

Predicting Virality with Extreme Gradient Boosting on Online News Popularity Data

Ednalyn C. De Dios

D214

September 1, 2023

Western Governors University

Predicting Virality with Extreme Gradient Boosting on Online News Popularity Data

In this digital age that we're in, understanding the different factors that contribute to the popularity of online news articles is a crucial endeavor for media organizations, marketing professionals, and content creators alike. Boundless amount of data exists and presents a challenge of distilling patterns to uncover hidden insights regarding user behavior.

Furthermore, identifying key factors that determine an online news article's popularity has become the holy grail of many, including data scientists. Machine learning techniques like XGBoost help uncover these hidden insights and translate them into actionable nuggets of information that stakeholders can act upon.

XGBoost, which stands for eXtreme Gradient Boosting, is a powerful and versatile machine learning algorithm that has gained popularity in recent years due to its effectiveness in handling a wide range of predictive tasks. Created by Tianqi Chen, XGBoost is "a scalable tree boosting system" (2023) equipped with both sequential and parallel architectures (Suginoo, 2022). A supervised learning algorithm, gradient boosting predicts "a target variable by combining the estimates of a set of simpler, weaker models" (How XGBoost Works, n.d.).

A – Research Question

In this study, eXtreme Gradient Boosting or XGBoost is used to analyze the Online News Popularity Data by Fernandes et al. (2015) and predict the popularity of online news articles. The aim is to identify which attributes in the available data are key factors driving the number of social media shares. Stakeholders would like to know this information to optimize their content for "going viral." This study will examine the different relationship between different attributes of the data in relation to the designation of whether the online news article is

popular or not as defined by the number of times the article has been shared on social media channels. Thus, the question can then be summarized as follows: Do certain factors affect the popularity of online news articles?

The null hypothesis of the research question is that certain factors do not have a significant effect on the number of social media shares. For example, an online news article will have the same number of social media shares regardless of how many keywords it is using. The alternative hypothesis is that one or more factors have a significant effect on the number of social media shares. For example, the number of social media shares an online news article will have been greatly dependent on the number of keywords that it uses. If a significant effect is found on the number of social media shares, then it will be worthwhile to optimize online news articles for those factors. Exploratory data analysis will be used to determine whether the null hypothesis can be rejected or not.

B – Data Collection

This study uses the “Online News Popularity” dataset which is publicly available from the UC Irvine Machine Learning Repository project (Fernandes et al., 2015). The dataset contains statistics on articles published by Mashable.com. The dataset contains 39,797 records and 61 attributes, of which 58 are predictive, two are non-predictive, and one goal field. The dataset is publicly available to the public and licensed under [Creative Commons Attribution 4.0 International](#) (CC BY 4.0). The following table lists the name, type, and description of the attributes that were included in this study.

Field	Type	Description
url	Categorical	URL of the article

timedelta	Continuous	Days between the article publication and the dataset acquisition
n_tokens_title	Continuous	Number of words in the title
n_tokens_content	Continuous	Number of words in the content
n_unique_tokens	Continuous	Rate of unique words in the content
n_non_stop_words	Continuous	Rate of non-stop words in the content
n_non_stop_unique_tokens	Continuous	Rate of unique non-stop words in the content
num_hrefs	Continuous	Number of links
num_self_hrefs	Continuous	Number of links to other articles published by Mashable
num_imgs	Continuous	Number of images
num_videos	Continuous	Number of videos
average_token_length	Continuous	Average length of the words in the content
num_keywords	Continuous	Number of keywords in the metadata
data_channel_is_lifestyle	Categorical	Is data channel 'Lifestyle'?
data_channel_is_entertainment	Categorical	Is data channel 'Entertainment'?
data_channel_is_bus	Categorical	Is data channel 'Business'?
data_channel_is_socmed	Categorical	Is data channel 'Social Media'?
data_channel_is_tech	Categorical	Is data channel 'Tech'?
data_channel_is_world	Categorical	Is data channel 'World'?
kw_min_min	Continuous	Worst keyword (min. shares)
kw_max_min	Continuous	Worst keyword (max. shares)
kw_avg_min	Continuous	Worst keyword (avg. shares)
kw_min_max	Continuous	Best keyword (min. shares)
kw_max_max	Continuous	Best keyword (max. shares)
kw_avg_max	Continuous	Best keyword (avg. shares)
kw_min_avg	Continuous	Avg. keyword (min. shares)
kw_max_avg	Continuous	Avg. keyword (max. shares)
kw_avg_avg	Continuous	Avg. keyword (avg. shares)

self_reference_min_shares	Continuous	Min. shares of referenced articles in Mashable
self_reference_max_shares	Continuous	Max. shares of referenced articles in Mashable
self_reference_avg_shares	Continuous	Avg. shares of referenced articles in Mashable
weekday_is_monday	Categorical	Was the article published on a Monday?
weekday_is_tuesday	Categorical	Was the article published on a Tuesday?
weekday_is_wednesday	Categorical	Was the article published on a Wednesday?
weekday_is_thursday	Categorical	Was the article published on a Thursday?
weekday_is_friday	Categorical	Was the article published on a Friday?
weekday_is_saturday	Categorical	Was the article published on a Saturday?
weekday_is_sunday	Categorical	Was the article published on a Sunday?
is_weekend	Categorical	Was the article published on the weekend?
LDA_00	Categorical	Closeness to LDA topic 0
LDA_01	Categorical	Closeness to LDA topic 1
LDA_02	Categorical	Closeness to LDA topic 2
LDA_03	Categorical	Closeness to LDA topic 3
LDA_04	Categorical	Closeness to LDA topic 4
global_subjectivity	Continuous	Text subjectivity
global_sentiment_polarity	Continuous	Text sentiment polarity
global_rate_positive_words	Continuous	Rate of positive words in the content
global_rate_negative_words	Continuous	Rate of negative words in the content
rate_positive_words	Continuous	Rate of positive words among non-neutral tokens
rate_negative_words	Continuous	Rate of negative words among non-neutral tokens
avg_positive_polarity	Continuous	Avg. polarity of positive words
min_positive_polarity	Continuous	Min. polarity of positive words
max_positive_polarity	Continuous	Max. polarity of positive words

avg_negative_polarity	Continuous	Avg. polarity of negative words
min_negative_polarity	Continuous	Min. polarity of negative words
max_negative_polarity	Continuous	Max. polarity of negative words
title_subjectivity	Continuous	Title subjectivity
title_sentiment_polarity	Continuous	Title polarity
abs_title_subjectivity	Continuous	Absolute subjectivity level
abs_title_sentiment_polarity	Continuous	Absolute polarity level
shares	Continuous	Number of shares (target)

