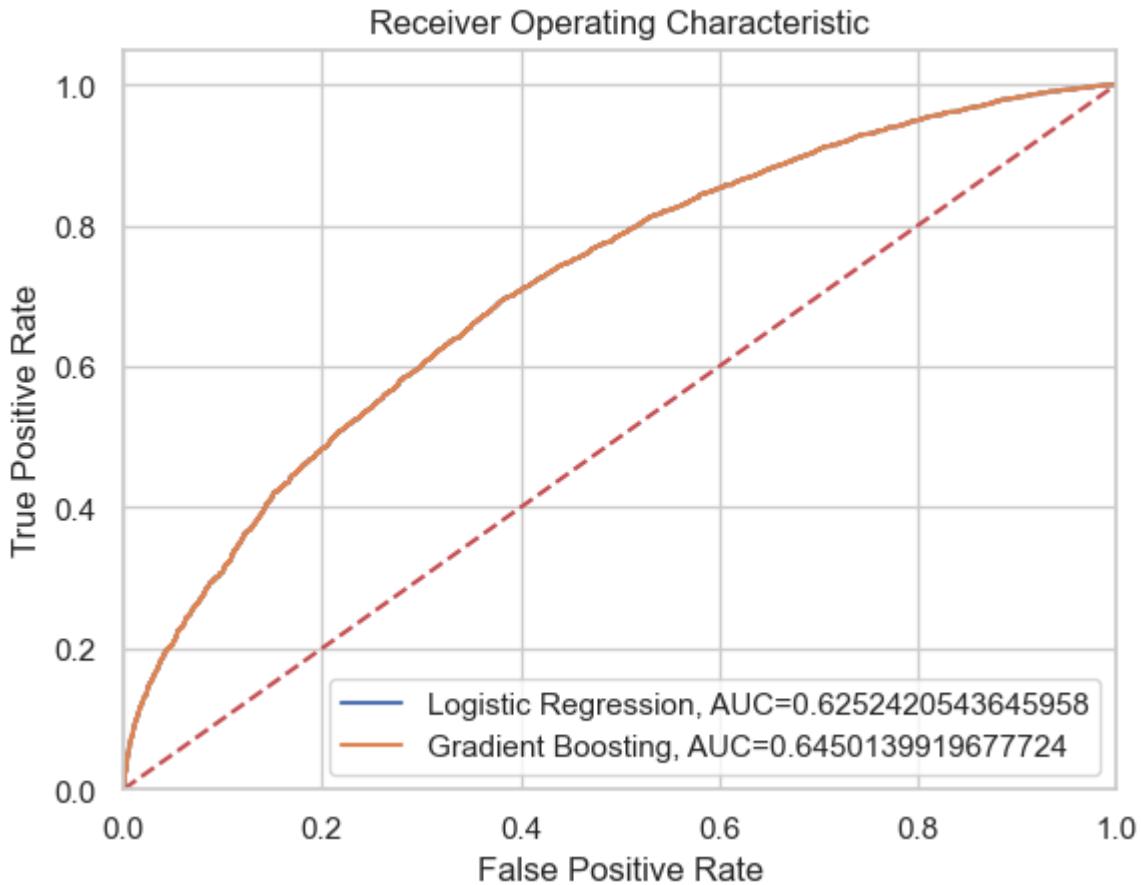
```
In [58]: xgbc_roc_auc = roc_auc_score(y_test, xgb2.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, xgb2.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Gradient Boosting (area = %0.2f)' % xgbc_roc_auc)
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
```

```
Out[58]: <Figure size 640x480 with 0 Axes>
Out[58]: [<matplotlib.lines.Line2D at 0x1af5898f370>]
Out[58]: [<matplotlib.lines.Line2D at 0x1af5898f760>]
Out[58]: (0.0, 1.0)
Out[58]: (0.0, 1.05)
Out[58]: Text(0.5, 0, 'False Positive Rate')
Out[58]: Text(0, 0.5, 'True Positive Rate')
Out[58]: Text(0.5, 1.0, 'Receiver Operating Characteristic')
Out[58]: <matplotlib.legend.Legend at 0x1af587266a0>
```



```
In [59]: #set up plotting area
plt.figure(0).clf()
plt.plot(fpr,tpr,label="Logistic Regression, AUC=" + str(logreg_roc_auc))
plt.plot(fpr,tpr,label="Gradient Boosting, AUC=" + str(xgbc_roc_auc))
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
```

```
Out[59]: [<matplotlib.lines.Line2D at 0x1af01ff2d30>]
Out[59]: [<matplotlib.lines.Line2D at 0x1af01ff2eb0>]
Out[59]: [<matplotlib.lines.Line2D at 0x1af02004190>]
Out[59]: (0.0, 1.0)
Out[59]: (0.0, 1.05)
Out[59]: Text(0.5, 0, 'False Positive Rate')
Out[59]: Text(0, 0.5, 'True Positive Rate')
Out[59]: Text(0.5, 1.0, 'Receiver Operating Characteristic')
Out[59]: <matplotlib.legend.Legend at 0x1af01ff2c70>
```

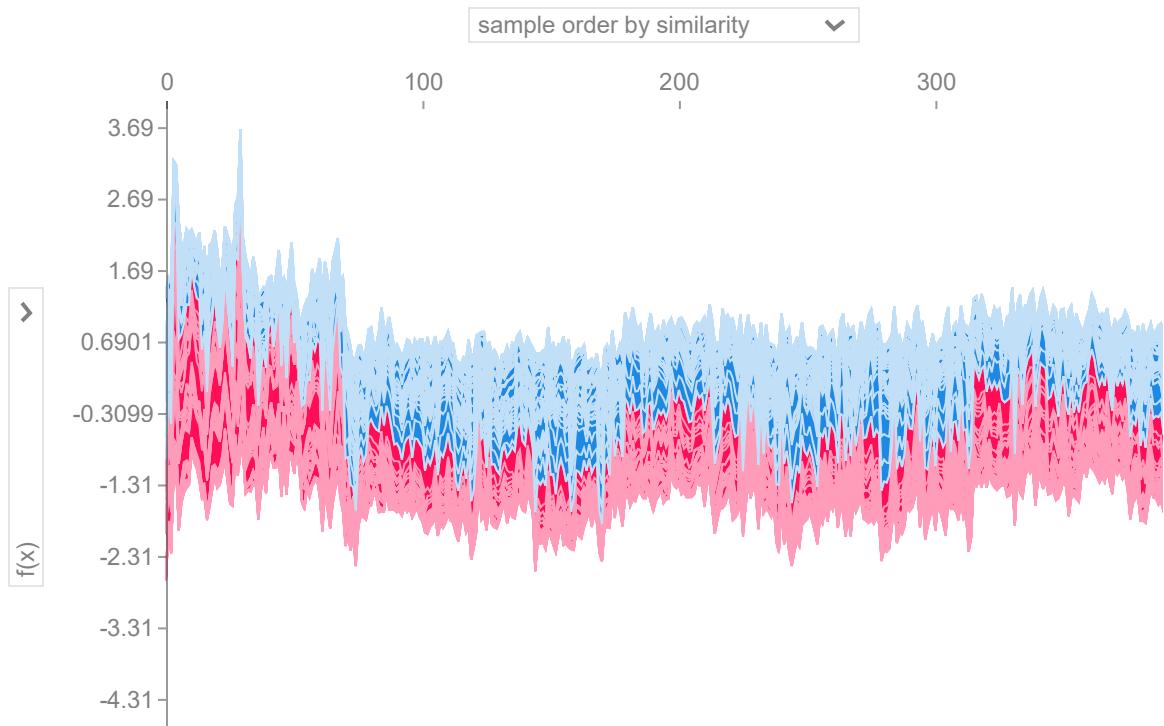


## Feature Importance

```
In [60]: explainer = shap.TreeExplainer(xgb2)
shap_values = explainer.shap_values(X)
```

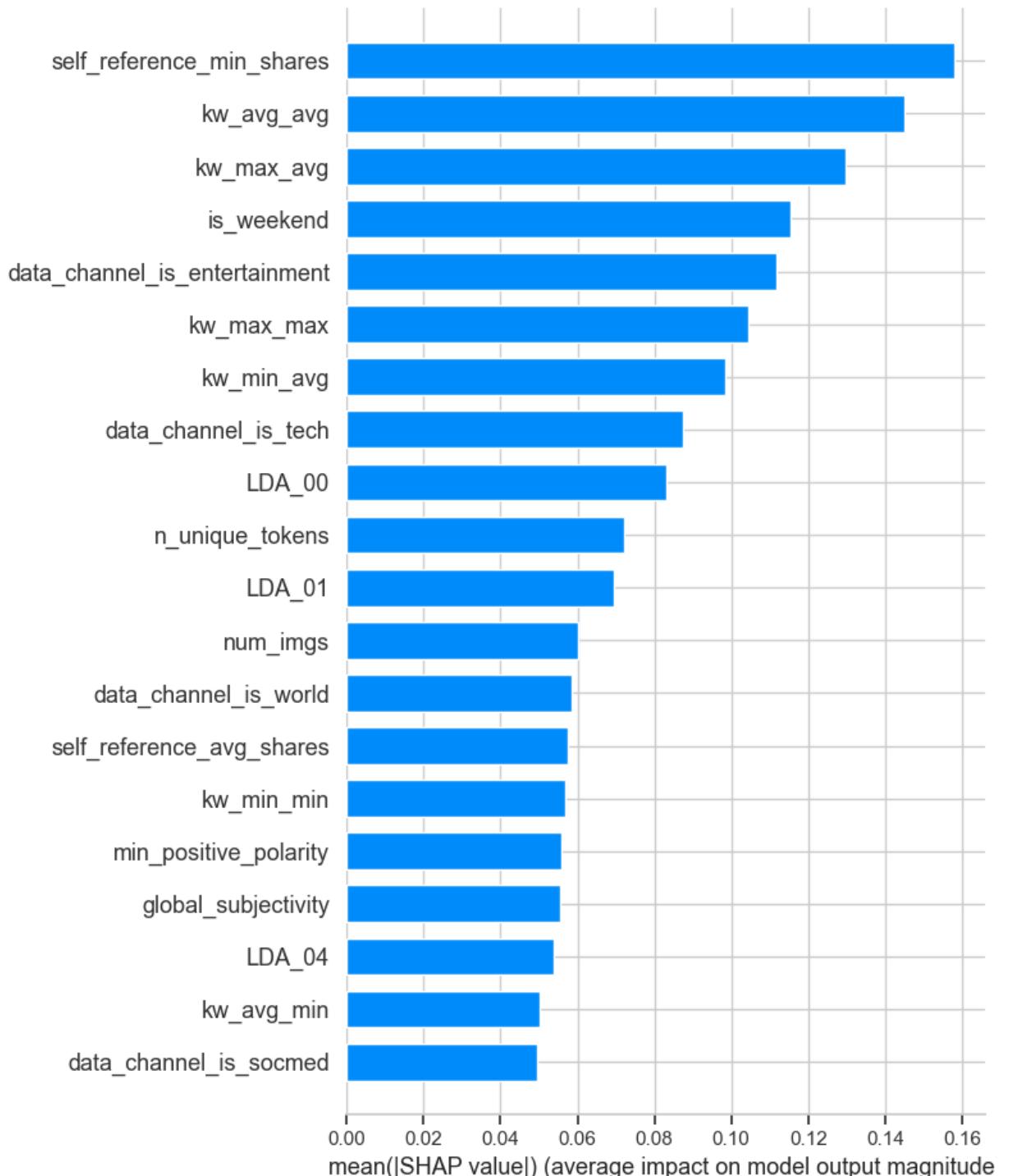
```
In [61]: shap.force_plot(explainer.expected_value, shap_values[:1000,:], X_train.iloc[:1000,
```

Out[61]:

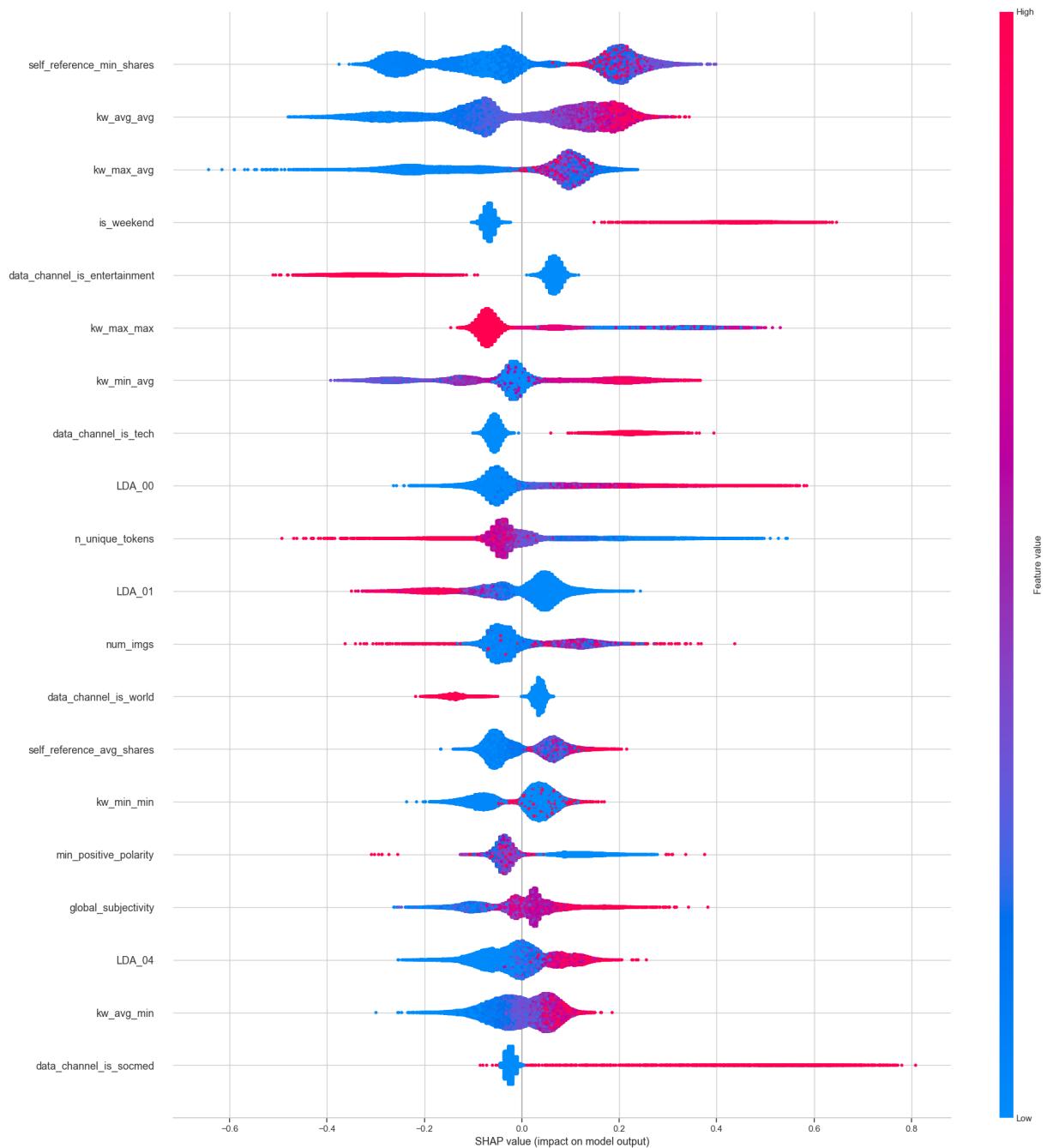


In [62]: `plt.figure(figsize = (20,20))  
shap.summary_plot(shap_values, X_train, plot_type="bar",)`

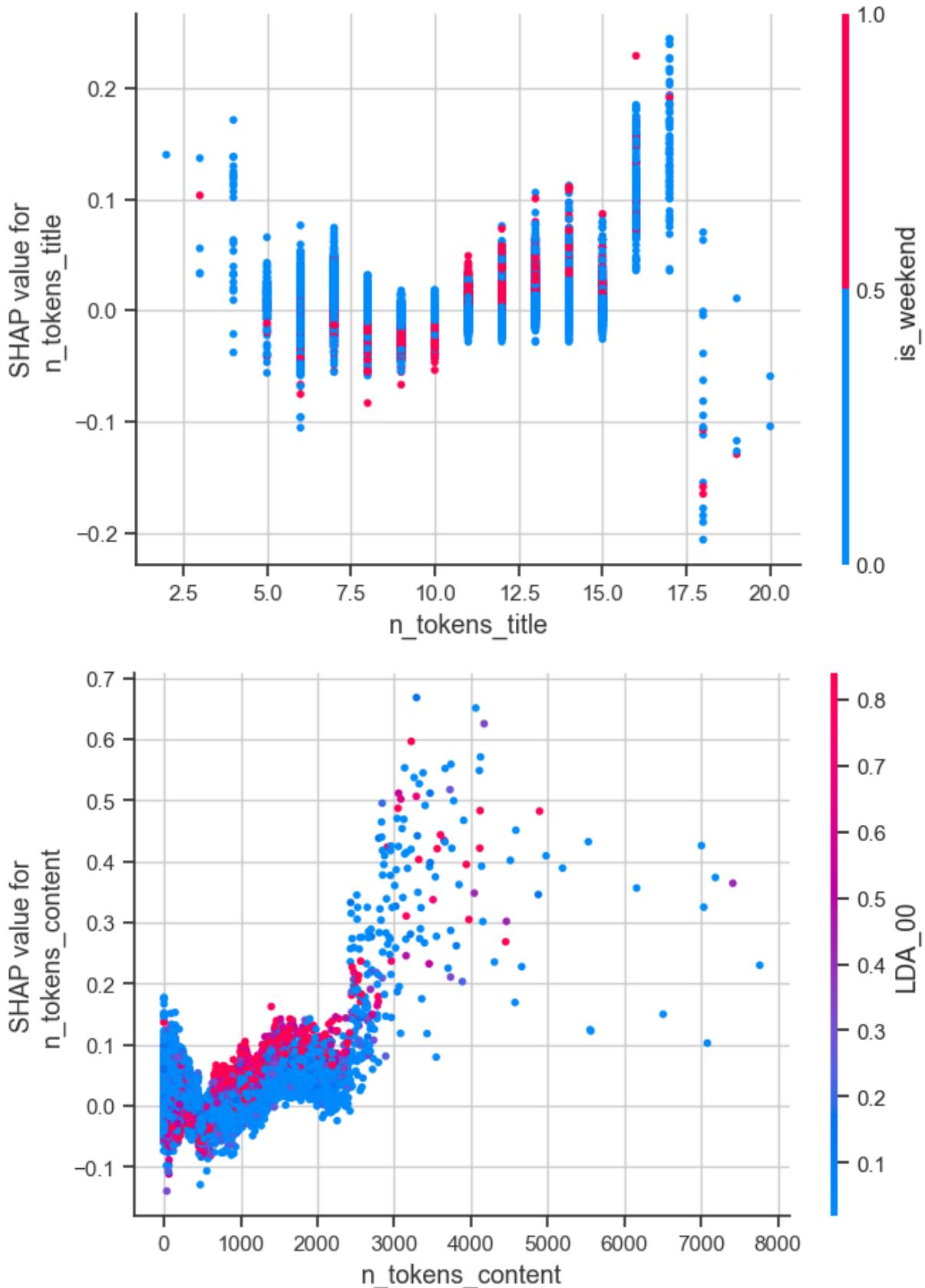
Out[62]: <Figure size 2000x2000 with 0 Axes>

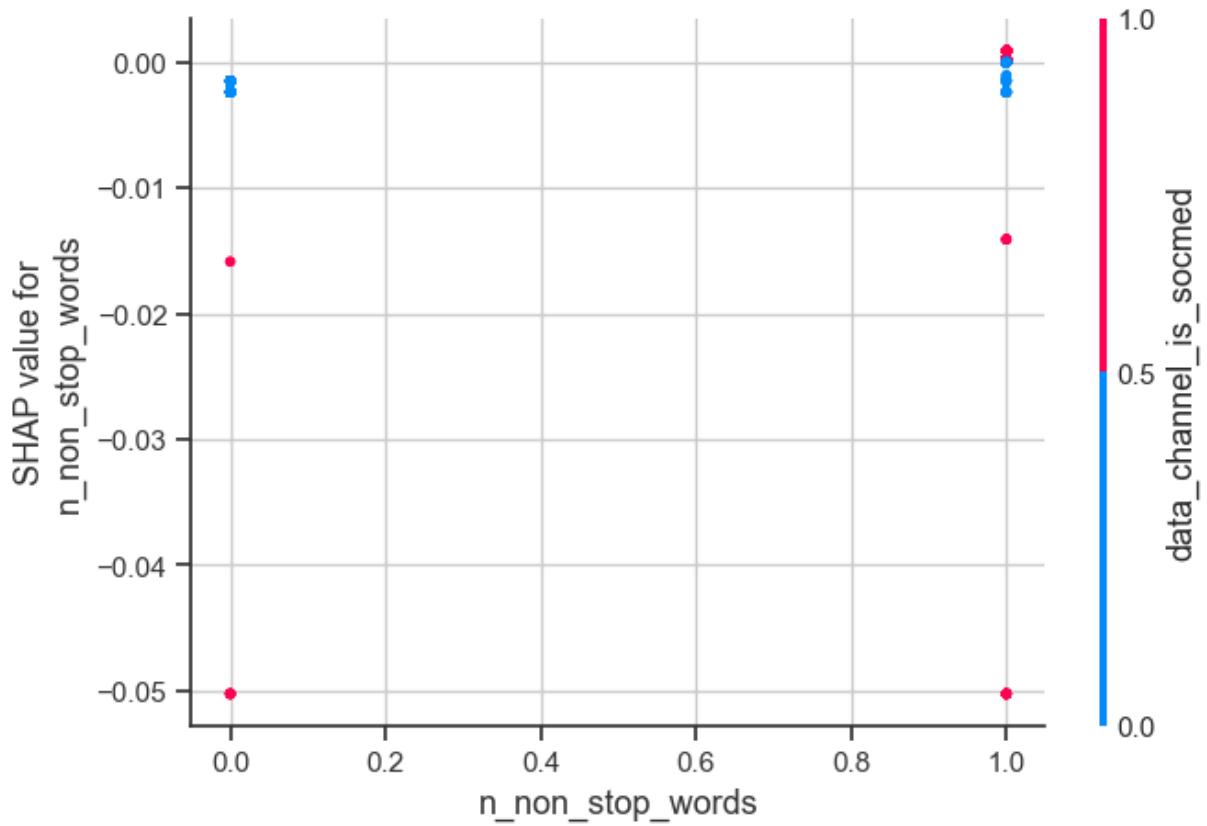
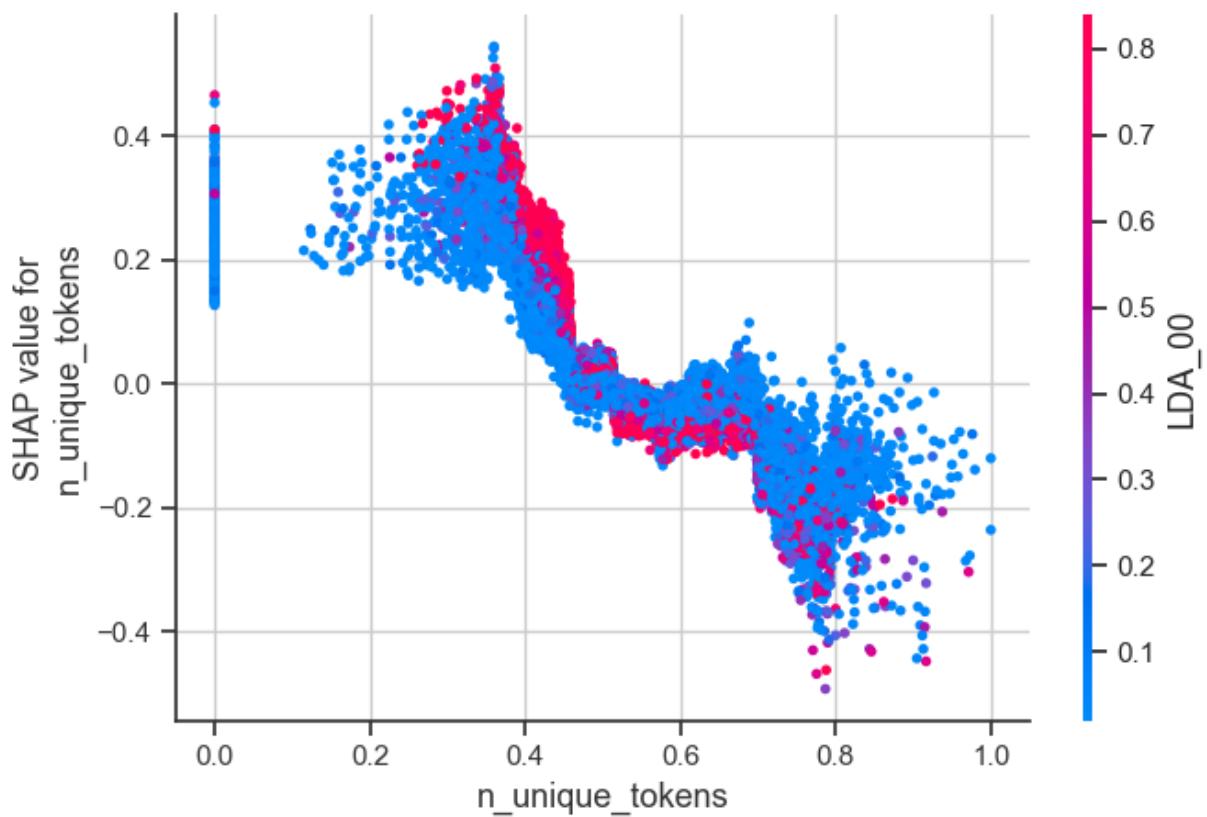


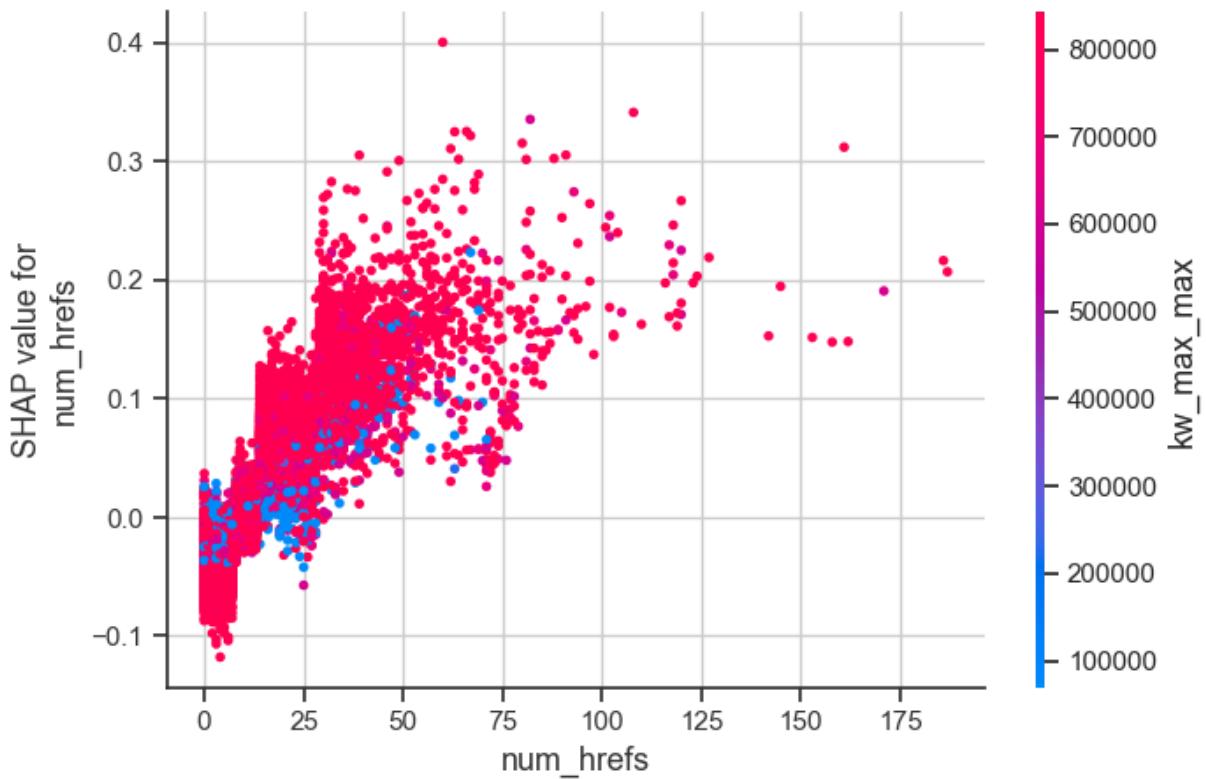
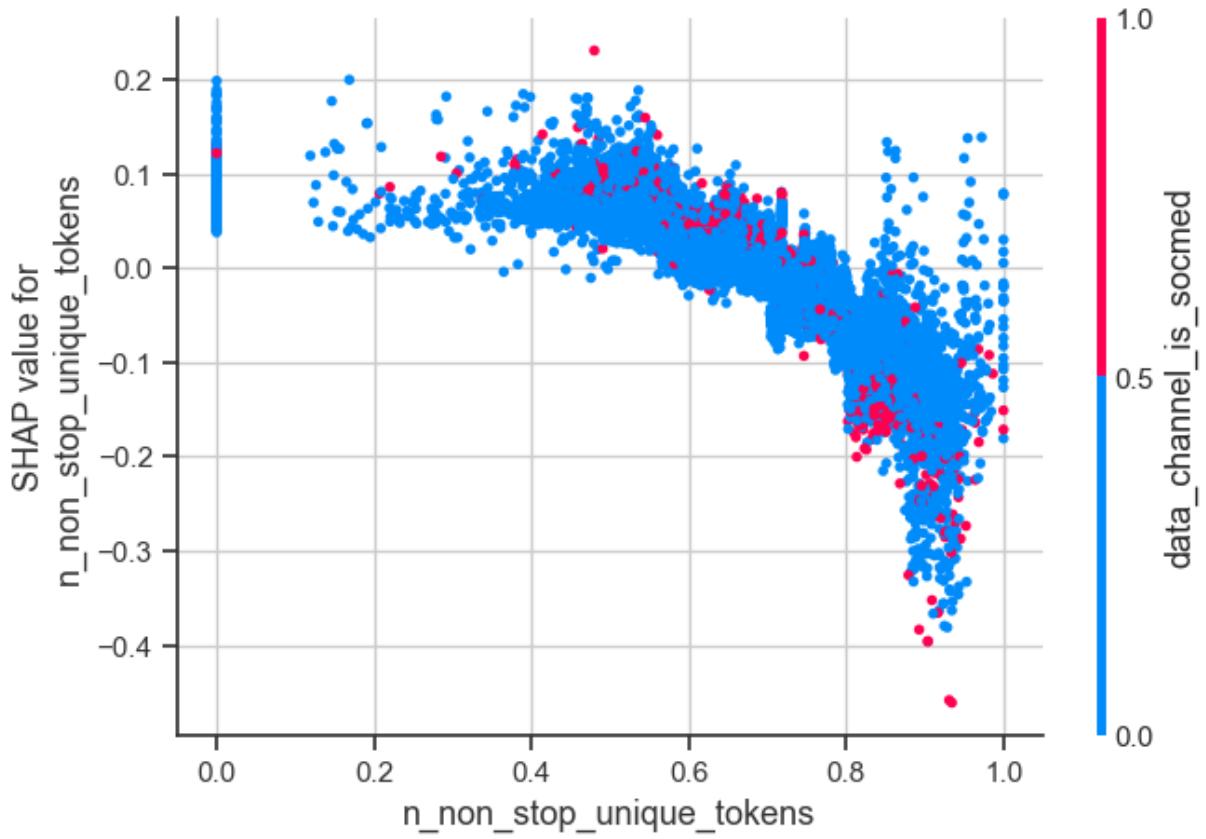
```
In [63]: shap.summary_plot(shap_values, X, plot_size=(20,20))
```

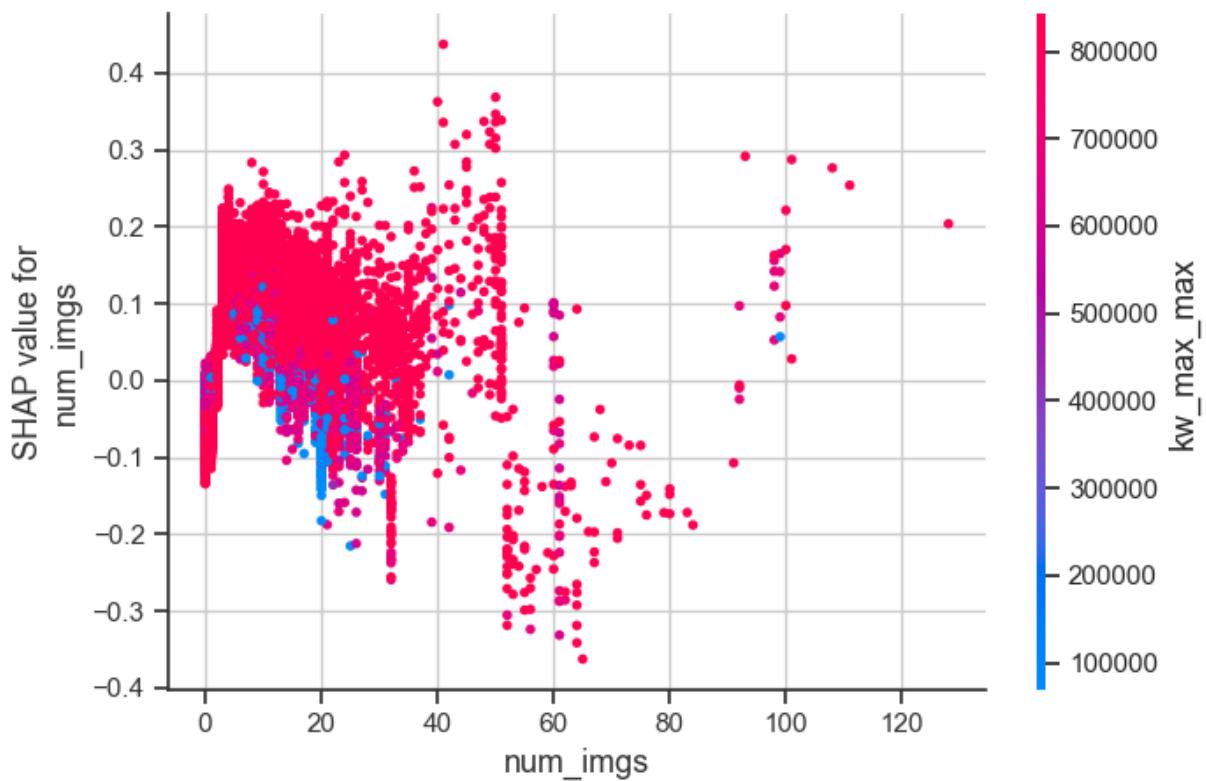
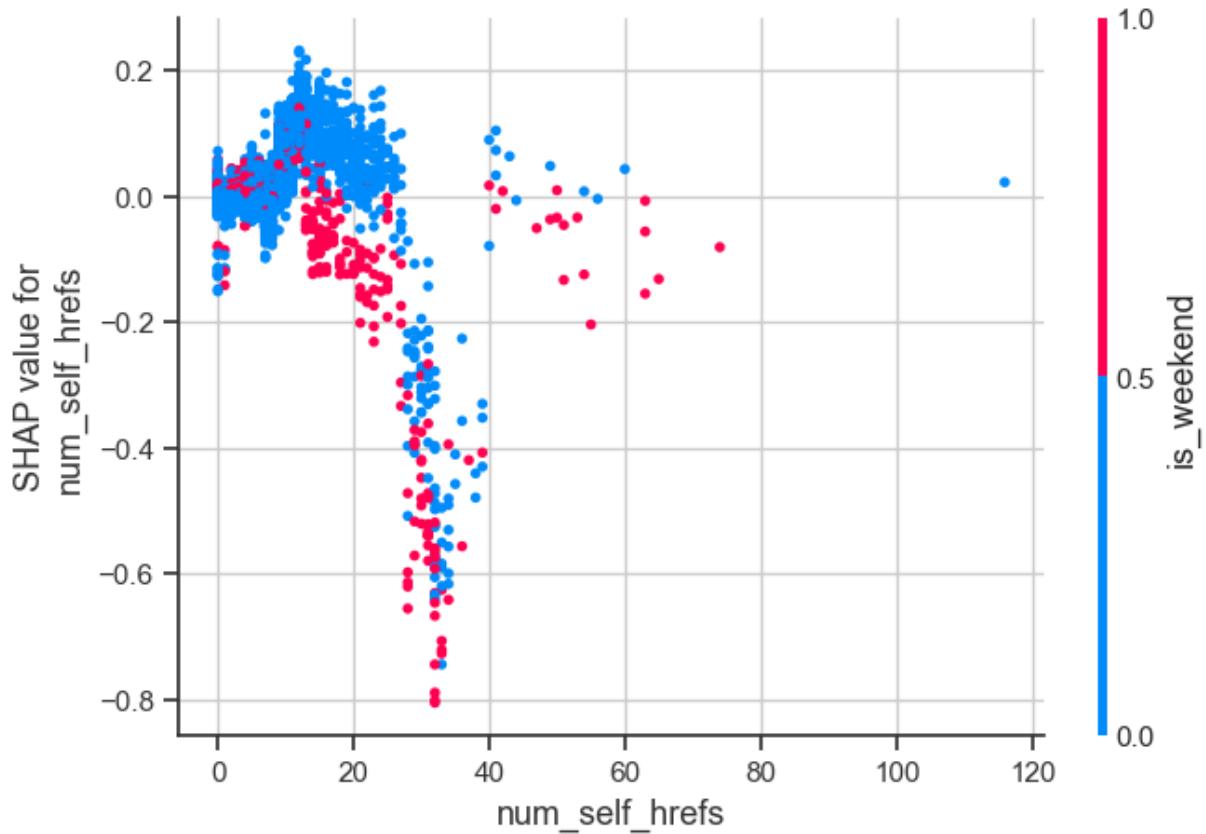


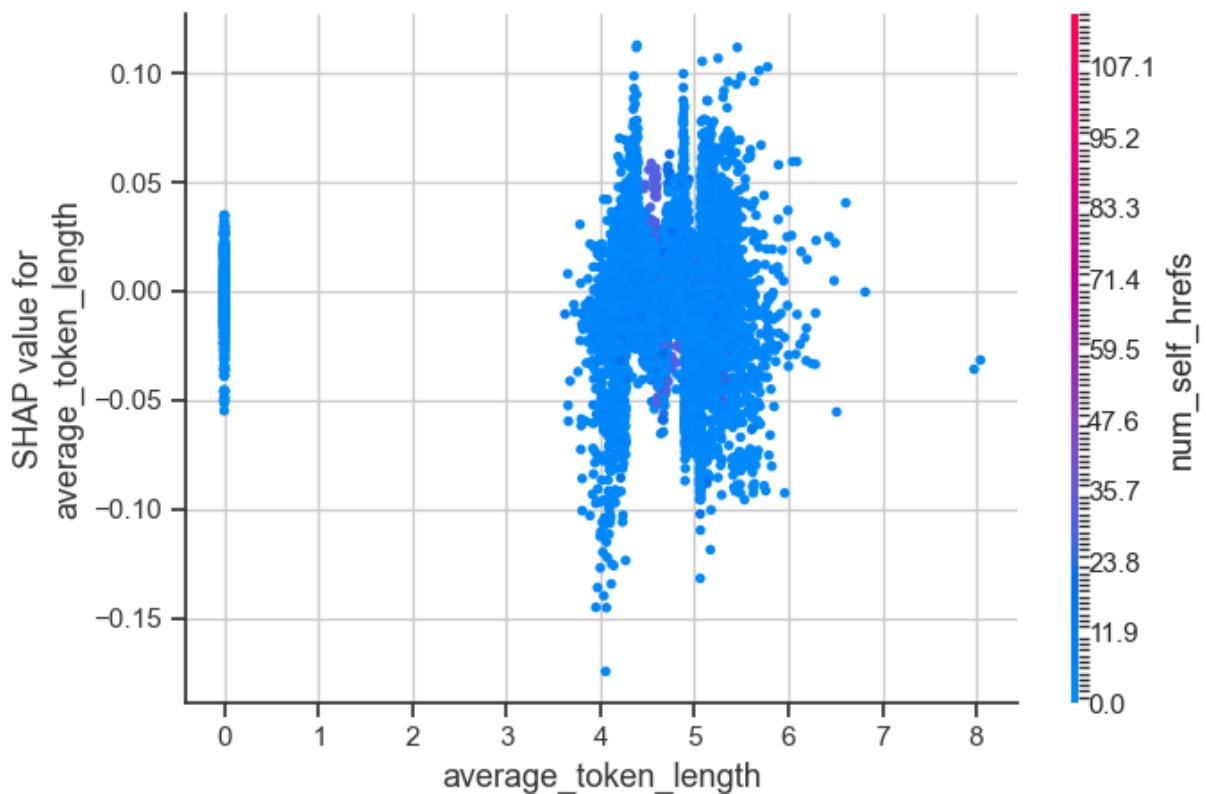
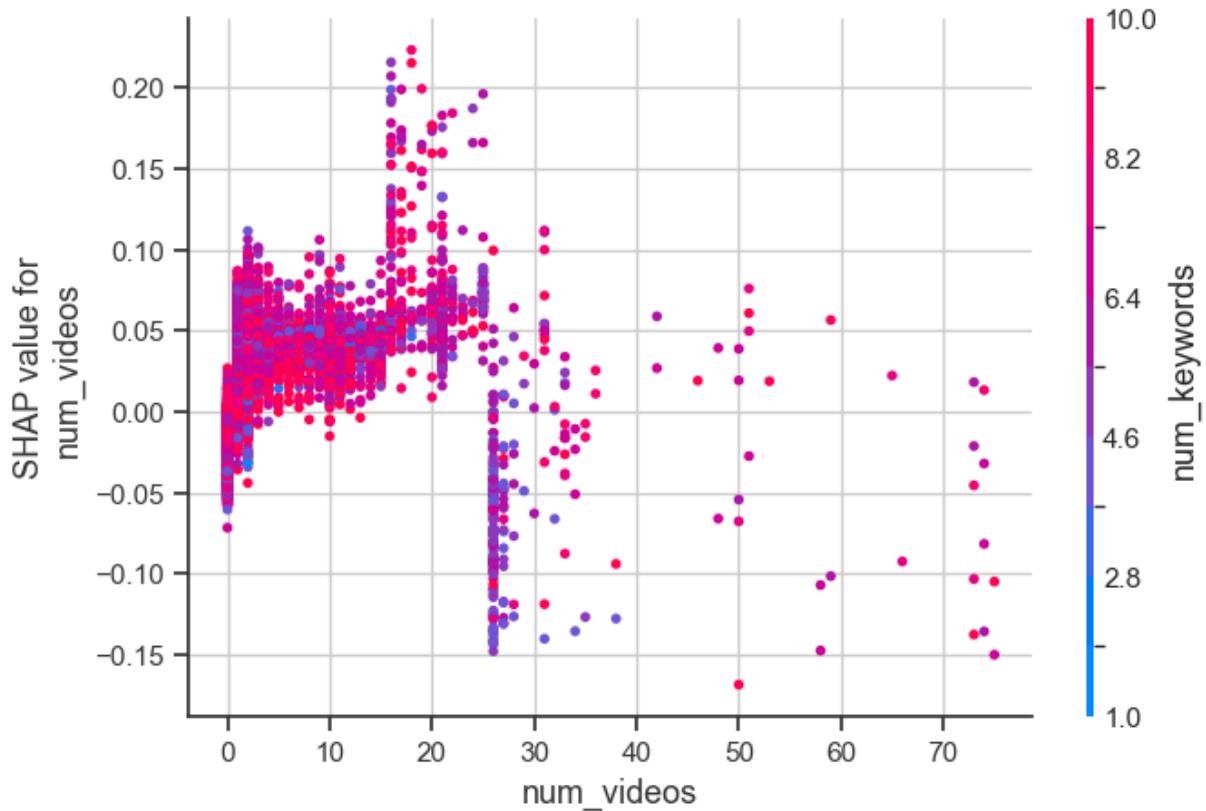
```
In [64]: for col in X_train.columns:
    shap.dependence_plot(col, shap_values, X)
```