There are several advantages with the chosen dataset. There are no missing values, the class distribution is relatively balanced, and the categorical variables have already been encoded into numerical form. However, one observed disadvantage is the sheer size of the dataset. One trial run of the experiment took almost three hours to run hyperparameter tuning. The challenge of long running time was overcome by setting the `tree_method` to 'gpu_hist' and limiting the range of the parameter search space.

```
# Read a csv file
df = pd.read_csv('../data/in/OnlineNewsPopularity.csv')
```

C – Data Extraction and Preparation

The next phase of the analysis is data preparation. The tool used is Python 3.9.9 and Jupyter Notebook 7.0.2 was used as the interactive development environment. Python was chosen for its versatility and easy-to-learn syntax. Jupyter Notebook was chosen for its markdown capability. One disadvantage of Python is performance while Jupyter makes good code versioning very difficult (Mueller, 2018).

After reading the data into a dataframe, the next step is to check for any missing values.

```
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_percentage,
                       'num_empty': empty_count, 'empty_percentage': empty_percentage,
                       'nan_count': nan_count, 'nan_percentage': nan_percentage})

    return dfx

show_missing(df)
```

Next is correcting the column names.

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

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

Then, duplicates are dropped if any. There weren't any as shown by df.shape before and after the operation.

```
df.shape
df = df.drop_duplicates(keep = False)
df.shape
```

```
(39644, 61)
```

```
(39644, 61)
```

The target variable was created by applying a condition on the 'shares' variable and designating 1 or 0 depending on the THRESHOLD value. The threshold value for this notebook is 1400.

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

Finally, the cleaned and prepared dataset is exported using pandas' to_csv() method.

```
df.to_csv('../data/out/online_news_popularity_clean.csv', index=False)
```

D – Analysis

Once the dataset is prepared, the next step in the analysis is to conduct EDA or explanatory data analysis. The high-levels steps are:

1. Get familiar with the data
2. Review class distribution
3. Get summary statistics
4. Remove outliers
5. Compare the interactions of the variables


```
df.head()
```

	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
0	12.0	219.0	0.663594	1.0	0.815385	4.0	2.0	1.0	0.0	4.68
1	9.0	255.0	0.604743	1.0	0.791946	3.0	1.0	1.0	0.0	4.97
2	9.0	211.0	0.575130	1.0	0.663866	3.0	1.0	1.0	0.0	4.39
3	9.0	531.0	0.503788	1.0	0.665635	9.0	0.0	1.0	0.0	4.40
4	13.0	1072.0	0.415646	1.0	0.540890	19.0	19.0	20.0	0.0	4.68

<

>

```
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

```

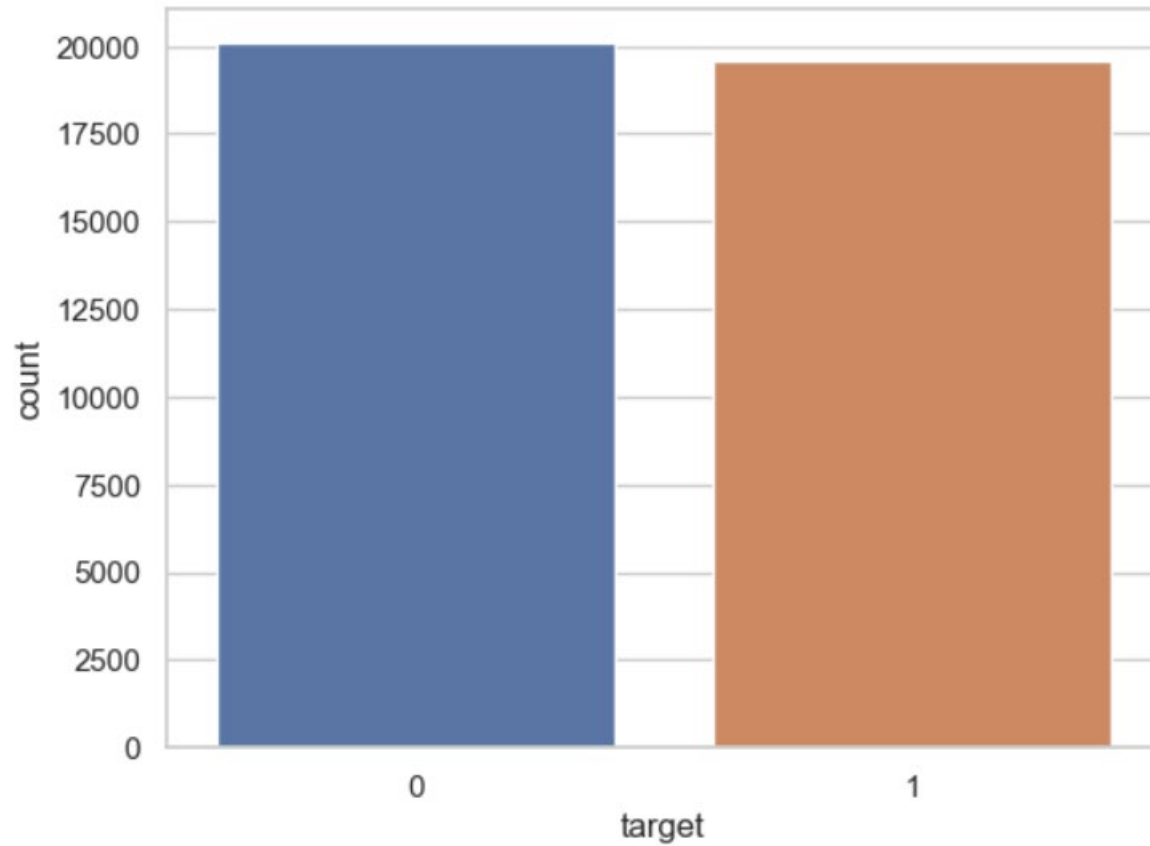
--      kw_min_max      39644 non-null float64
20      kw_max_max      39644 non-null float64
21      kw_avg_max      39644 non-null float64
22      kw_min_avg      39644 non-null float64
23      kw_max_avg      39644 non-null float64
24      kw_avg_avg      39644 non-null float64
25      self_reference_min_shares 39644 non-null float64
26      self_reference_max_shares 39644 non-null float64
27      self_reference_avg_shares 39644 non-null float64
28      weekday_is_monday 39644 non-null float64
29      weekday_is_tuesday 39644 non-null float64
30      weekday_is_wednesday 39644 non-null float64
31      weekday_is_thursday 39644 non-null float64
32      weekday_is_friday 39644 non-null float64
33      weekday_is_saturday 39644 non-null float64
34      weekday_is_sunday 39644 non-null float64
35      is_weekend      39644 non-null float64
36      LDA_00          39644 non-null float64
37      LDA_01          39644 non-null float64
38      LDA_02          39644 non-null float64
39      LDA_03          39644 non-null float64
40      LDA_04          39644 non-null float64
41      global_subjectivity 39644 non-null float64
42      global_sentiment_polarity 39644 non-null float64
43      global_rate_positive_words 39644 non-null float64
44      global_rate_negative_words 39644 non-null float64
45      rate_positive_words 39644 non-null float64
46      rate_negative_words 39644 non-null float64
47      avg_positive_polarity 39644 non-null float64
48
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

```

As shown in the graph below, the dataset has a relatively balanced class.