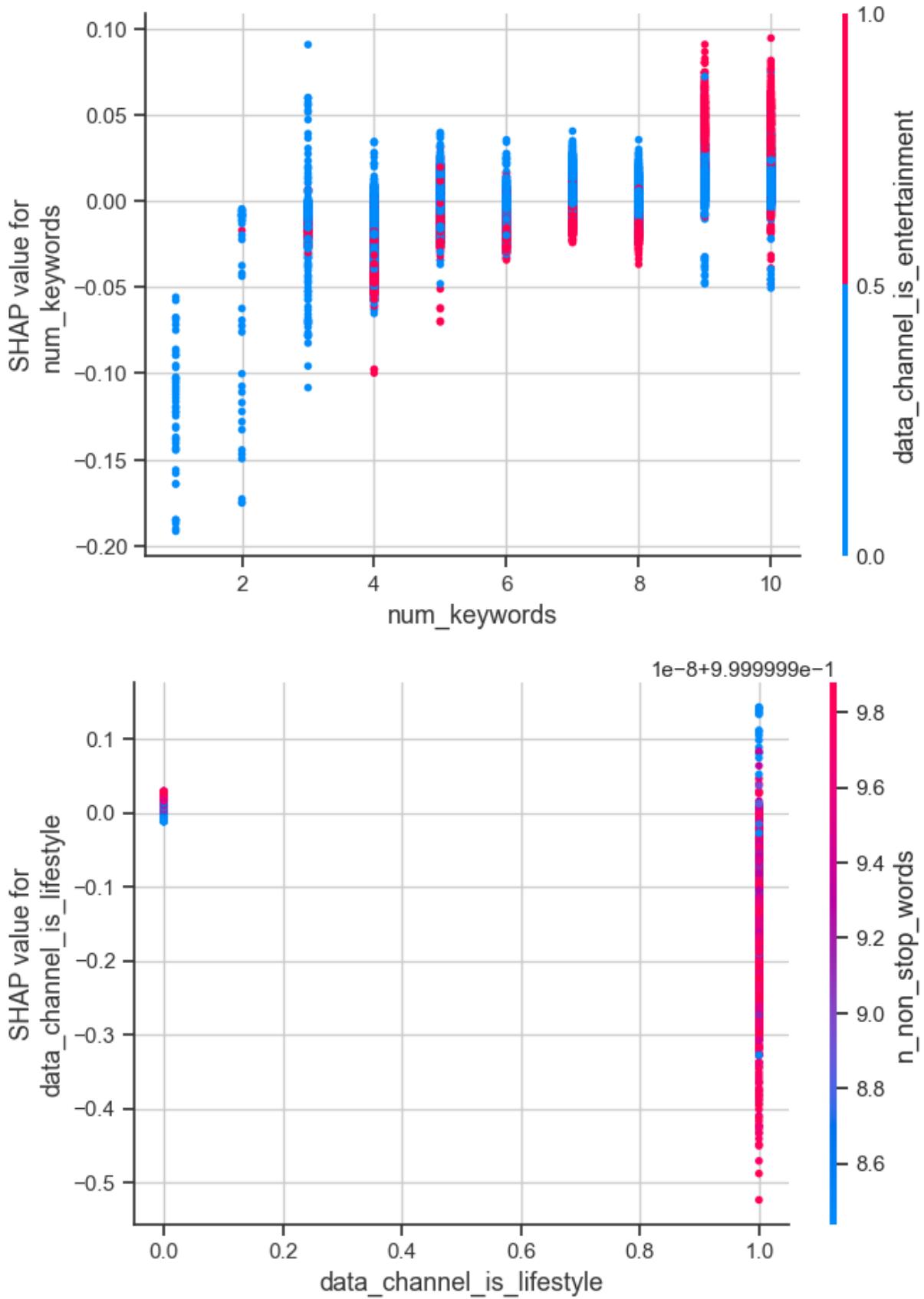


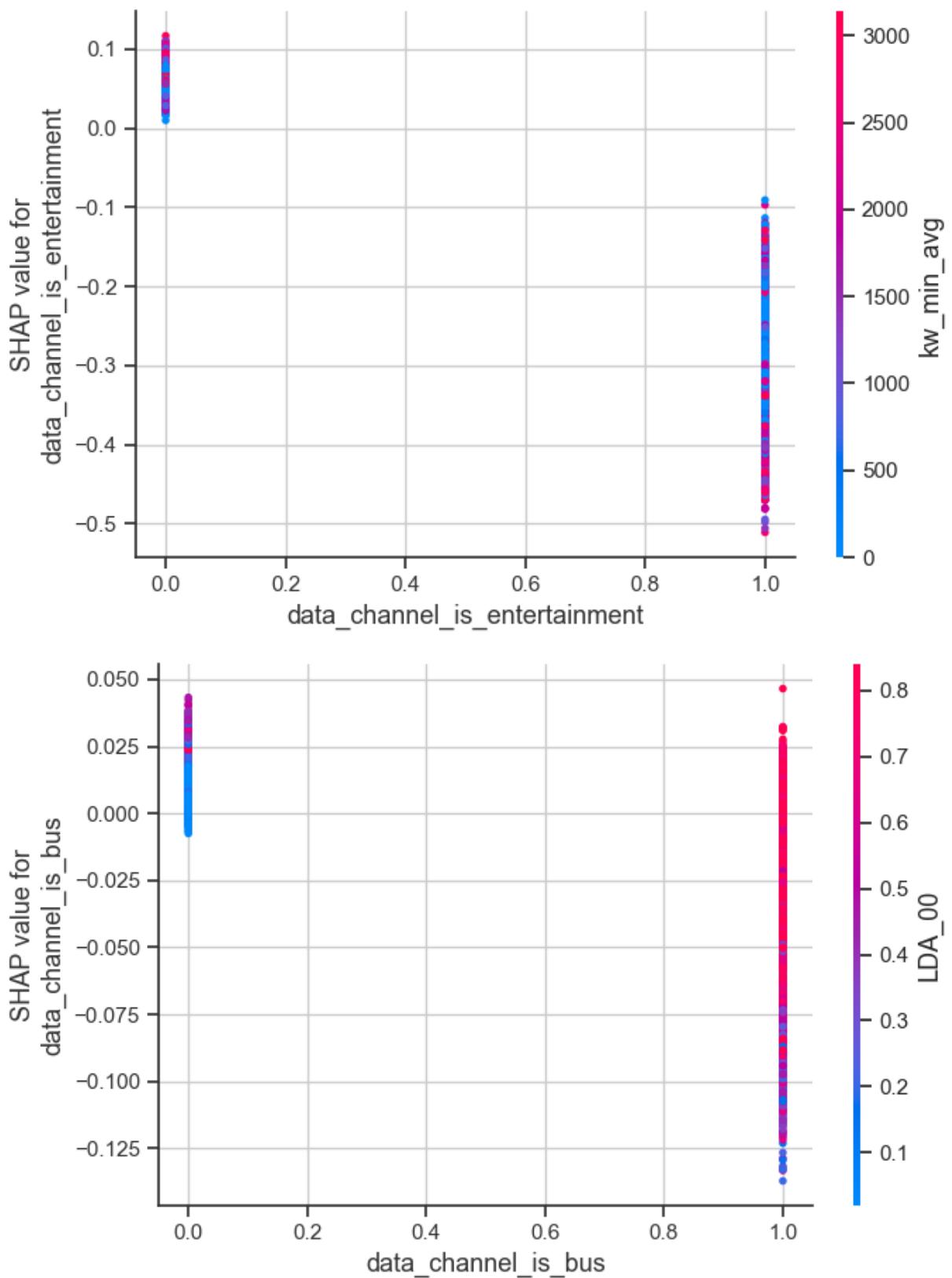


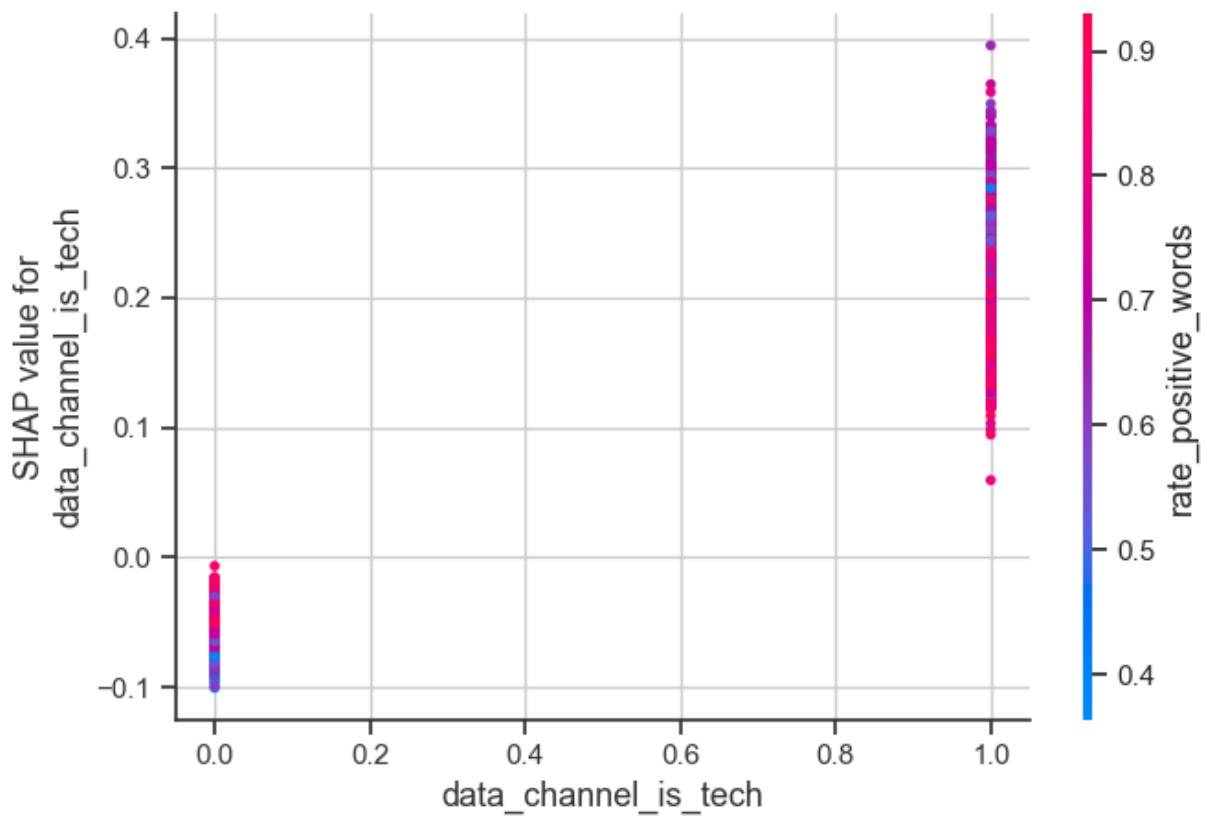
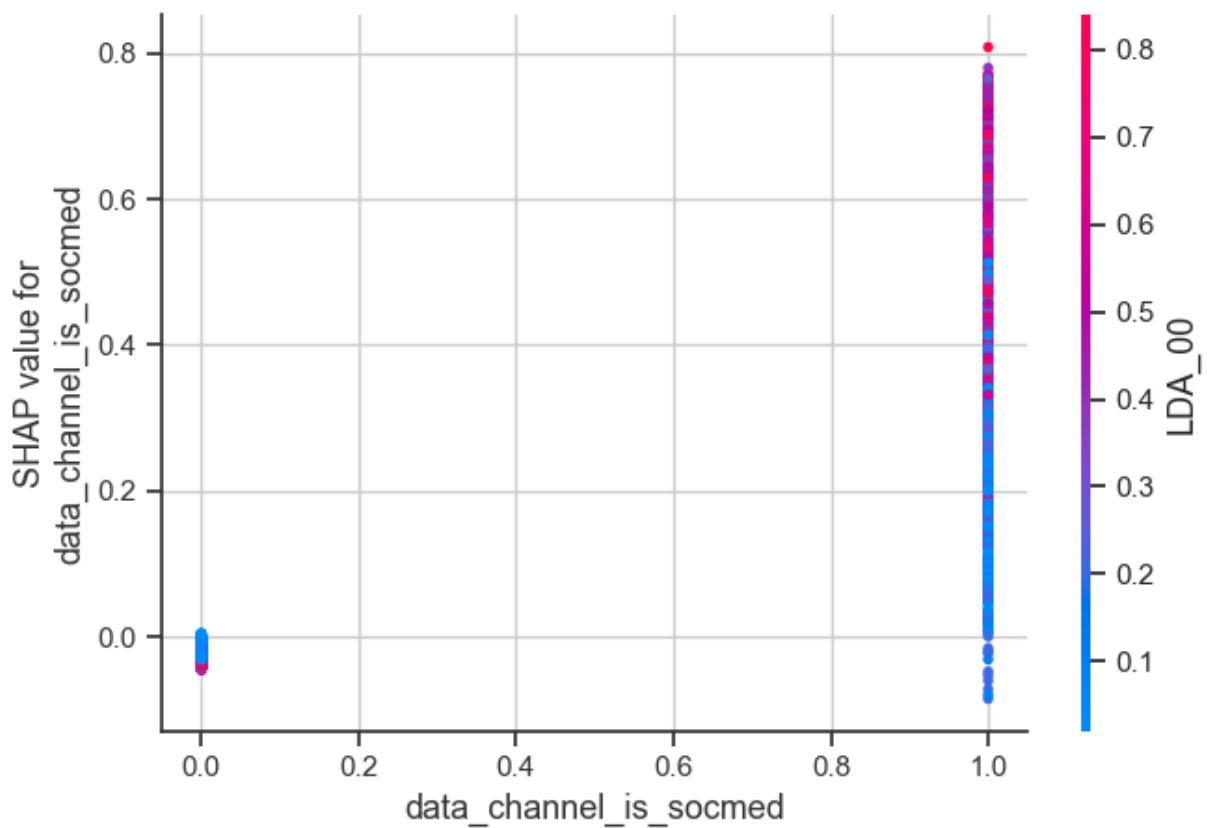


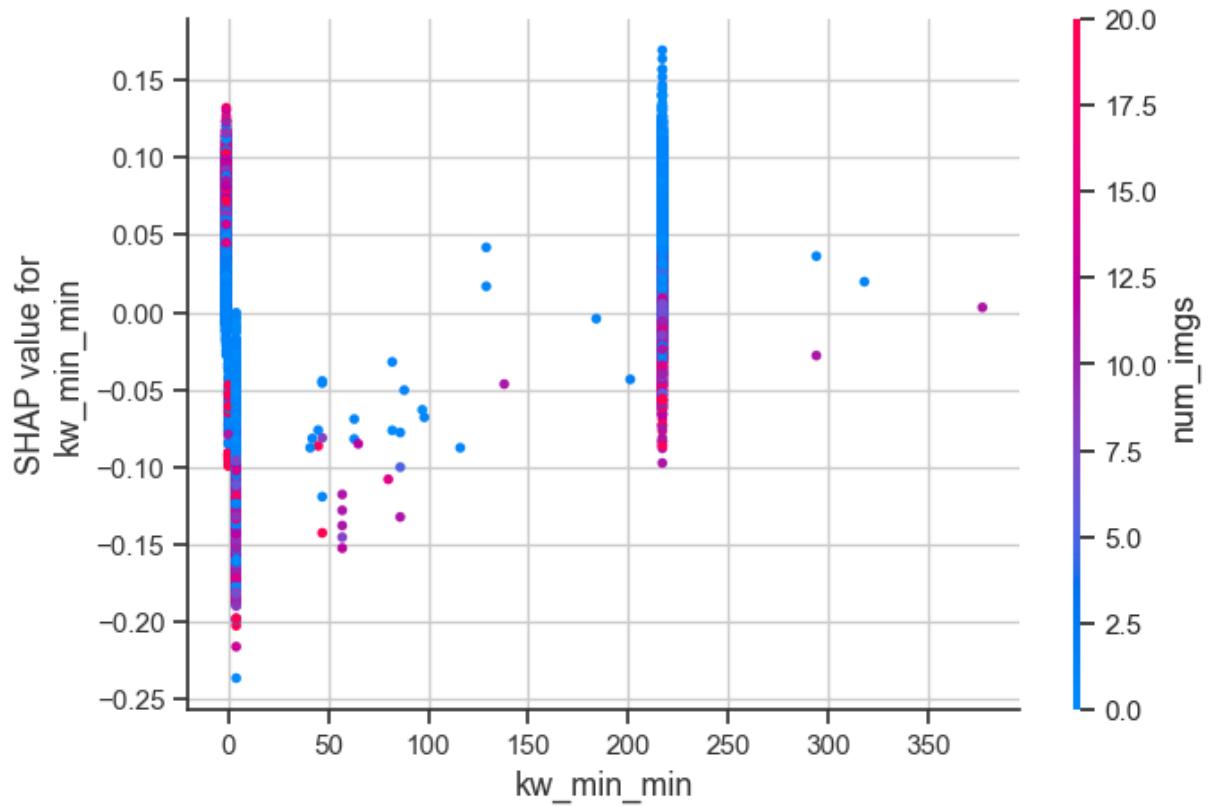
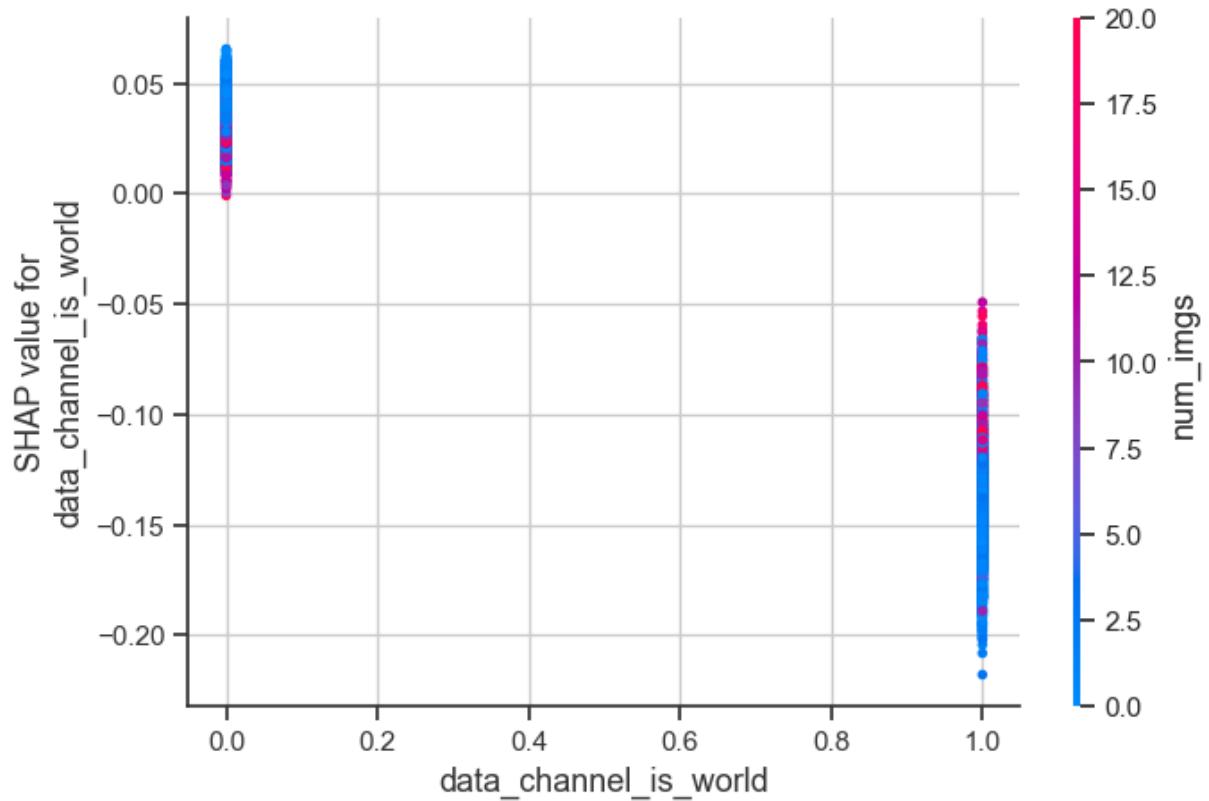


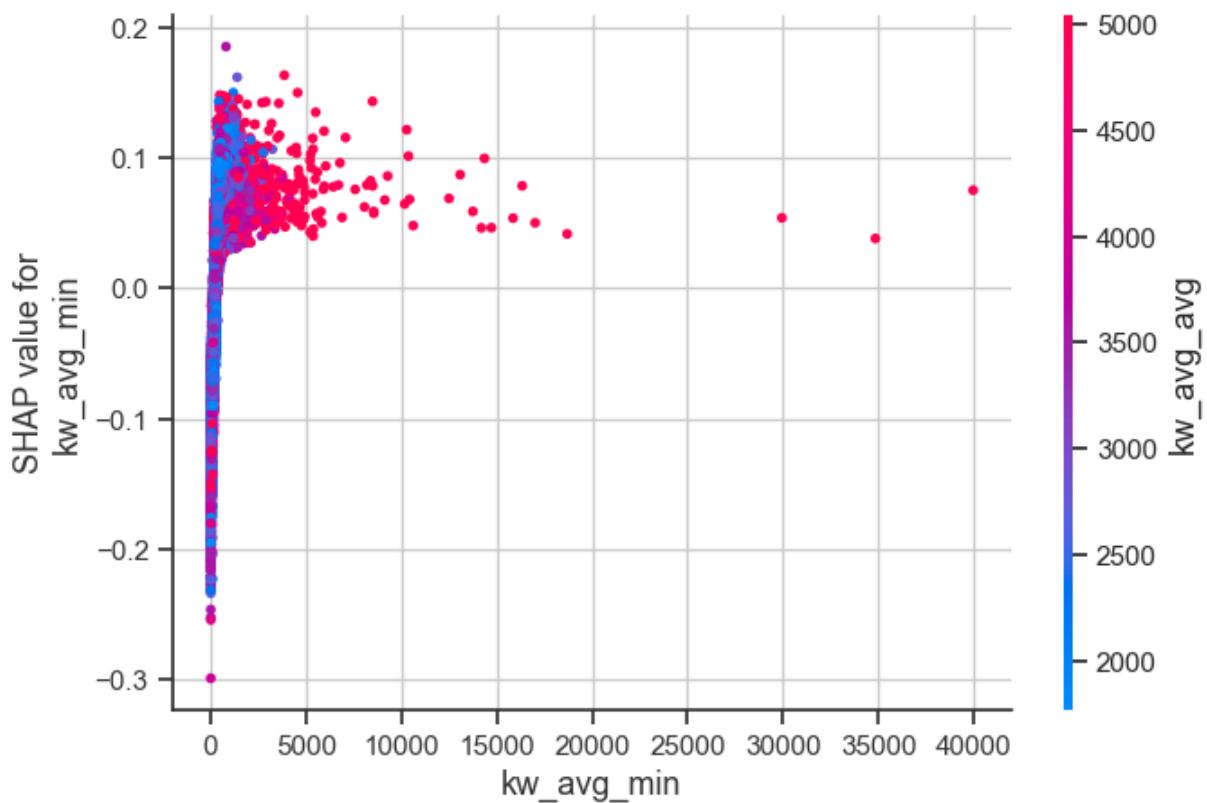
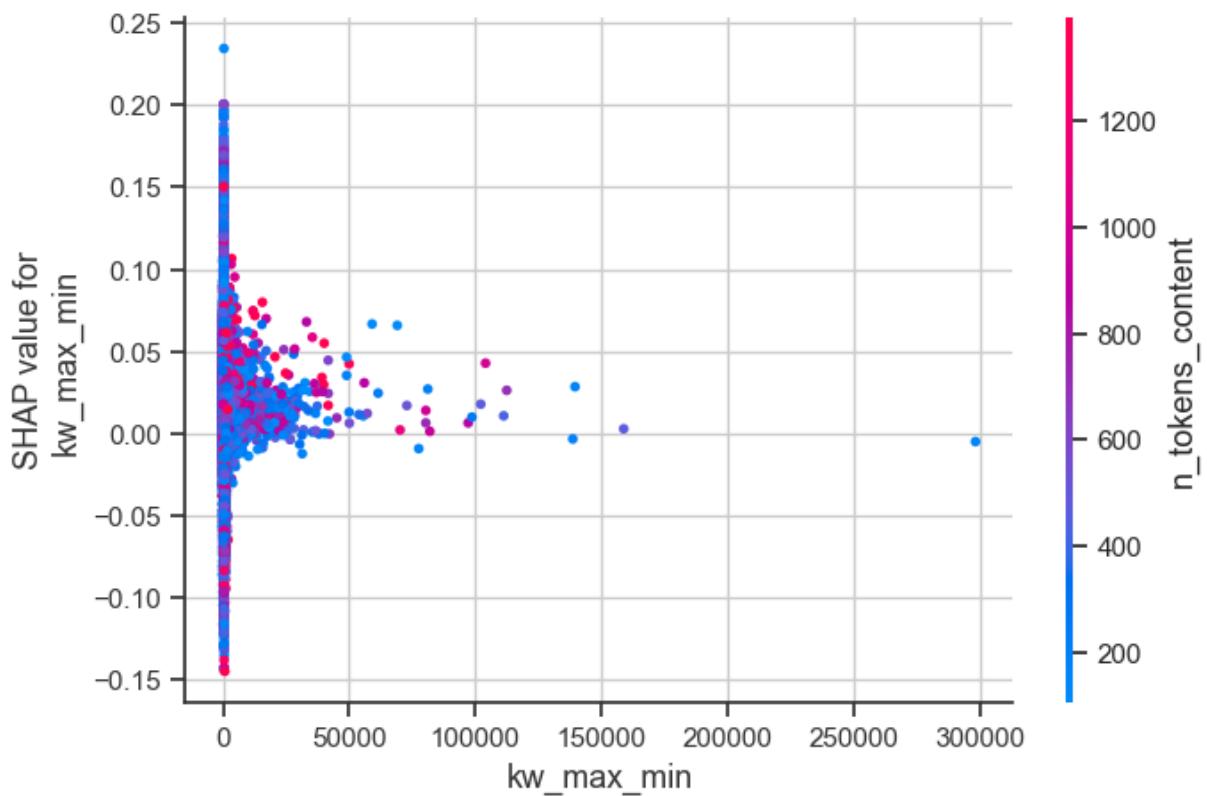


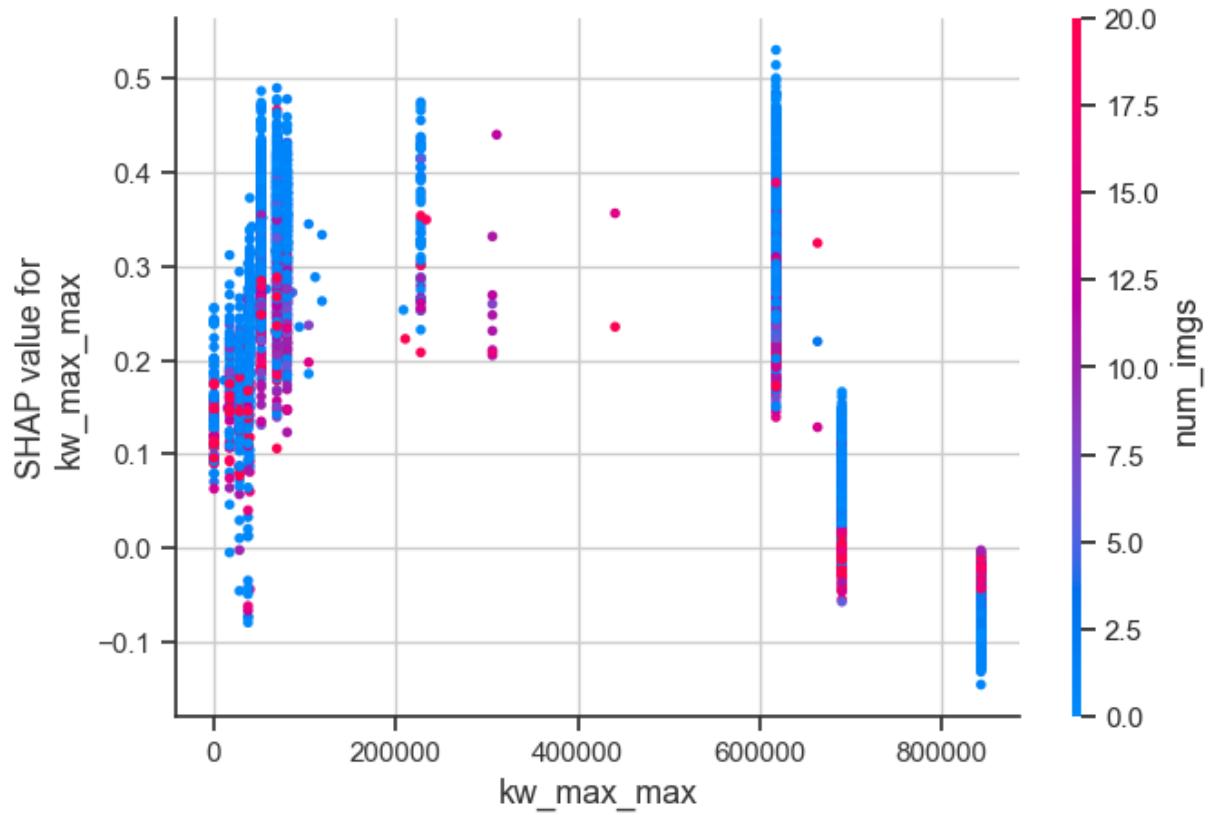
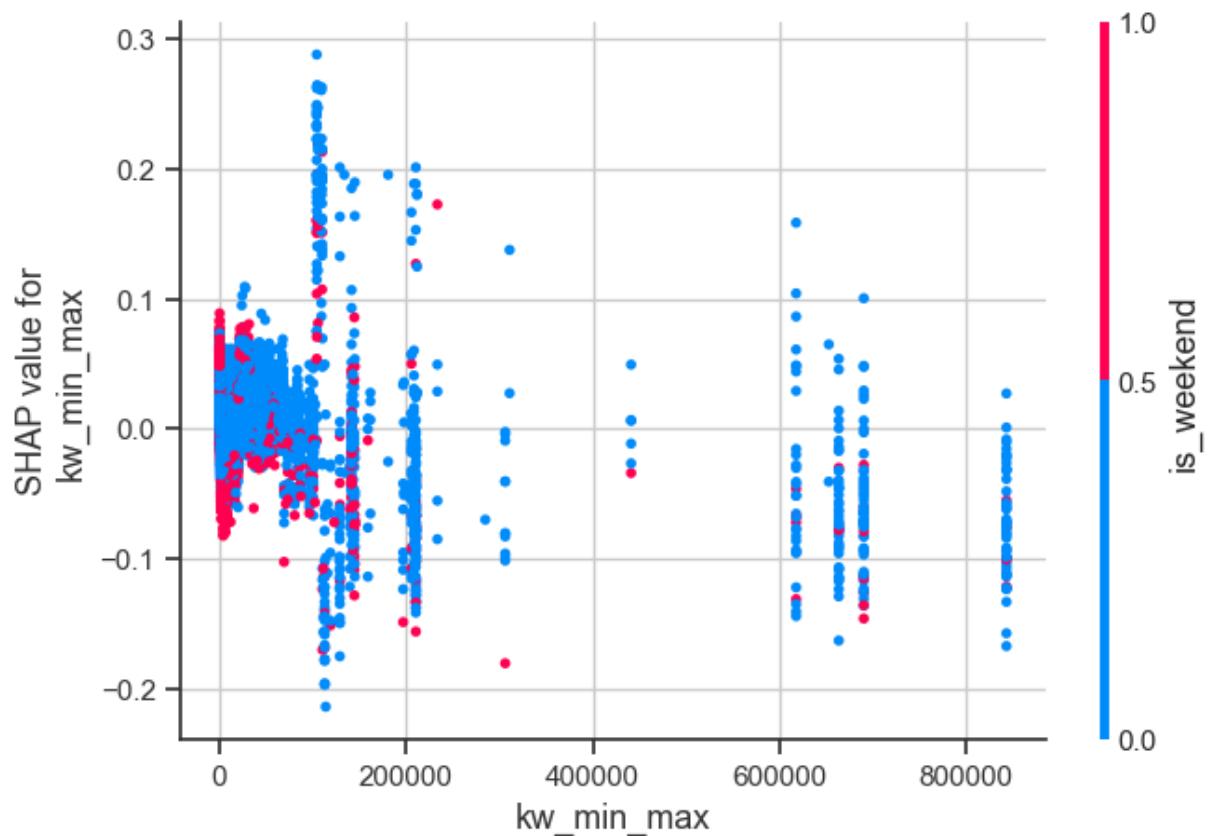


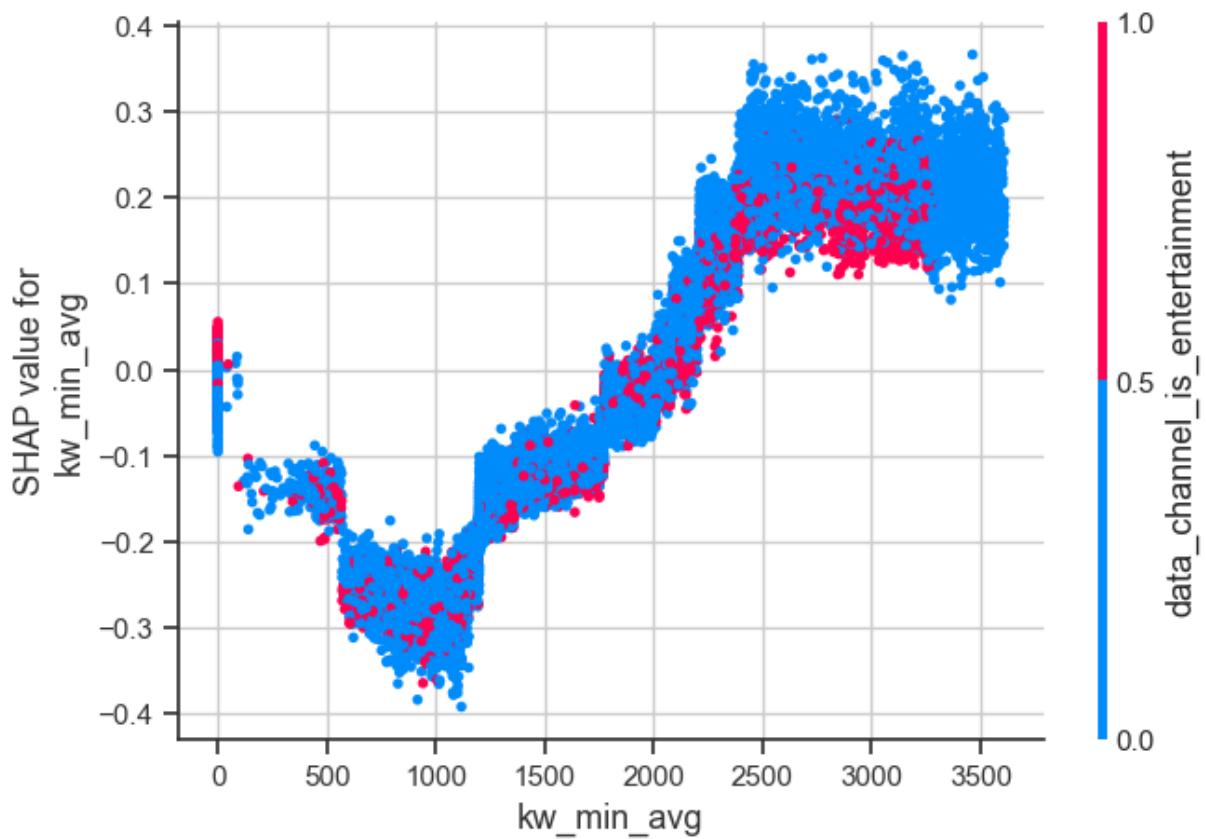
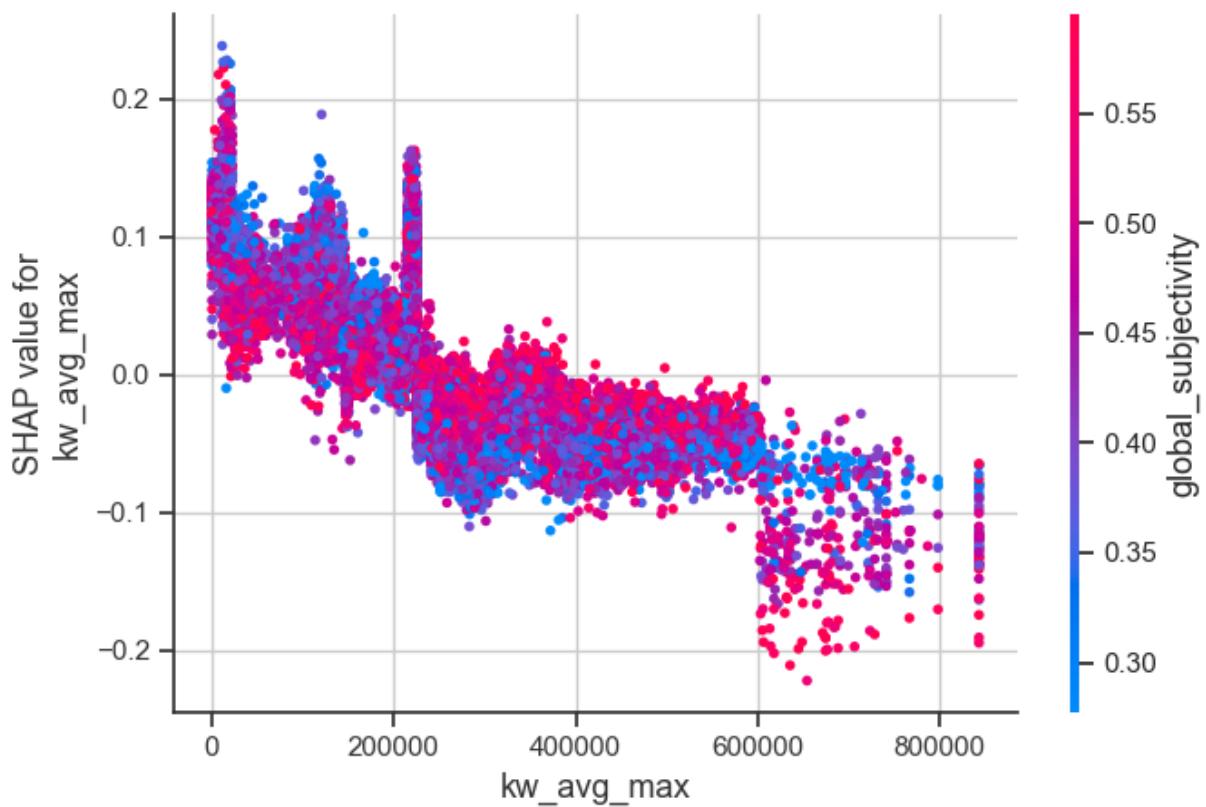


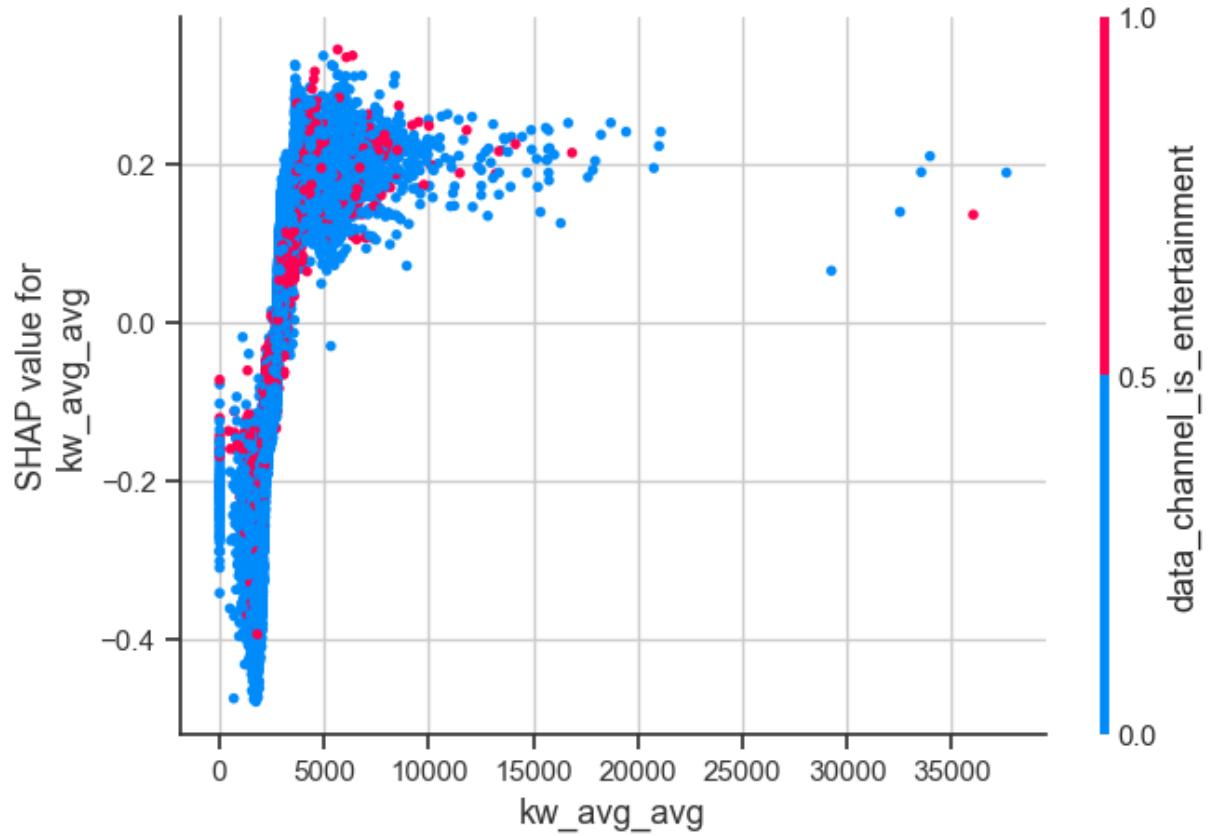
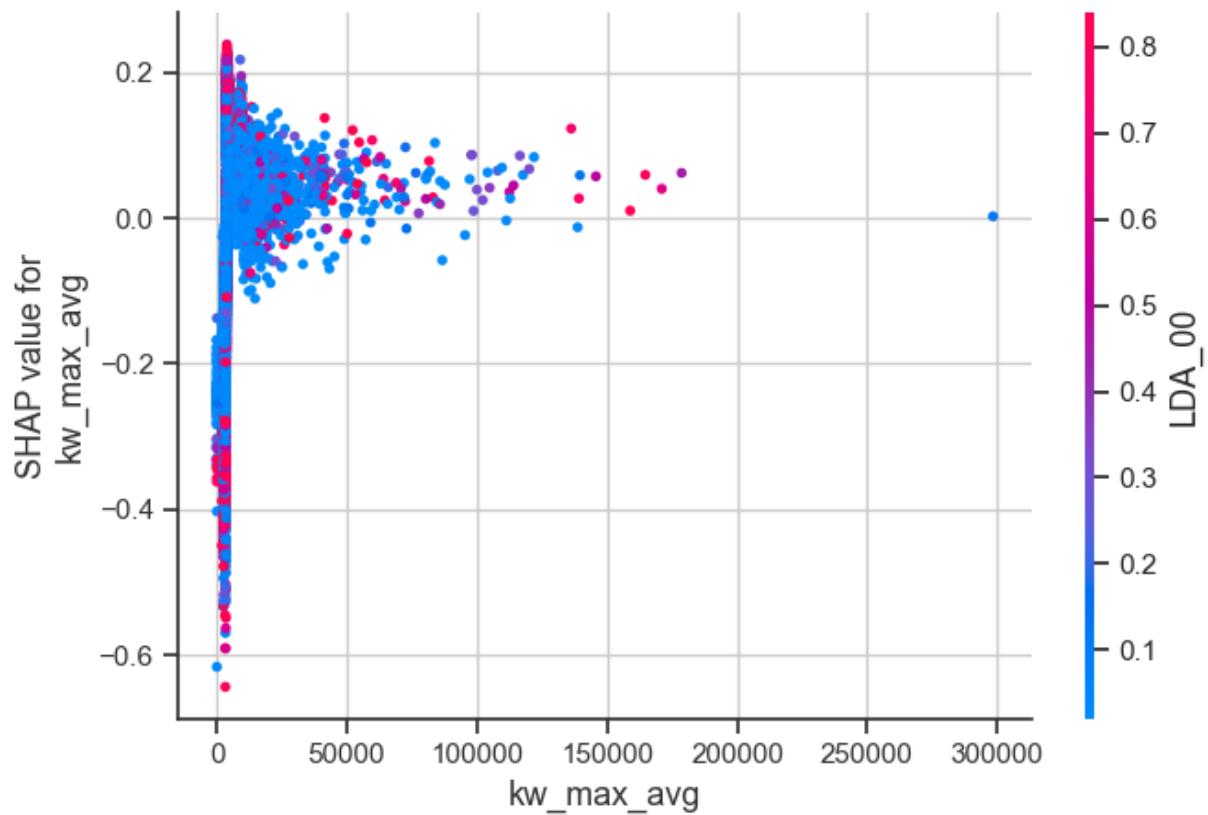