```
sns.countplot(x='target', data=df)
```

```
<Axes: xlabel='target', ylabel='count'>
```



```
df.describe()
```

	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	avera
count	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	39644.000000	
mean	10.398749	546.514731	0.548216	0.996469	0.689175	10.883690	3.293638	4.544143	1.249874	
std	2.114037	471.107508	3.520708	5.231231	3.264816	11.332017	3.855141	8.309434	4.107855	
min	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	9.000000	246.000000	0.470870	1.000000	0.625739	4.000000	1.000000	1.000000	0.000000	
50%	10.000000	409.000000	0.539226	1.000000	0.690476	8.000000	3.000000	1.000000	0.000000	
75%	12.000000	716.000000	0.608696	1.000000	0.754630	14.000000	4.000000	4.000000	1.000000	
max	23.000000	8474.000000	701.000000	1042.000000	650.000000	304.000000	116.000000	128.000000	91.000000	

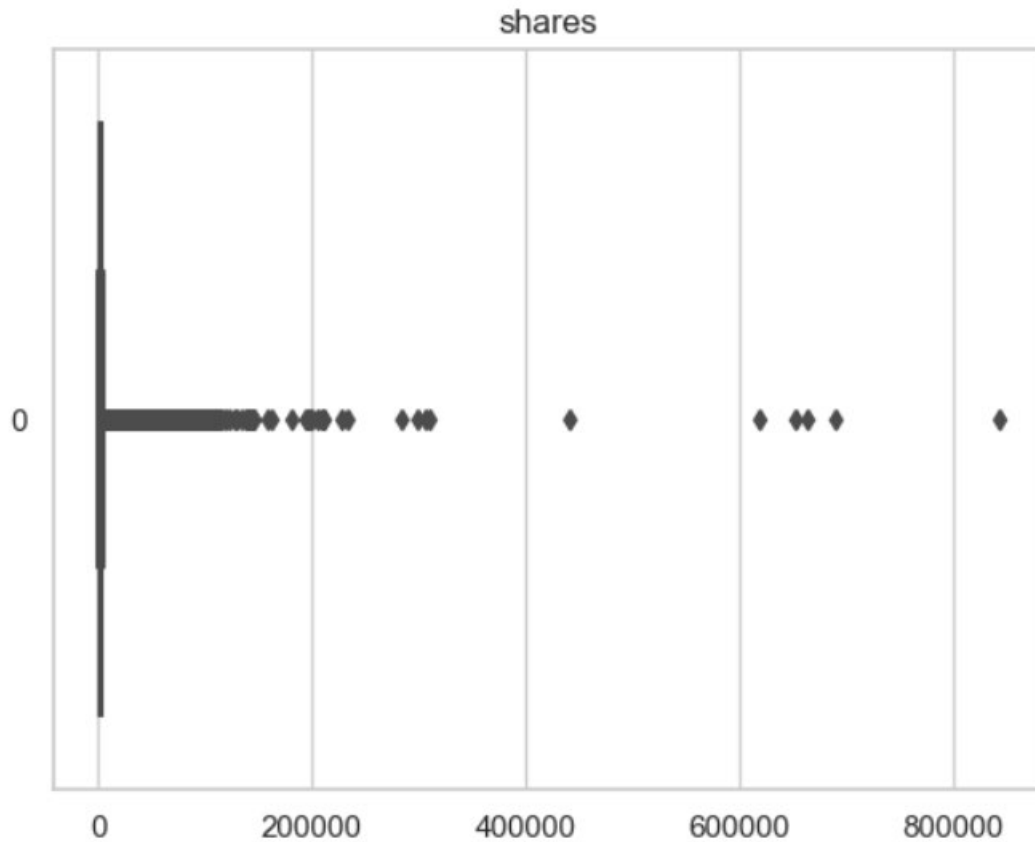
◀

▶

Next is to conduct a visual inspection of the boxplot to look for outliers in the dataset.

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