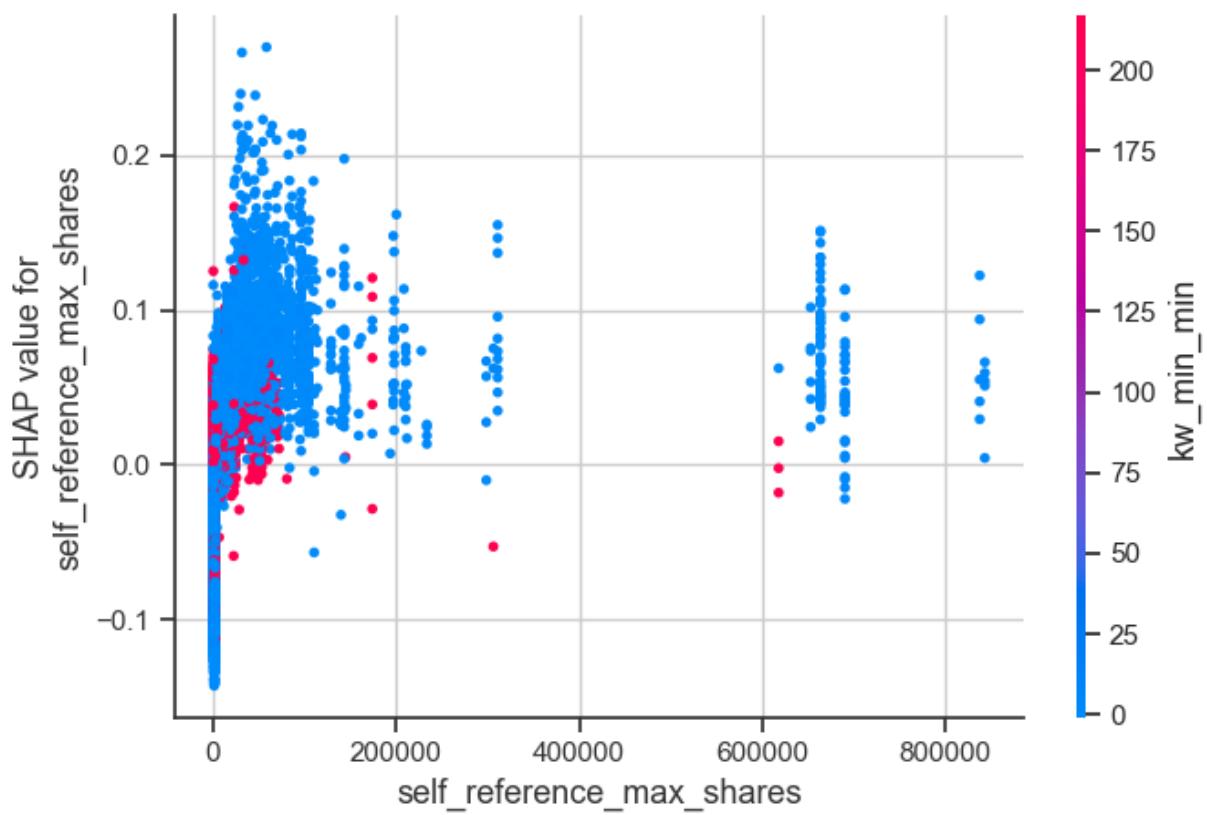
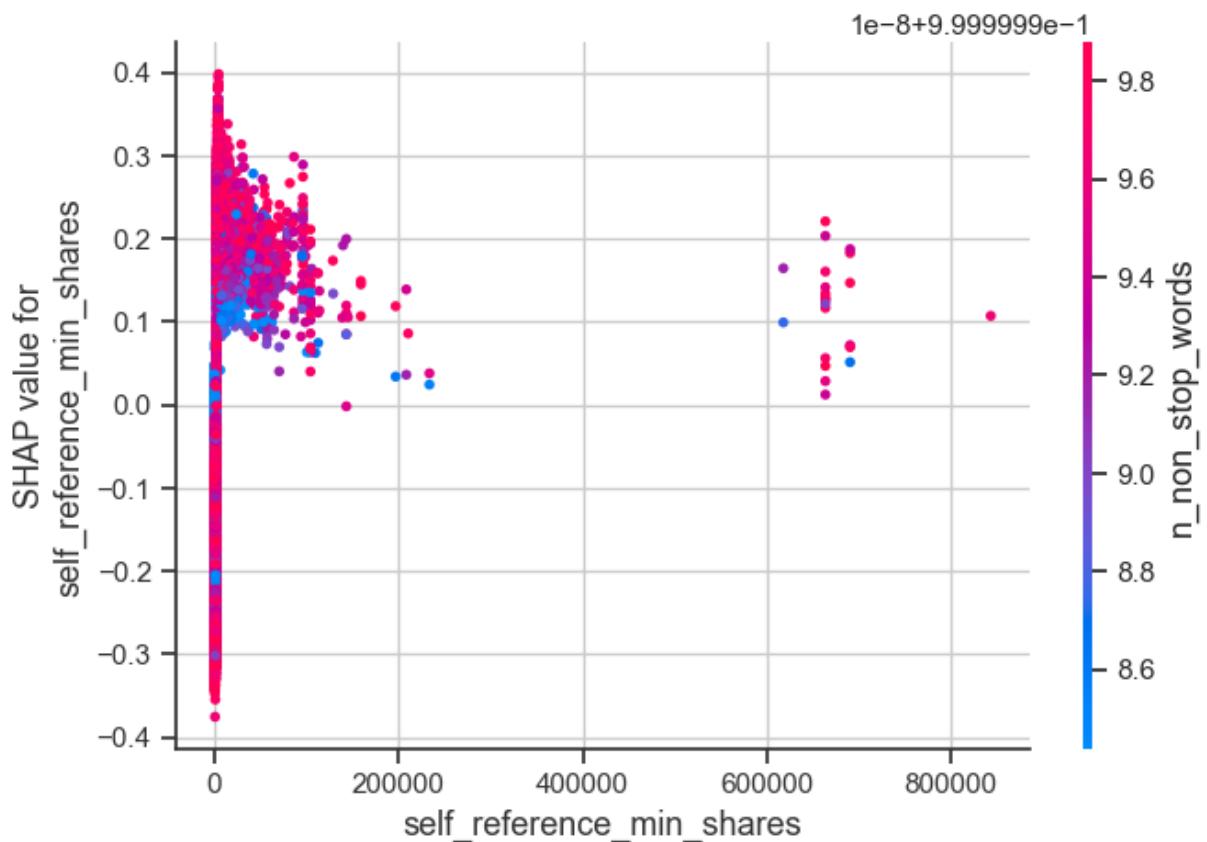


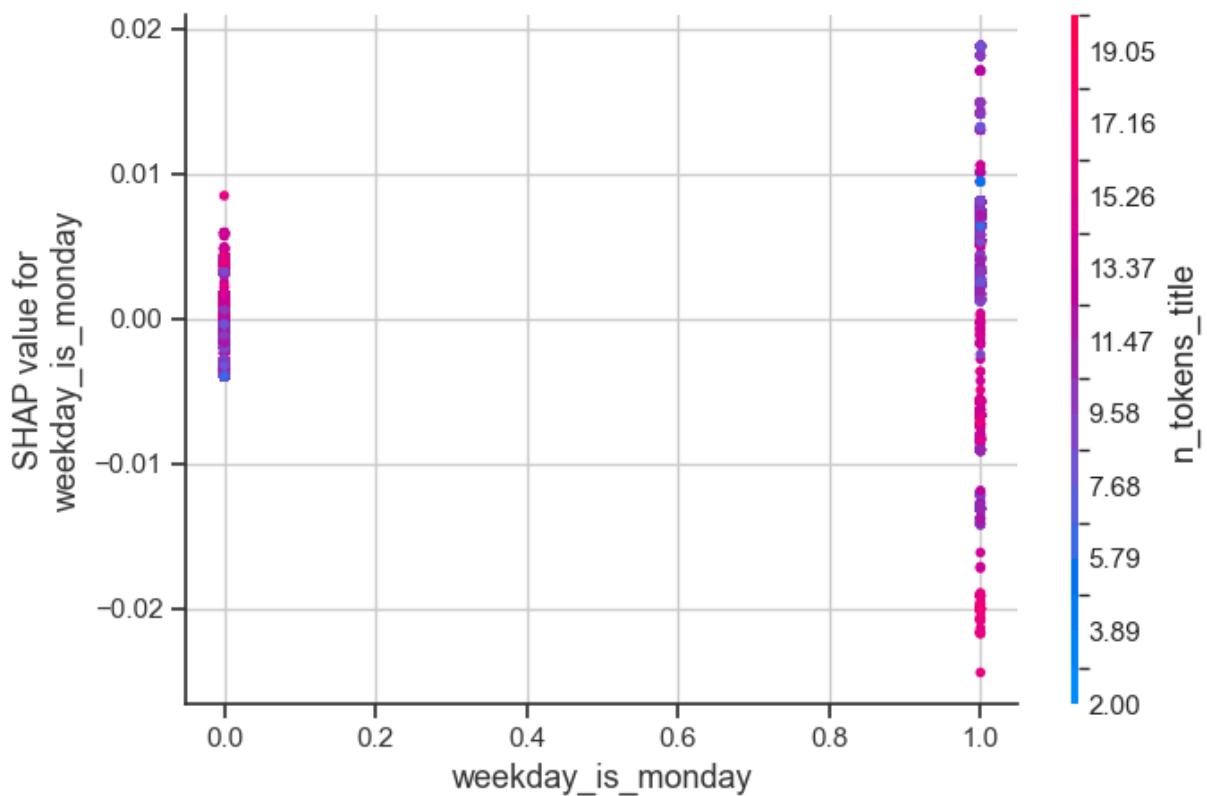
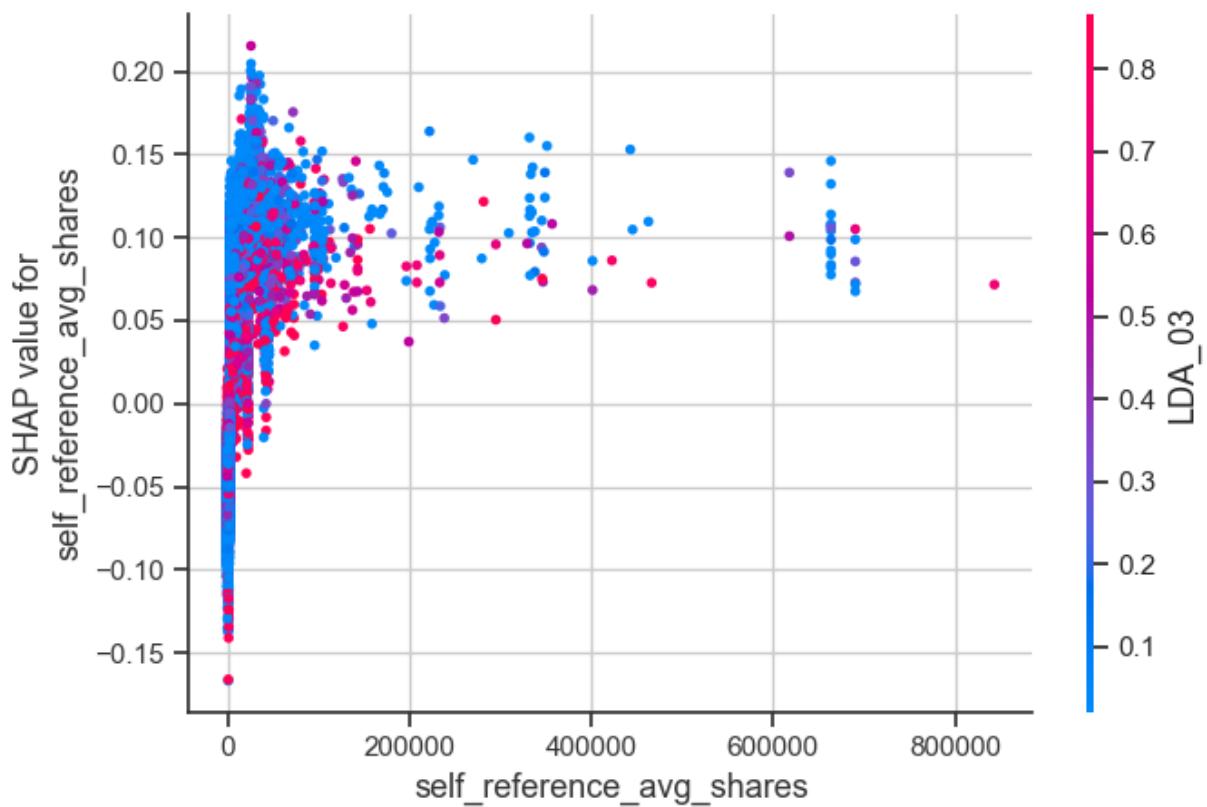


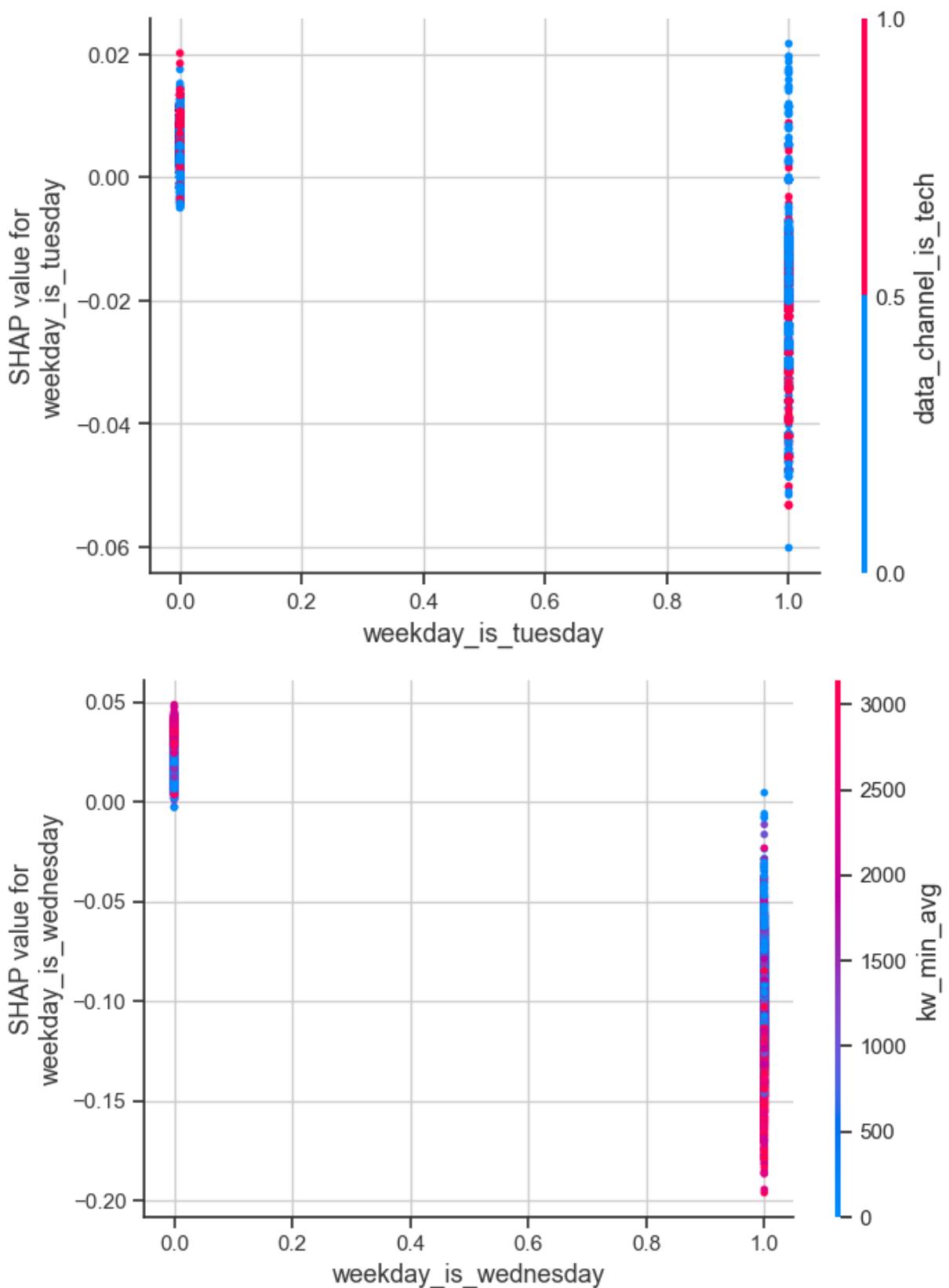


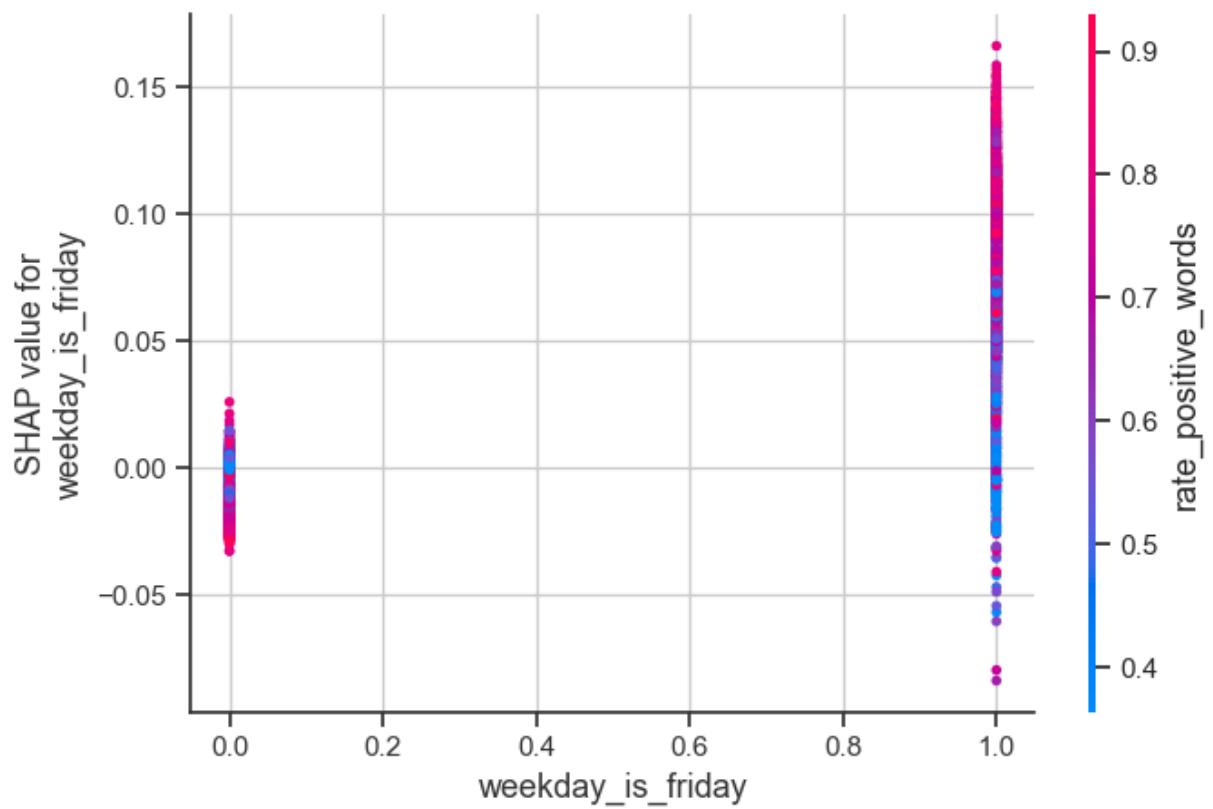
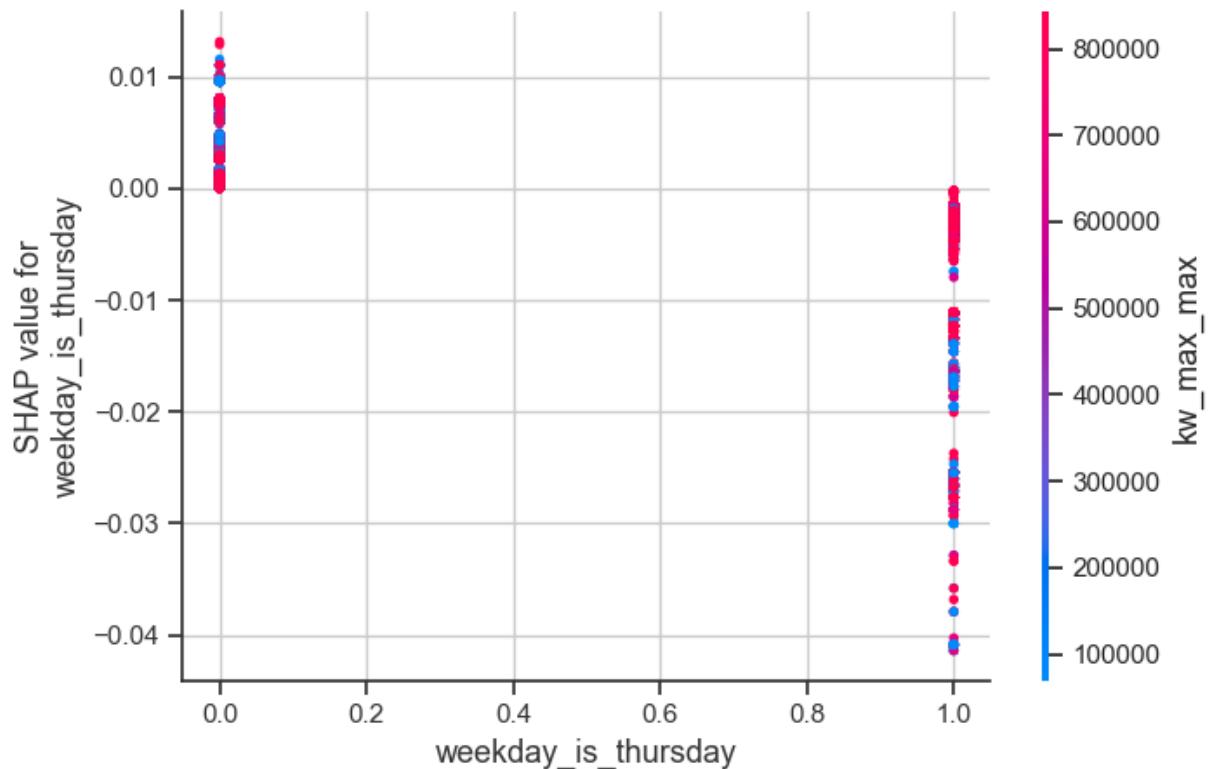


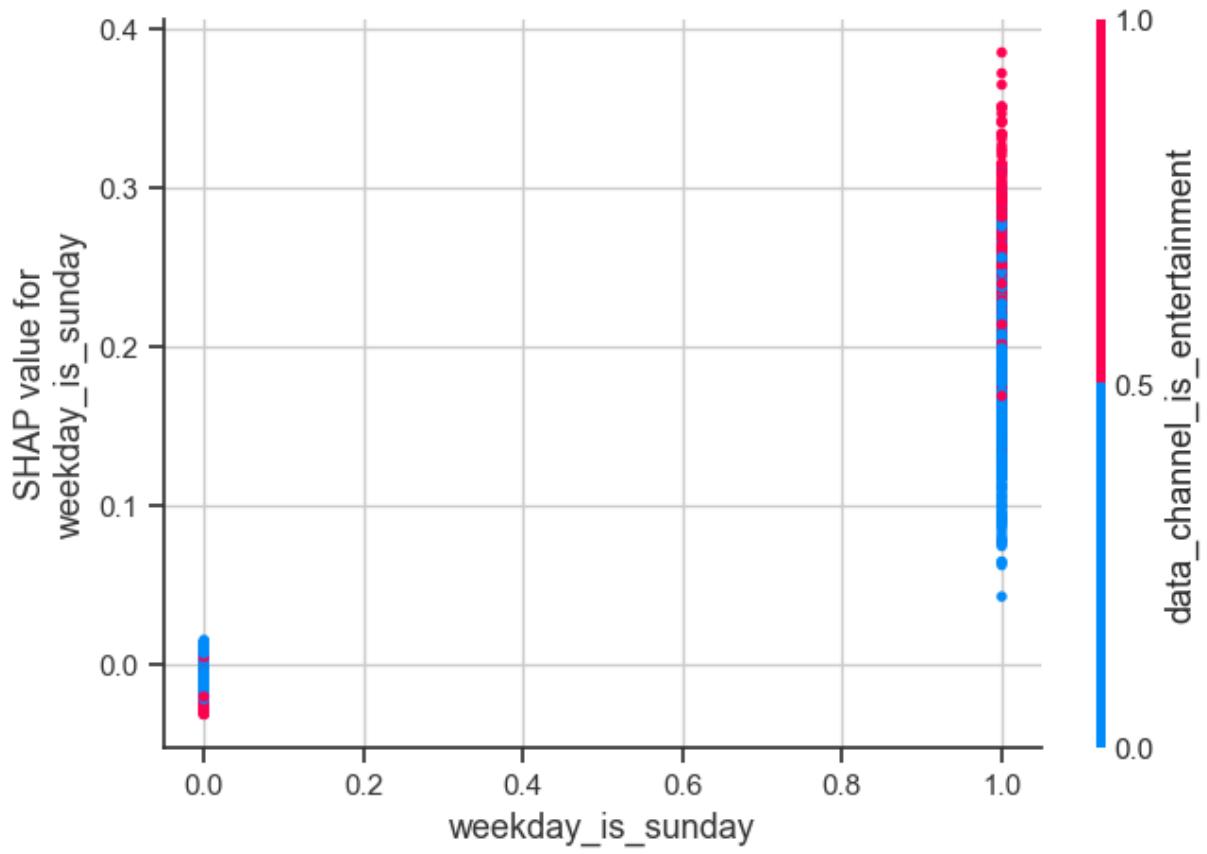
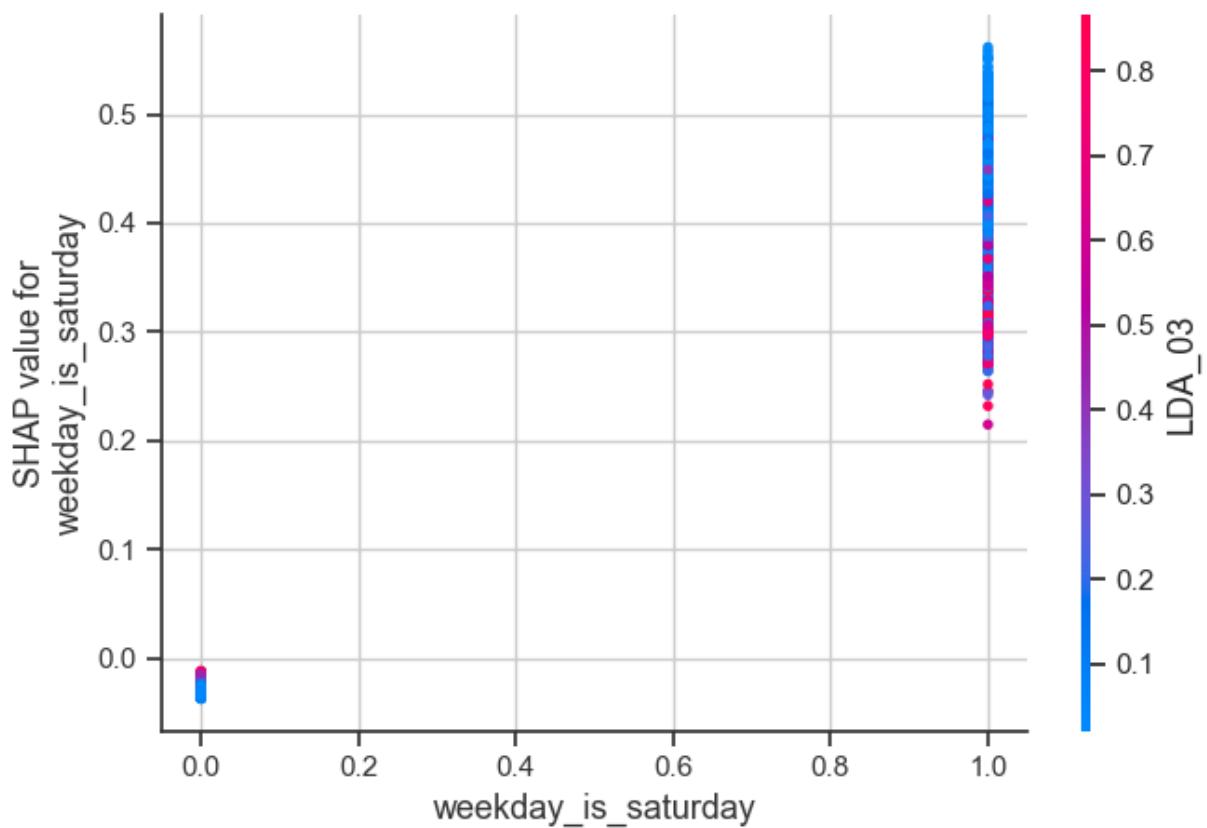


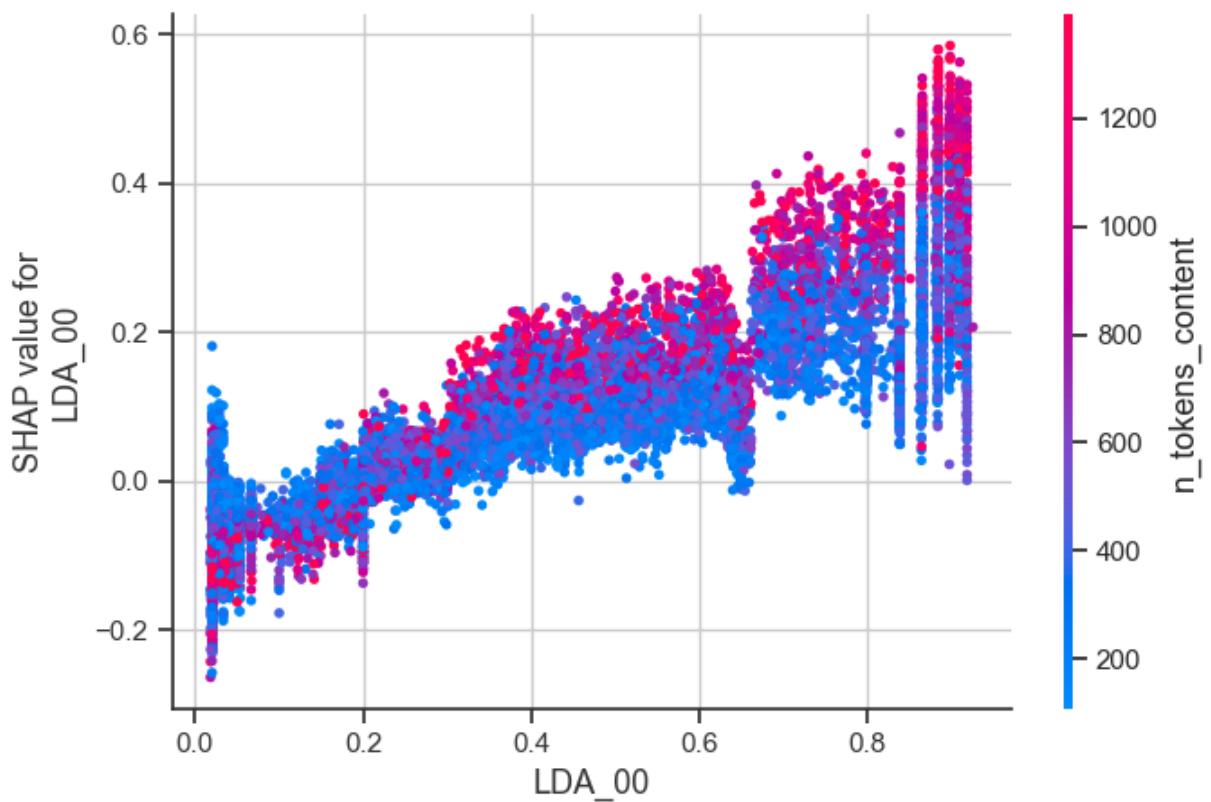
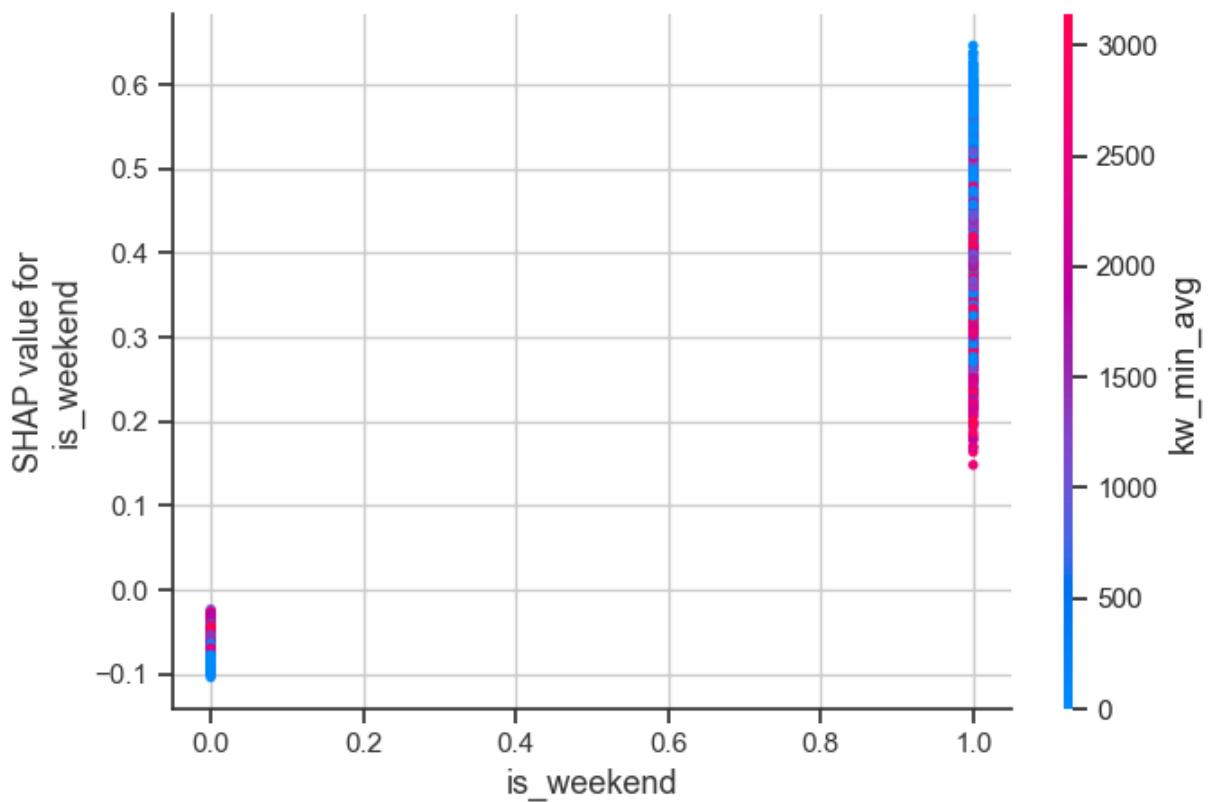


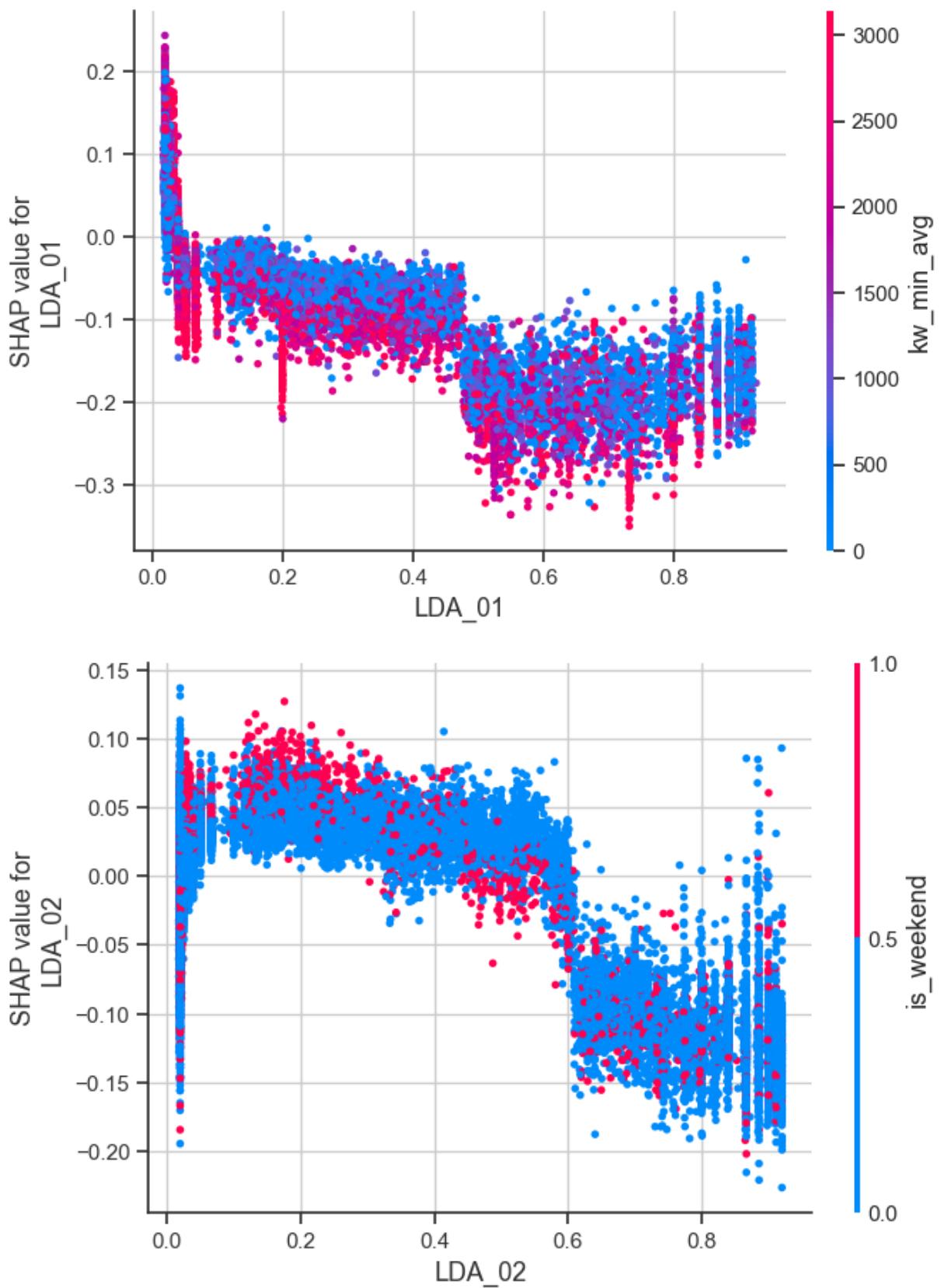


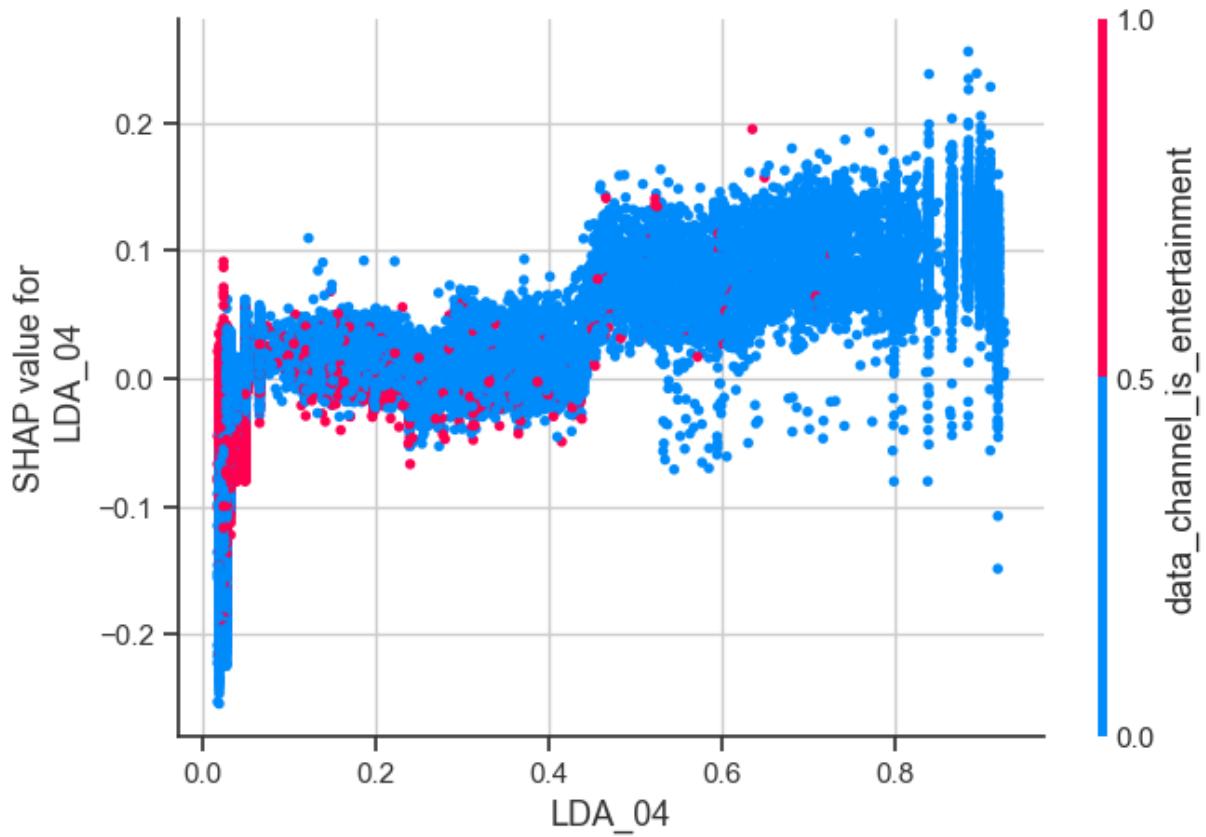
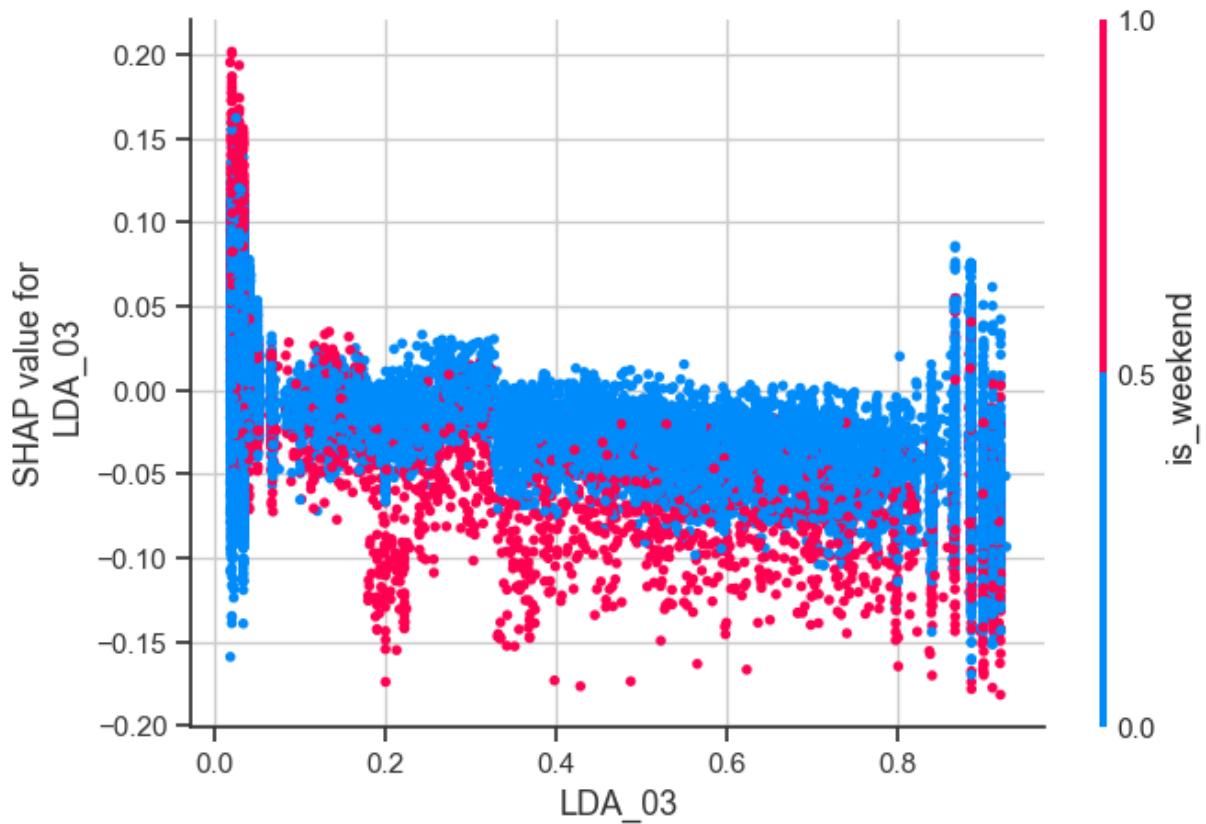