```
viz_box(df, 'shares')
```



After confirming the existence of outliers visually, the interquartile range or IQR was calculated so that the records outside the lower and upper bound could be removed.

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

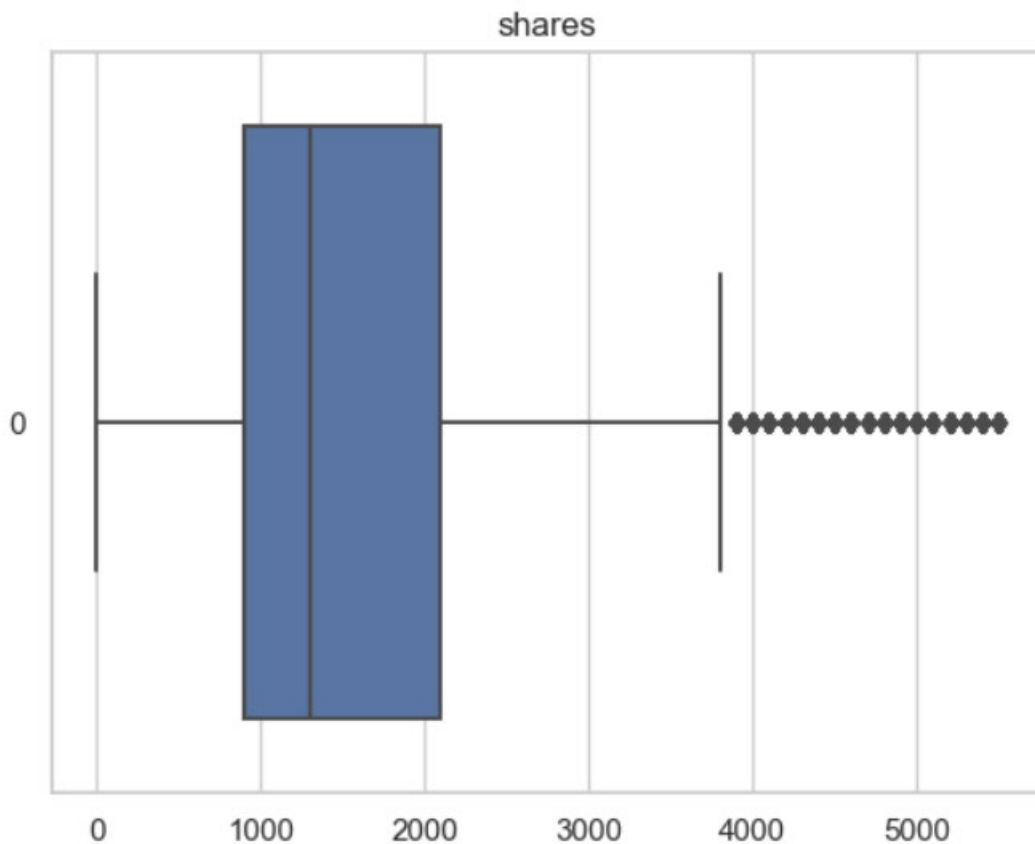
print("75th quartile: ", percentile75)
print("25th quartile: ", percentile25)

iqr = percentile75 - percentile25

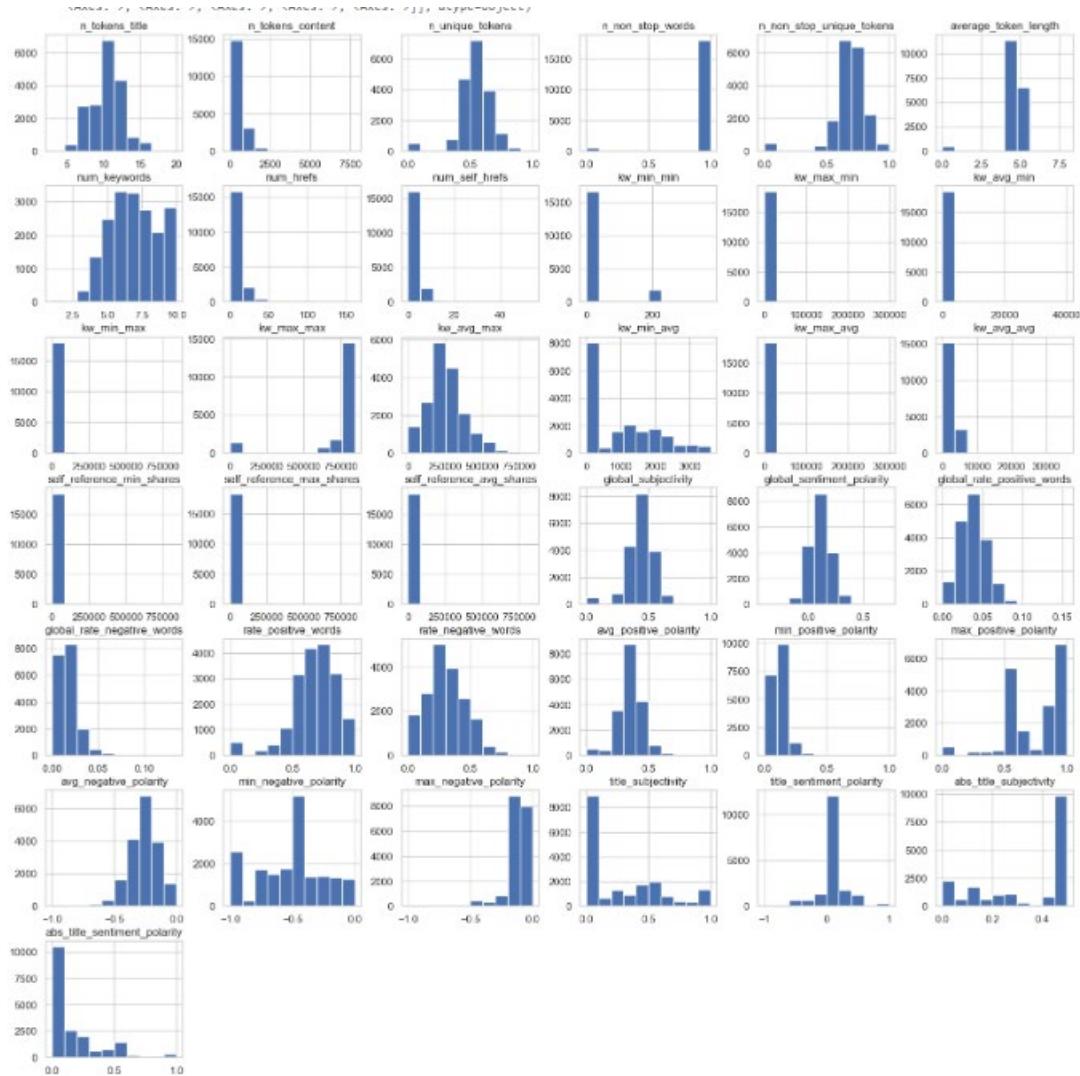
upper_bound = percentile75 + 1.5 * iqr
lower_bound = percentile25 - 1.5 * iqr

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