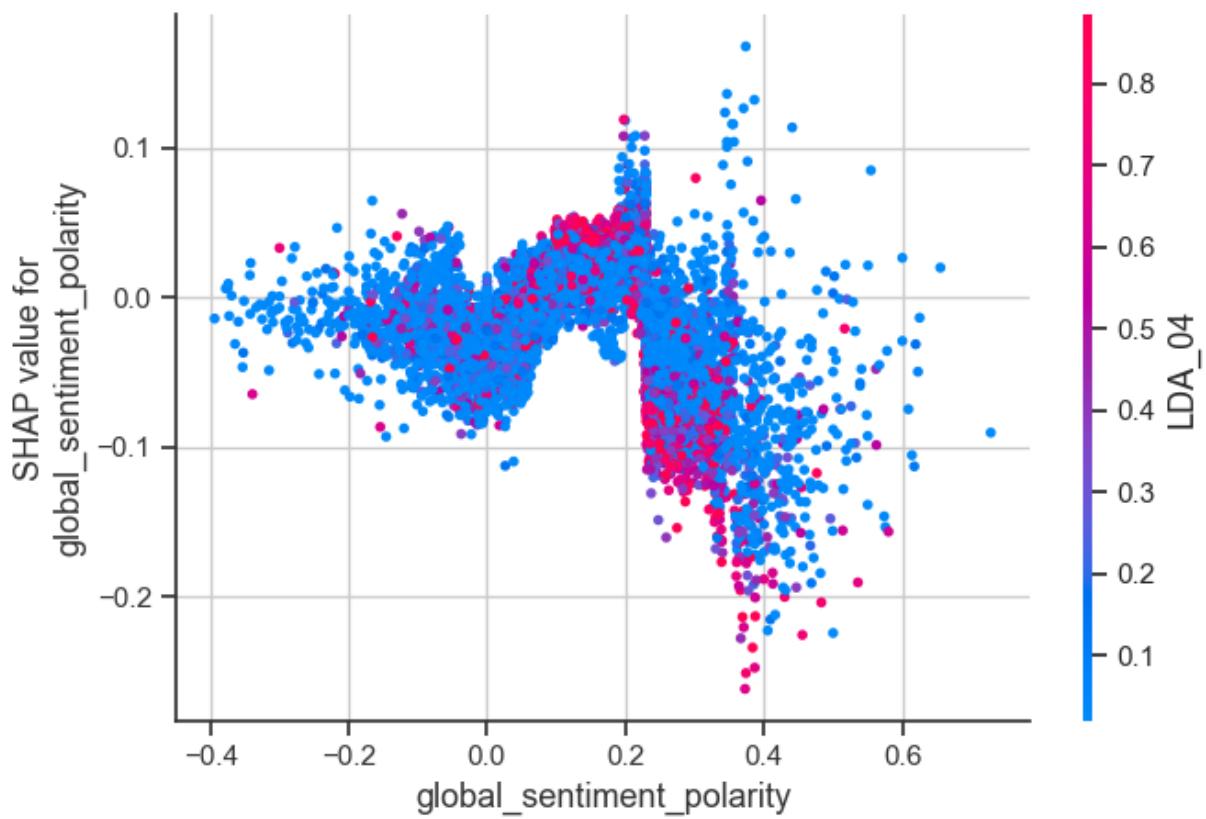
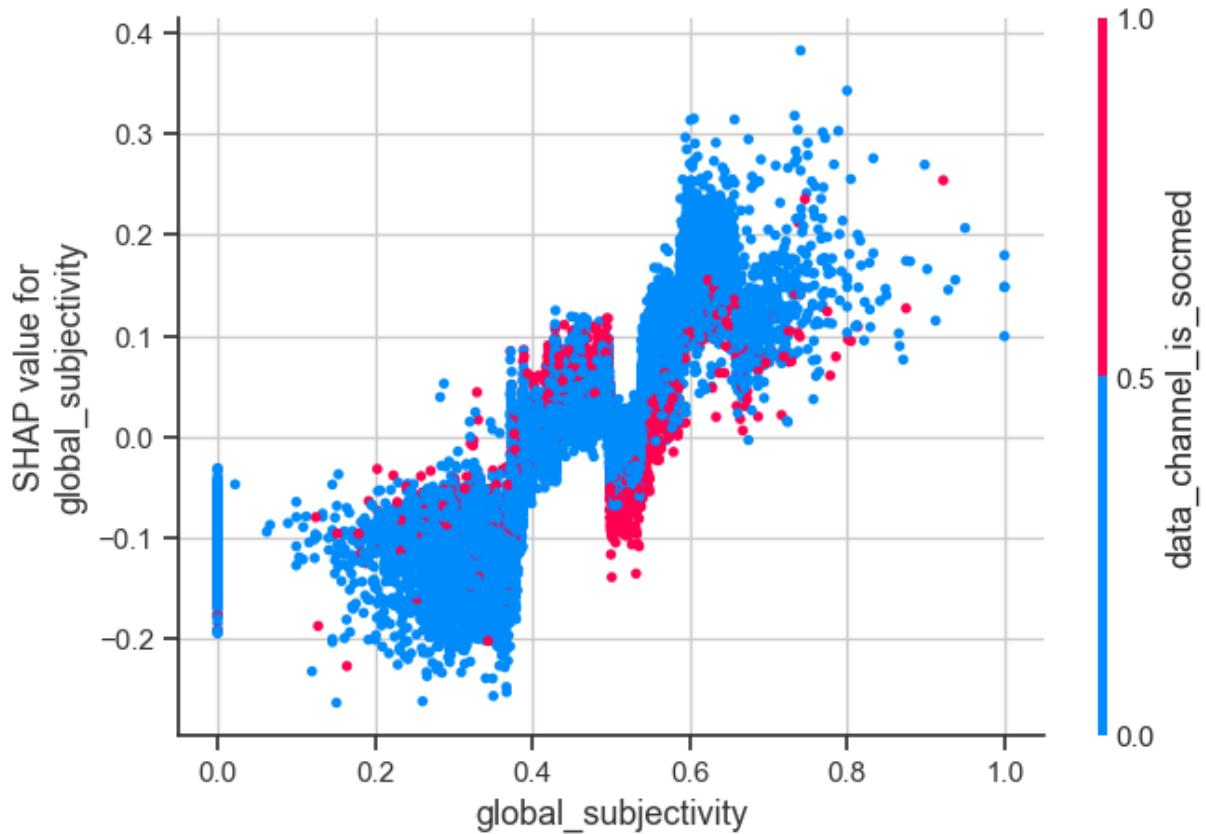


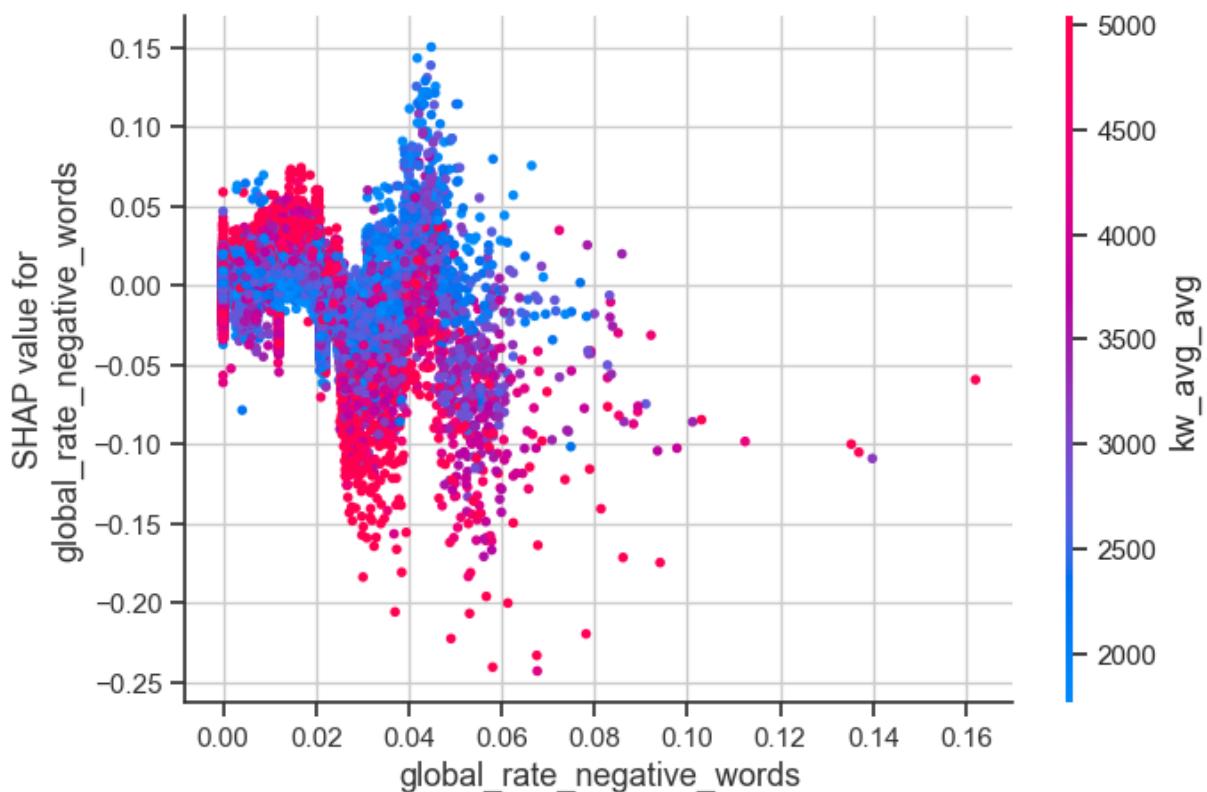
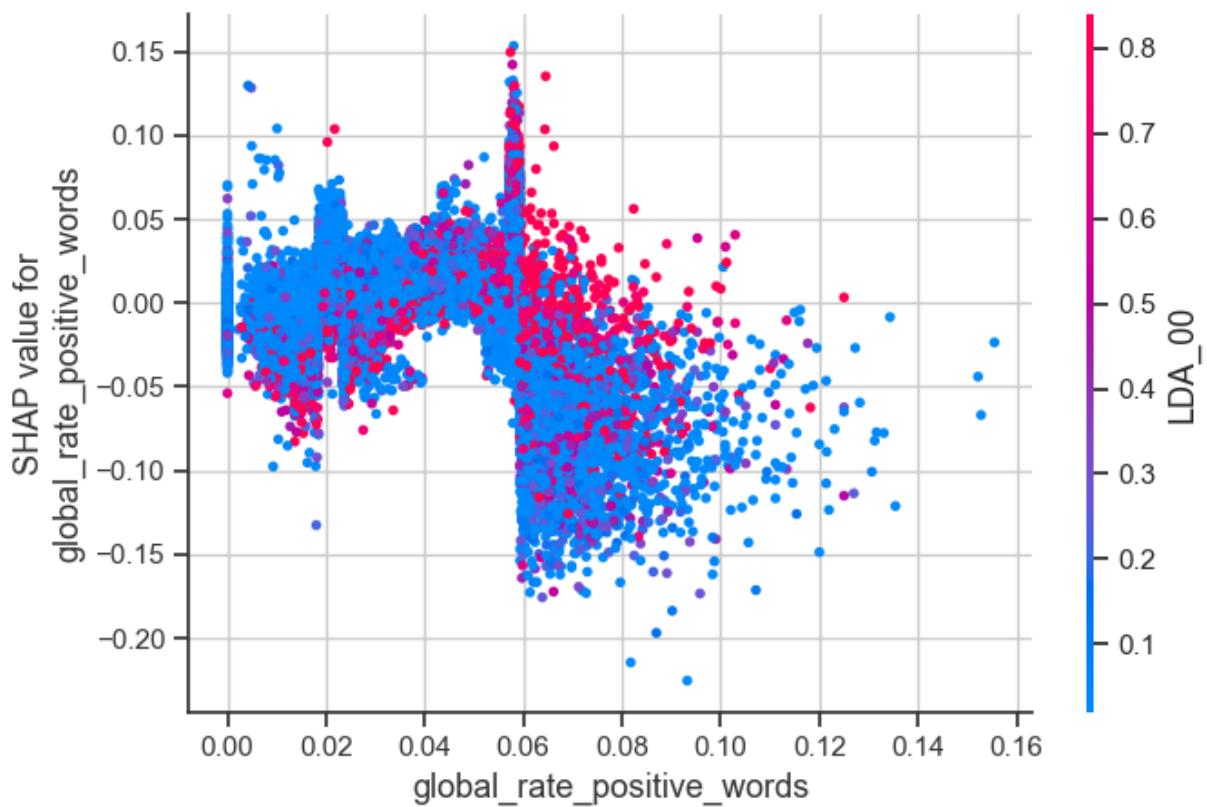


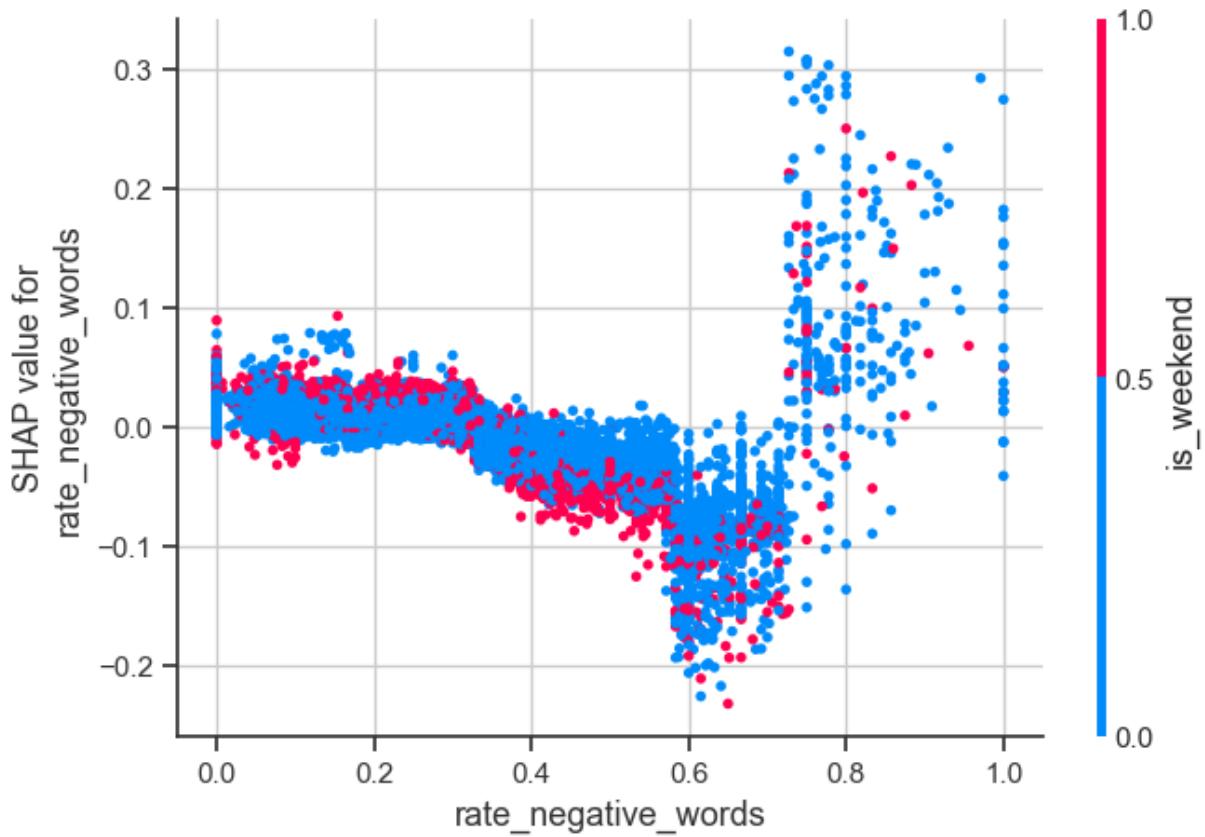
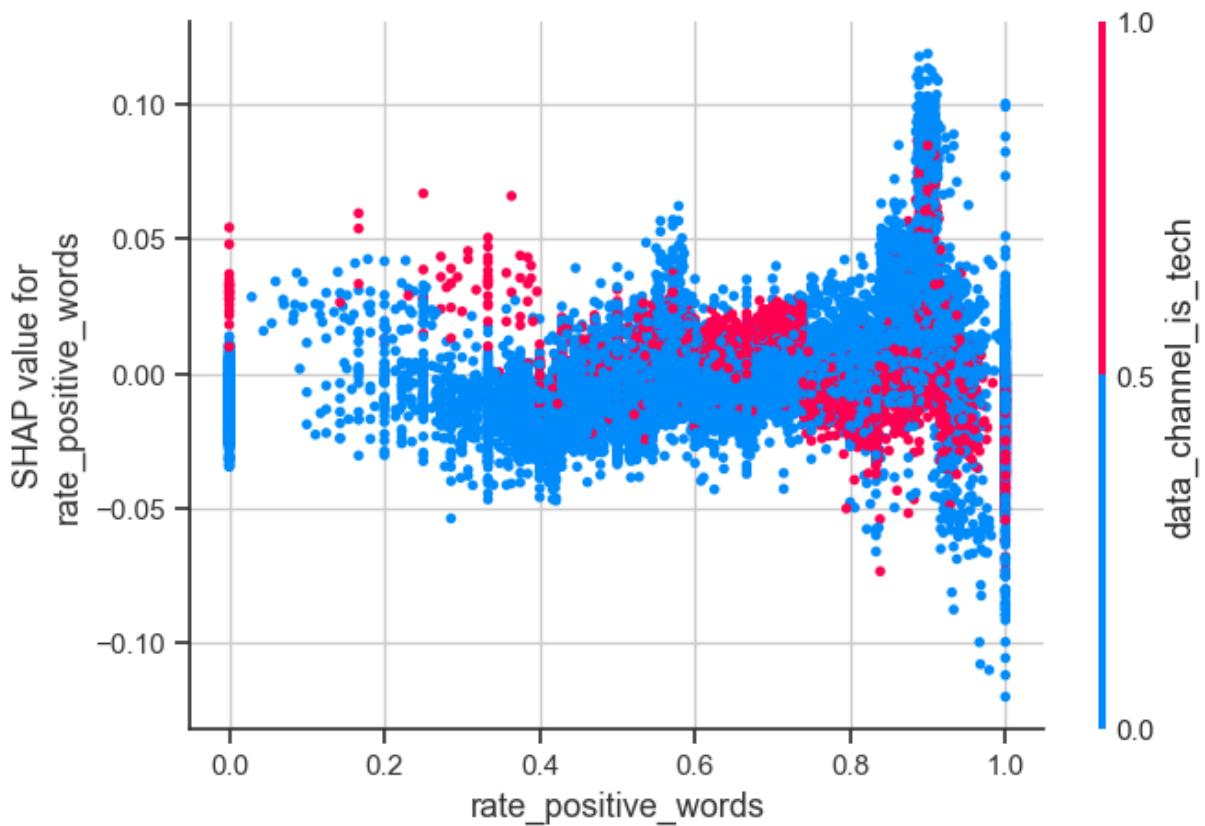