```
viz_box(df, 'shares')
```

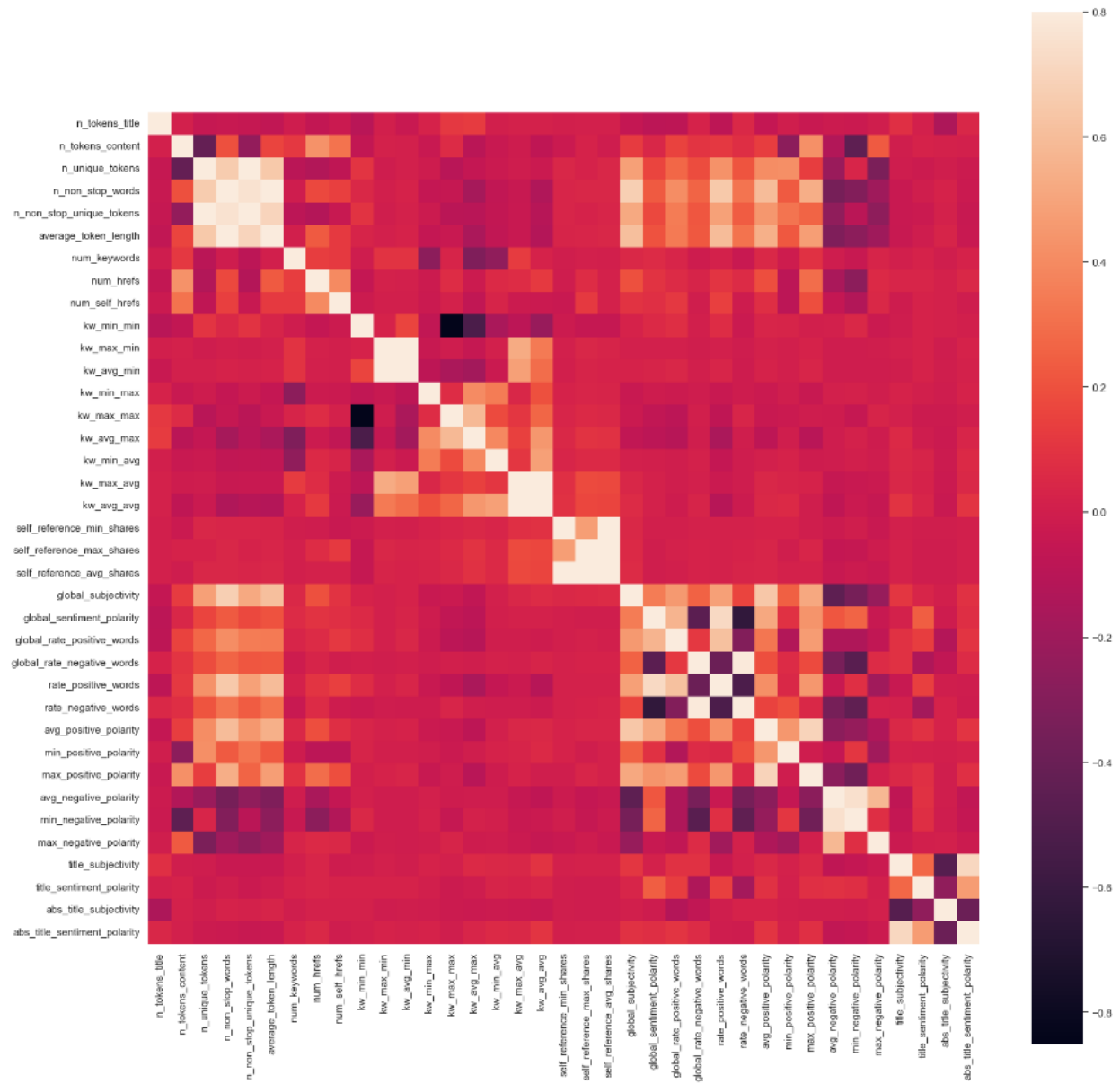


Next is the plotting of histograms for the continuous variables. As shown, the continuous variables are not distributed normally.

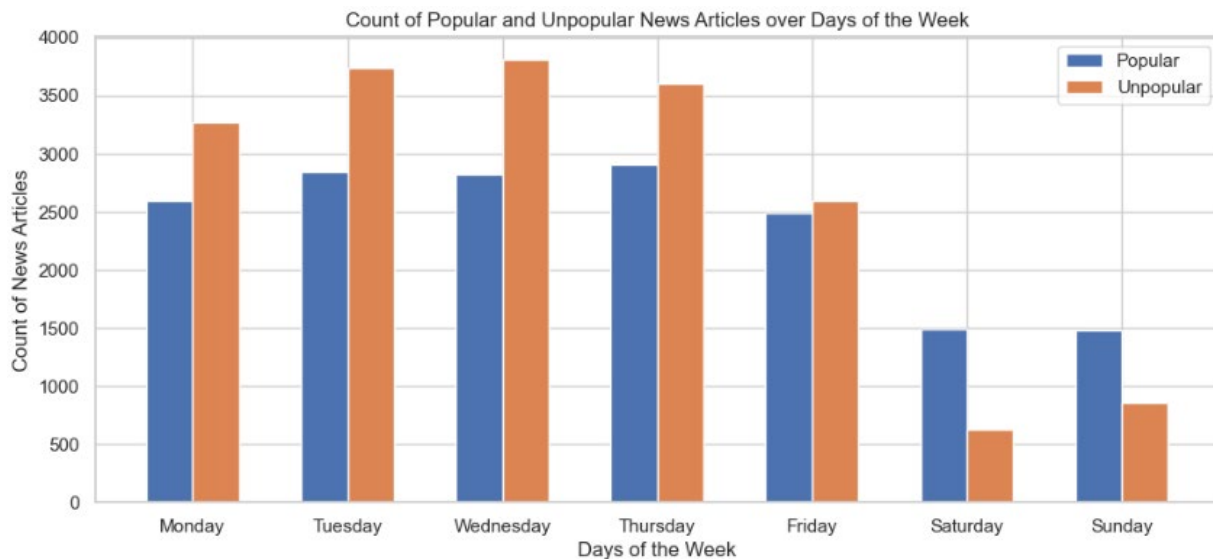


The correlation heatmap shows a few interesting relationship between several variables.

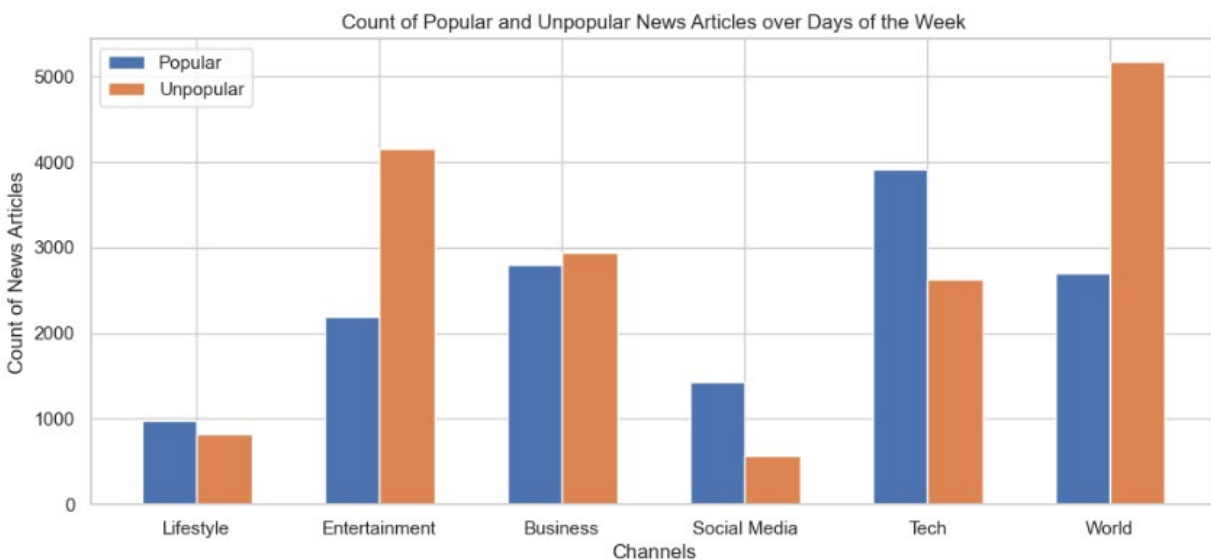
```
corr = popular_df[continuous_cols].corr()
fig = plt.figure(figsize = (20,20))
sns.heatmap(corr, vmax = .8, square = True)
plt.show()
```



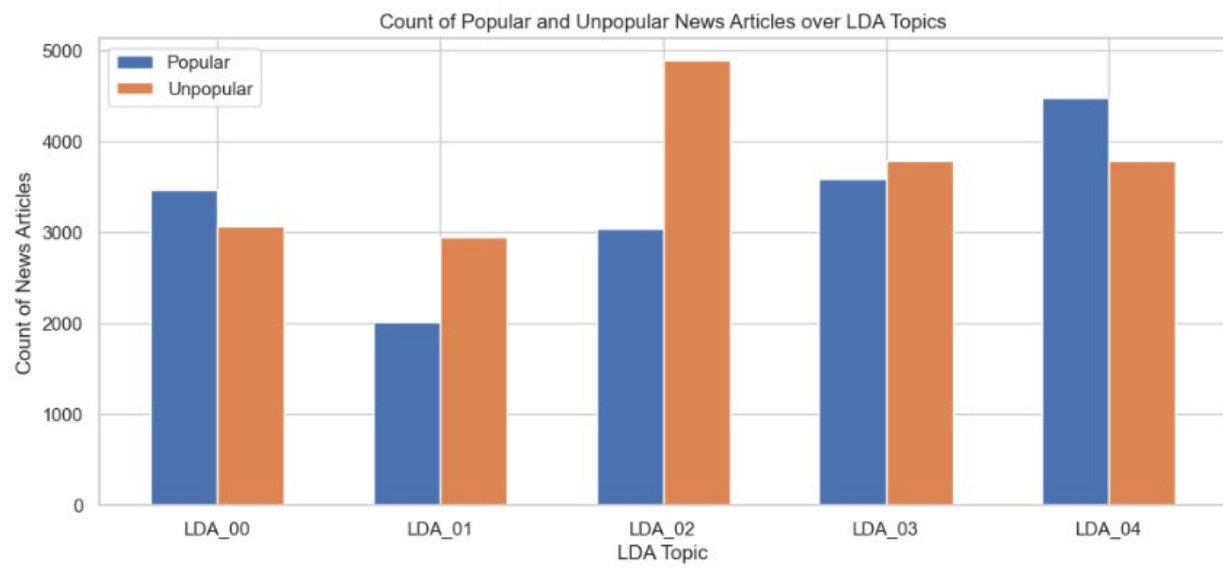
According to the graph below, the weekend is a slow news cycle for both popular and unpopular articles. It is worth noting that articles published in a weekend are more likely to be popular than not.



Evident in the graph below, Tech, Business, and Entertainment dominate both popular and unpopular articles. Volume wise, the difference between the count of popular and unpopular articles in the World and Entertainment is noteworthy.



Similarly, LDA Topic #2 shows the same imbalance between popular and unpopular articles.



Conducting an T-test revealed more significant differences between the popular and unpopular groups than insignificant ones. In this case, `n_non_stop_words`, `kw_min_max`, `kw_avg_max`, `min_negative_polarity`, `max_negative_polarity`, and `abs_title_subjectivity` have insignificant differences in samples.

```
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
1.244140676190898e-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
```

Confirming the visual inspection earlier, Shapiro test indicate that none of the variables are distributed normally.

```
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
```

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

The exploration of the dataset involved both visual exploration and statistical testing. An advantage of visualization is the ease it provides the reader to grasp the characteristics of the data that is being inspected. In addition, the output of the statistical testing eliminates guesswork by supplying a statistic and p-values. One disadvantage is that visual inspection can only go so far. It does not provide a value up front without extensive coding of matplotlib methods.

These are the steps involved in the modeling part of the analysis:

1. Splitting the dataset into training and test sets
2. Building logistic regression models for reference
3. Building XGBoost classifier models
4. Extracting feature importance based on the best XGBoost model

Let's split the data.

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

# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=493)
```

Logistics Regression Models

```
logreg0 = LogisticRegression()
logreg0.fit(X_train, y_train)
```

► LogisticRegression

```
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
```

```
y_pred = logreg0.predict(X_test)
print(classification_report(y_test, y_pred))
```

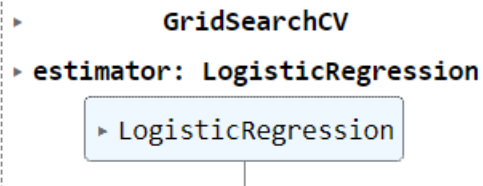
	precision	recall	f1-score	support
0	0.60	0.84	0.70	5952
1	0.56	0.26	0.35	4579
accuracy			0.59	10531
macro avg	0.58	0.55	0.52	10531
weighted avg	0.58	0.59	0.55	10531

After splitting the data and building the initial logistic regression model, GridSearchCV was utilized to determine the ideal parameters that maximize the accuracy of the logistic regression model.