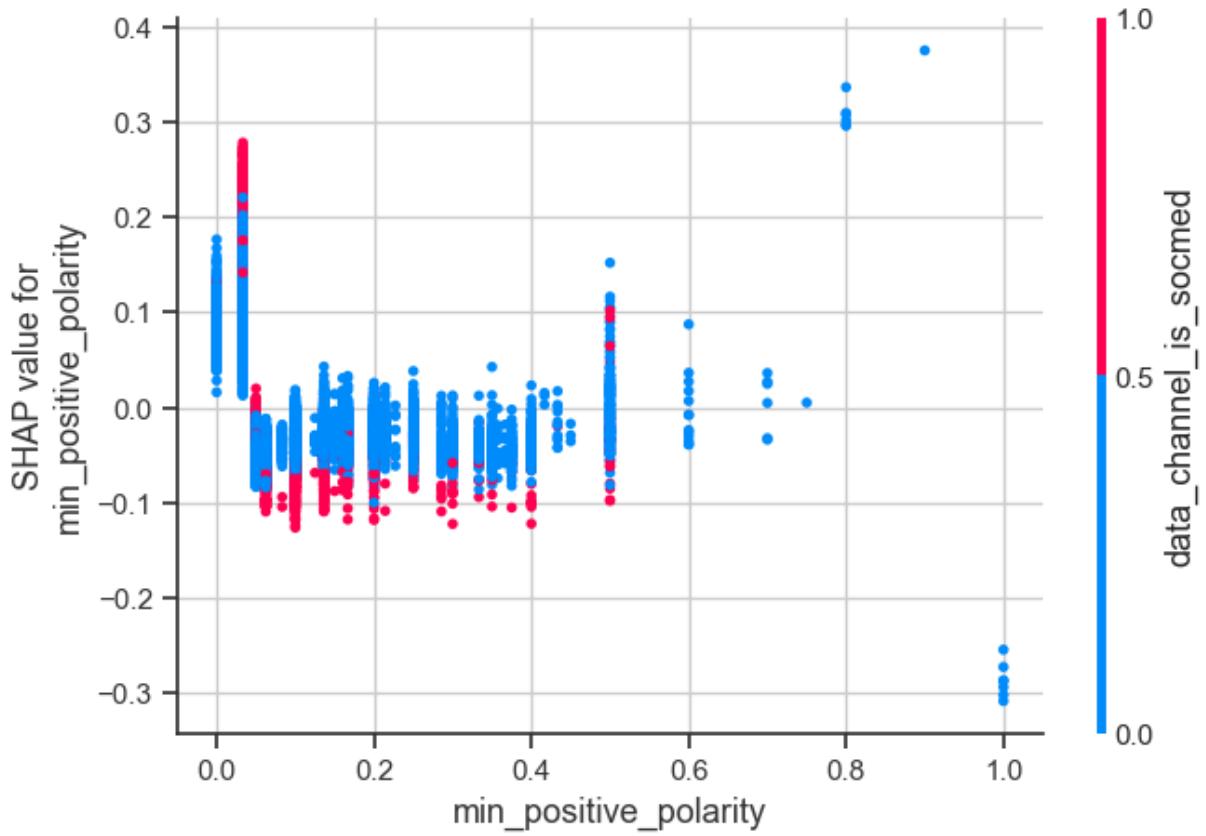
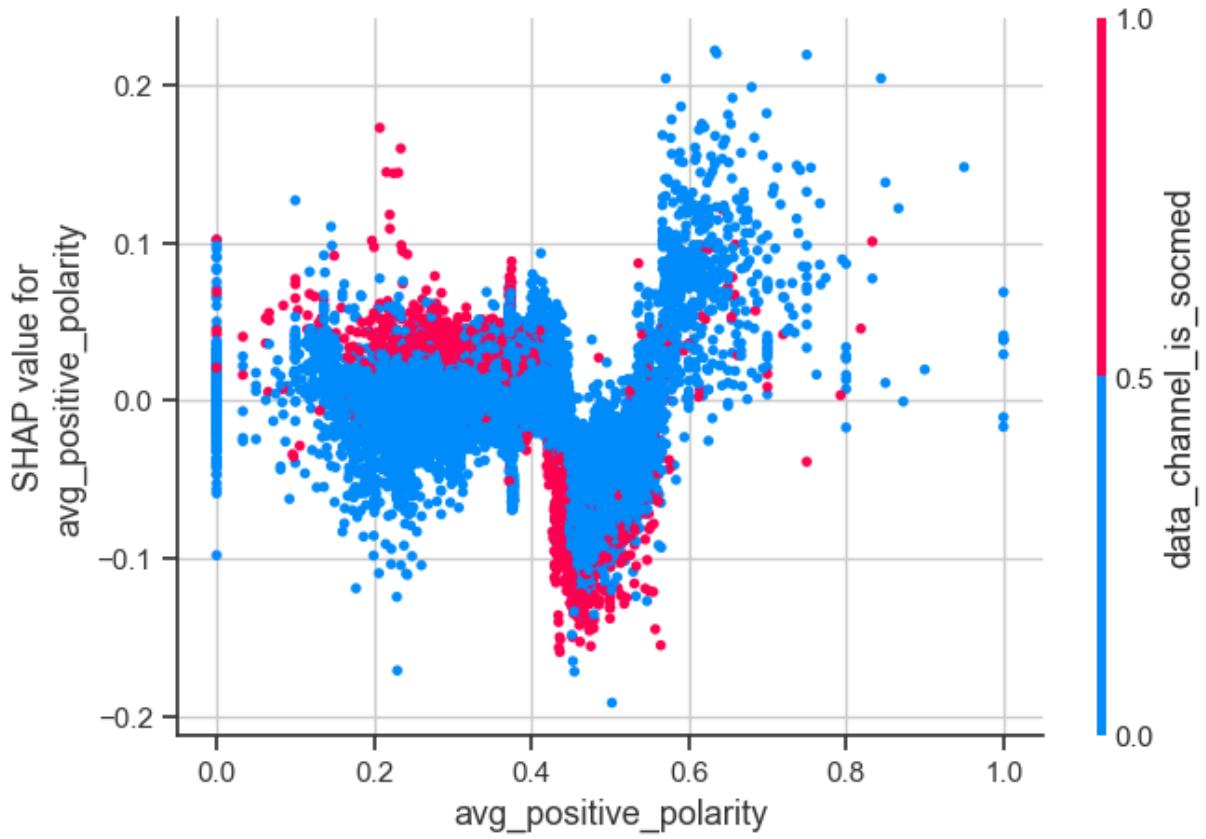


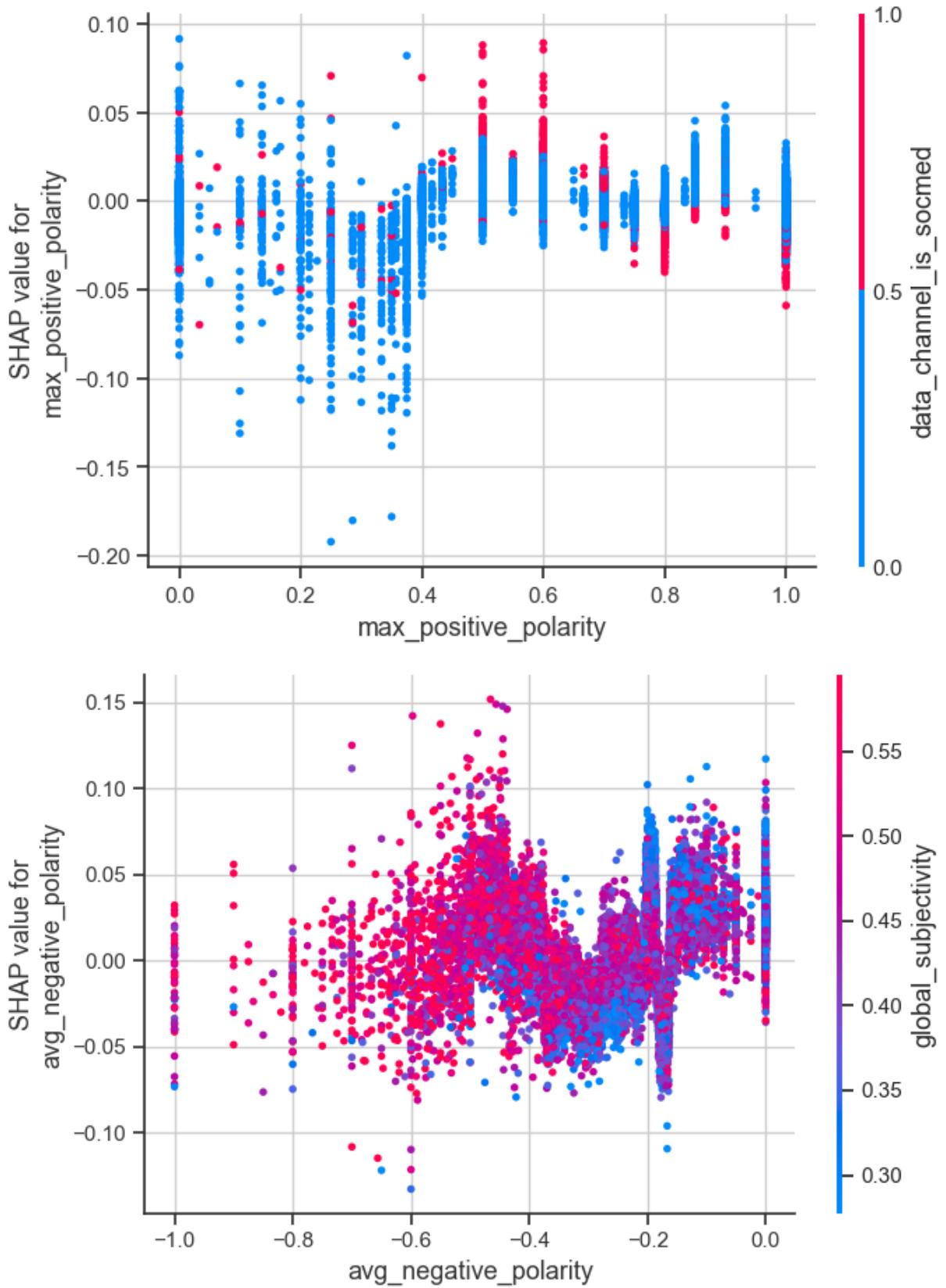


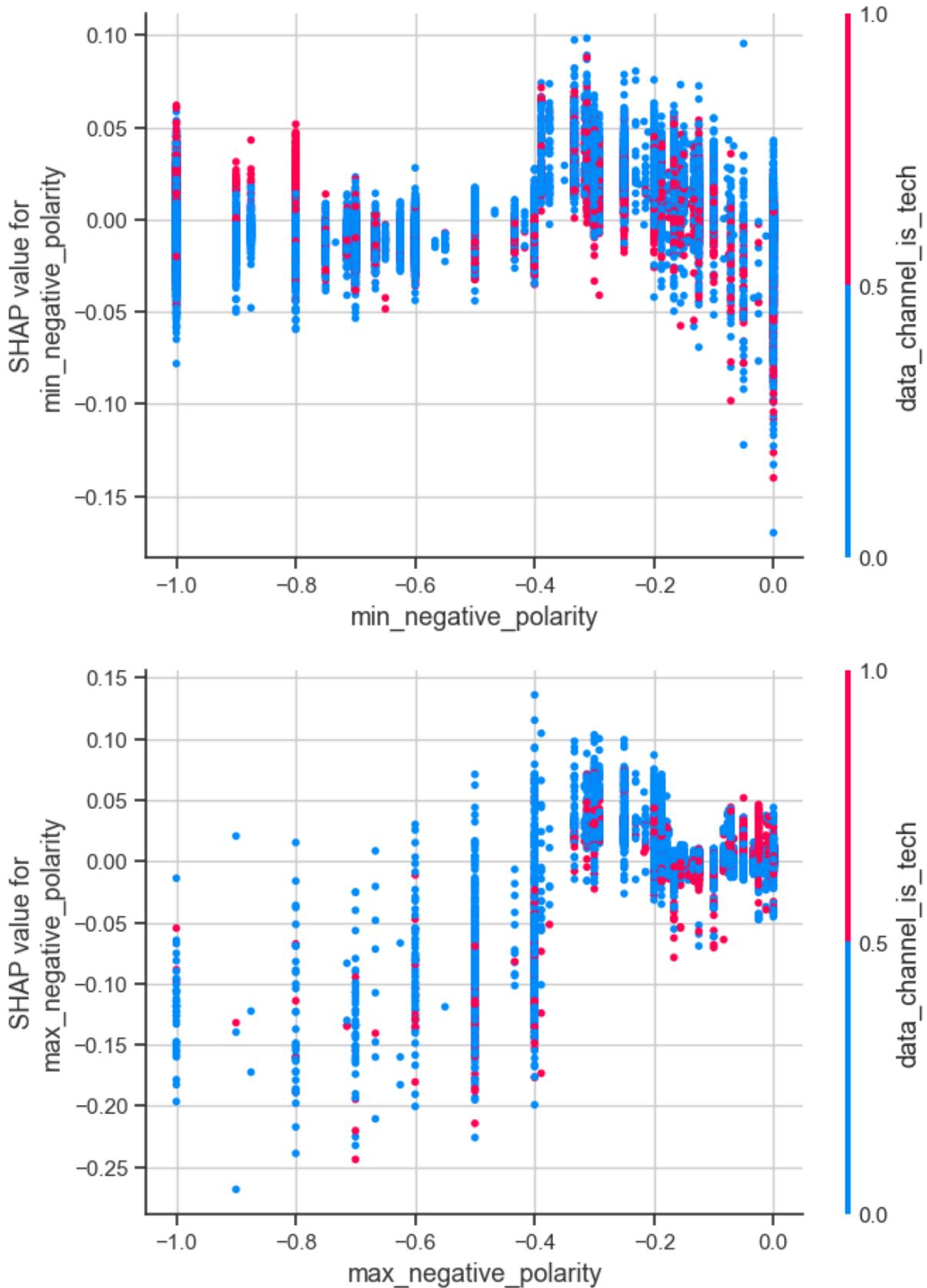


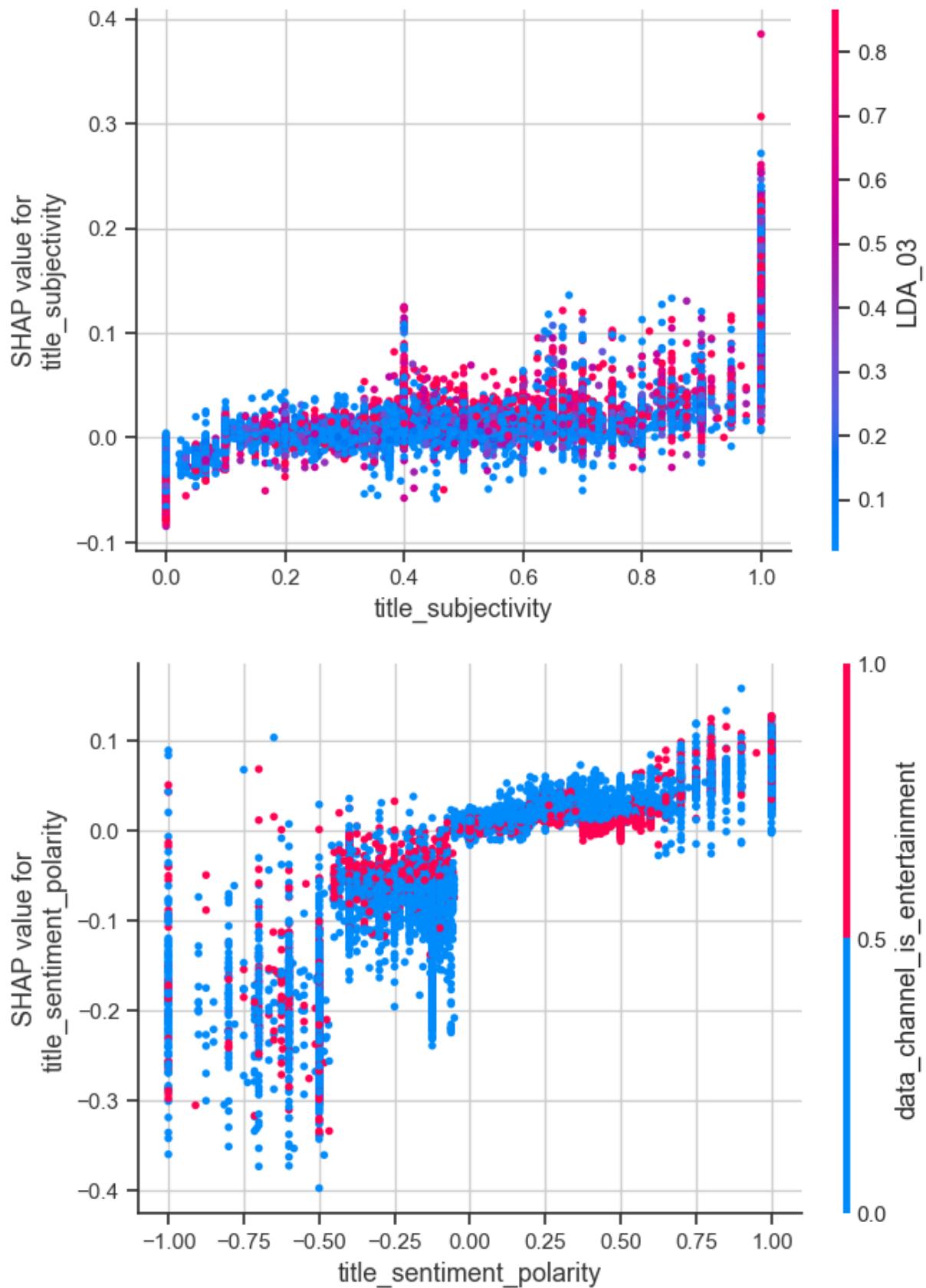


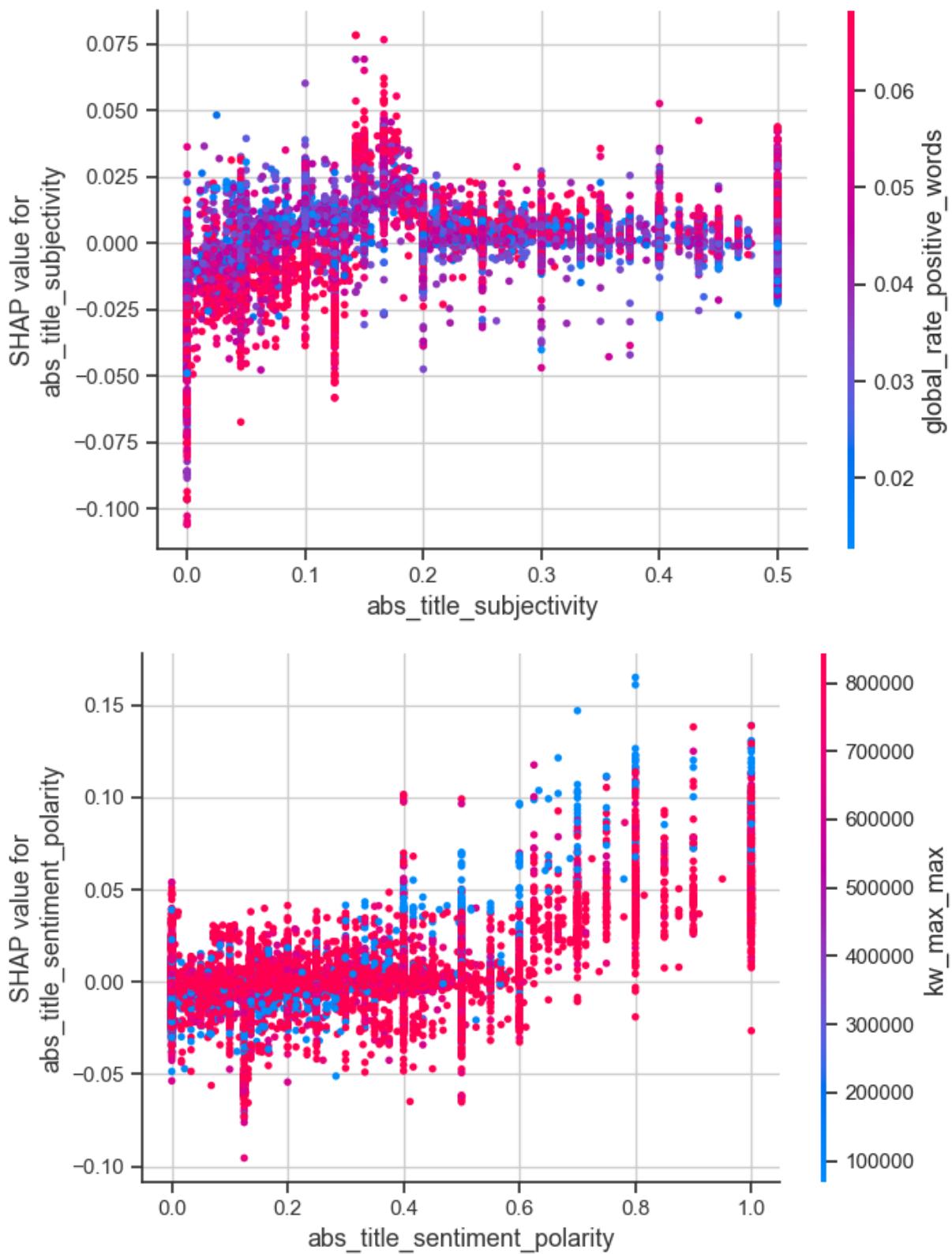












## References

- De Dios, Ednalyn. (2020). PAPEM-DM: 7 Steps Towards a Data Science Win. Towards Data Science. <https://towardsdatascience.com/papem-dm-7-steps-towards-a-data-science-win-f8cac4f8e02f>

- Fernandes, Kelwin, Vinagre, Pedro, Cortez, Paulo, and Sernadela, Pedro. (2015). Online News Popularity. UCI Machine Learning Repository. <https://doi.org/10.24432/C5NS3V>.

```
In [65]: print('Successful run!')
```

```
Successful run!
```