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

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)
```



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

Tuned Hyperparameters : {'C': 10.0, 'penalty': 'l1', 'solver': 'liblinear'}
Logistic Regression) Accuracy : 0.6561539190098994
```

In the following code, the final logistic regression model is built using the best parameters given by grid search. The accuracy of the final logistic regression model is 0.65, a value that is relatively close to 0.64 which is the accuracy of the training set. This means that the model generalizes well.

```
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)))
```

LogisticRegression

LogisticRegression(C=10, penalty='l1', solver='liblinear')

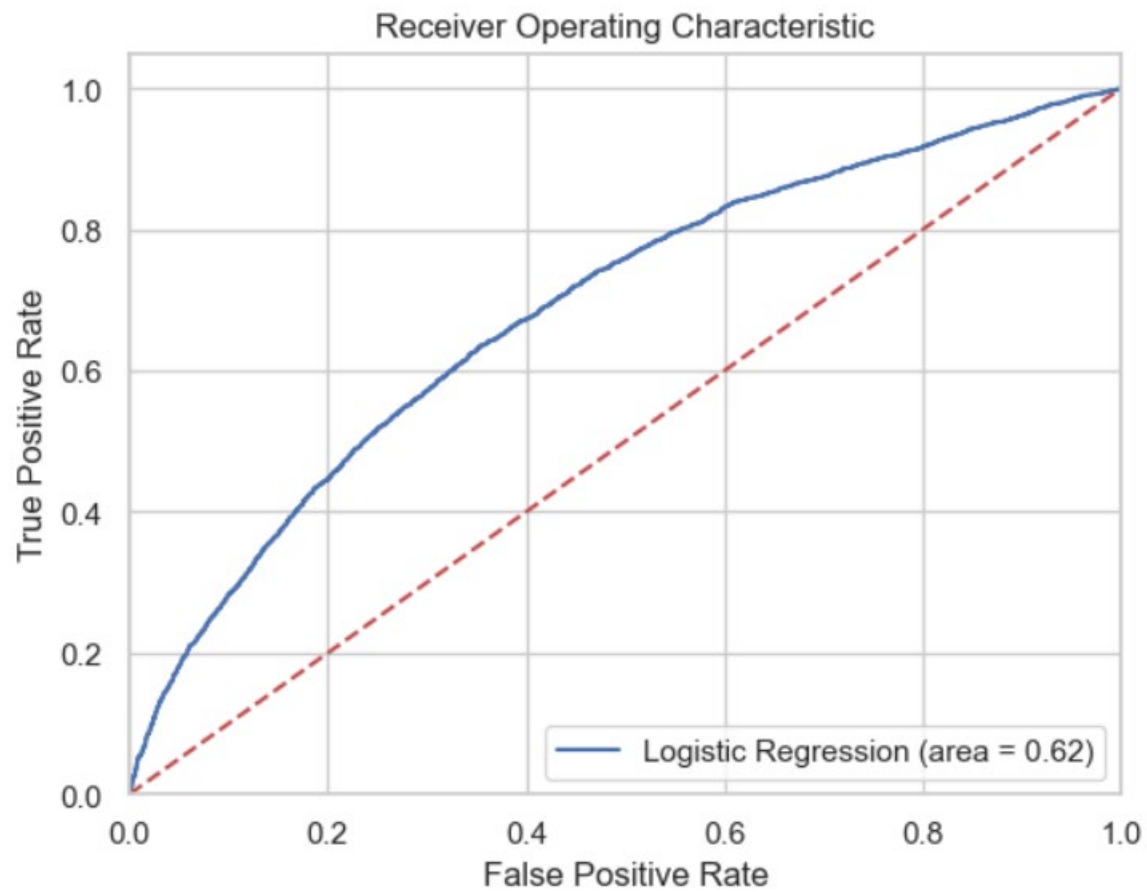
Accuracy of logistic regression classifier on train set: 0.66
Accuracy of logistic regression classifier on test set: 0.65

```
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	0.45	0.53	4579
accuracy			0.65	10531
macro avg	0.64	0.62	0.62	10531
weighted avg	0.64	0.65	0.64	10531

Below, the area-under-the-curve (AUC) score was calculated using the test set and the receiver operating characteristic ROC was plotted. This graph will be used later to compare against the final XGBoost model.

```
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")
```



The ROC curve is larger than the unskilled (0.50) line which signifies that the model is slightly a little bit better at predicting an event than flipping a standard coin.

The initial XGBoost model that was trained had an accuracy of 0.90 on the training set and 0.64 on the test set. The initial model did not generalize well and it well perhaps overfitted.

Xgboost Models

```
# initial XGBOOST model
xgb0 = XGBClassifier(tree_method = 'gpu_hist')
xgb0.fit(X_train, y_train)
```

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytrees=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,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=None, max_leaves=None,
               min_child_weight=None, missing=None, monotone_constraints=None,
               n_estimators=100, n_jobs=None, num_parallel_tree=None,
               predictor=None, random_state=None, ...)
```

```
print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb0.score(X_train, y_train)))
print('Accuracy of xgboost regression classifier on test set: {:.2f}'.format(xgb0.score(X_test, y_test)))
```

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

```
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

To remedy overfitting, we created a pipeline that utilized grid search again to determine the best parameter for XGBoost.

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

# 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],
    }
]

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

# 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

This time around, the accuracy is 0.69 on the train set and 0.67 on the test set. Although the accuracy is not stellar, the model generalized well and did not overfit.

```
print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb1.score(X_train, y_train)))
print('Accuracy of xgboost regression classifier on test set: {:.2f}'.format(xgb1.score(X_test, y_test)))

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

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

```
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 0x00000275C3F0C040>}
```

The best parameters were used in the final XGBoost model.

```
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)
```

```
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=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=4, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=200, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

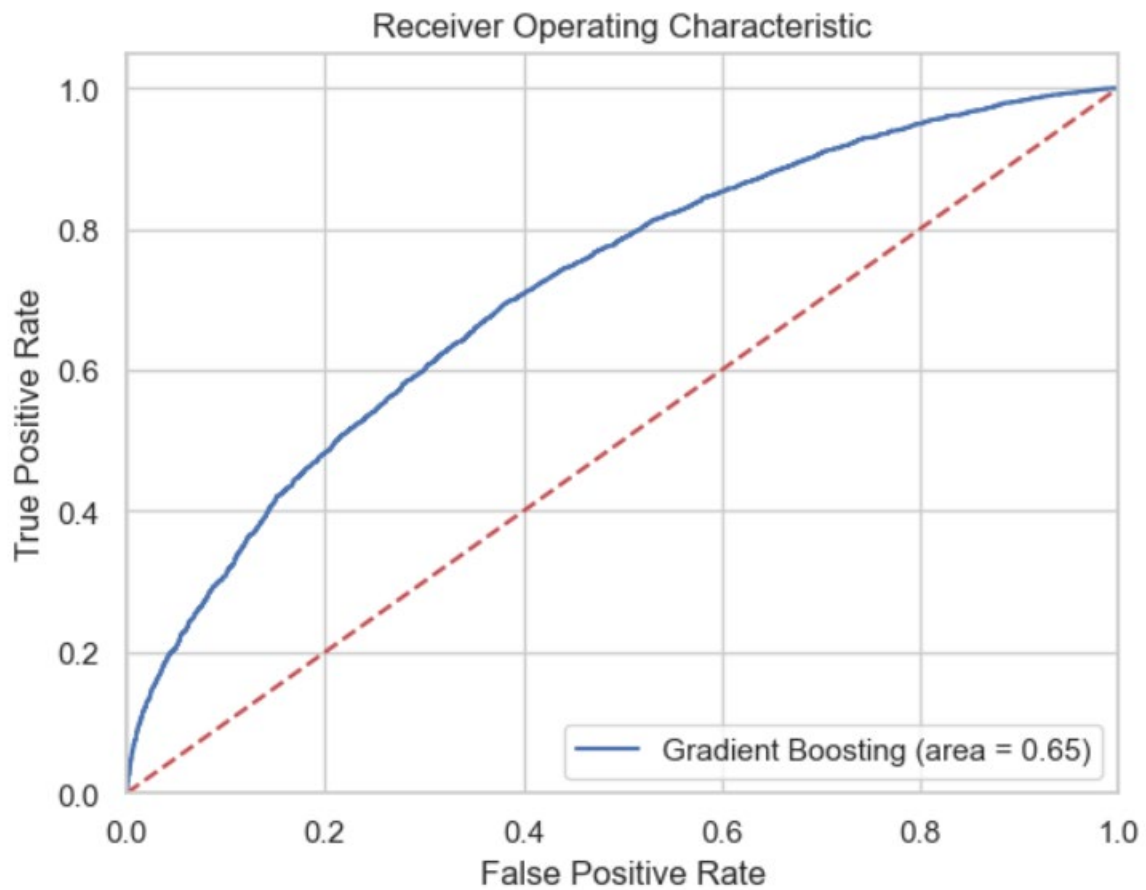
```
print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb2.score(X_train, y_train)))
print('Accuracy of xgboost regression classifier on test set: {:.2f}'.format(xgb2.score(X_test, y_test)))
```

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

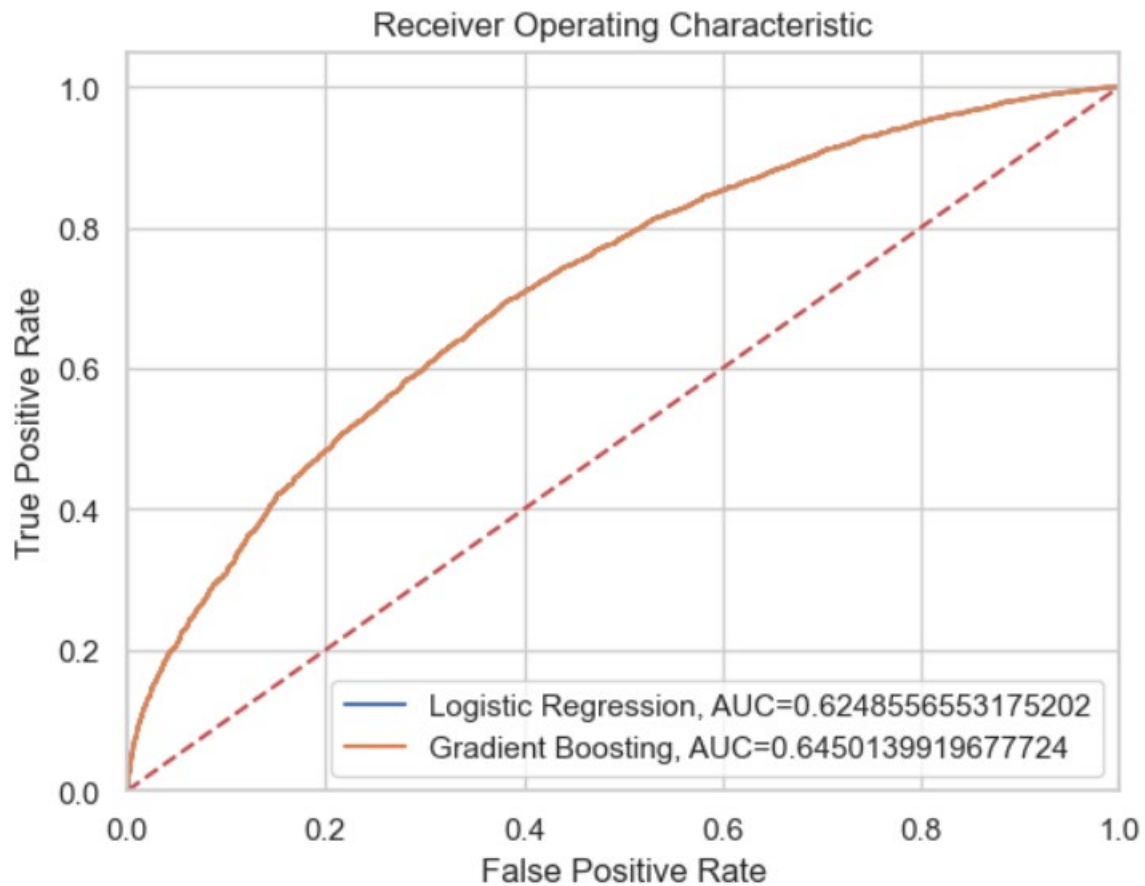
```
y_pred = xgb2.predict(X_test)
print(classification_report(y_test, y_pred))
```

	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

The final XGBoost model sports an accuracy of 0.73 on the train set and 0.66 on the test set. The AUC was calculated and ROC plotted.



When the best logistic regression model and the best XGBoost were plotted together, there was no indication that the two models were different at all. However, the AUC says otherwise. Nevertheless, XGBoost saw an improvement over logistic regression by one point in accuracy and two points in AUC score.

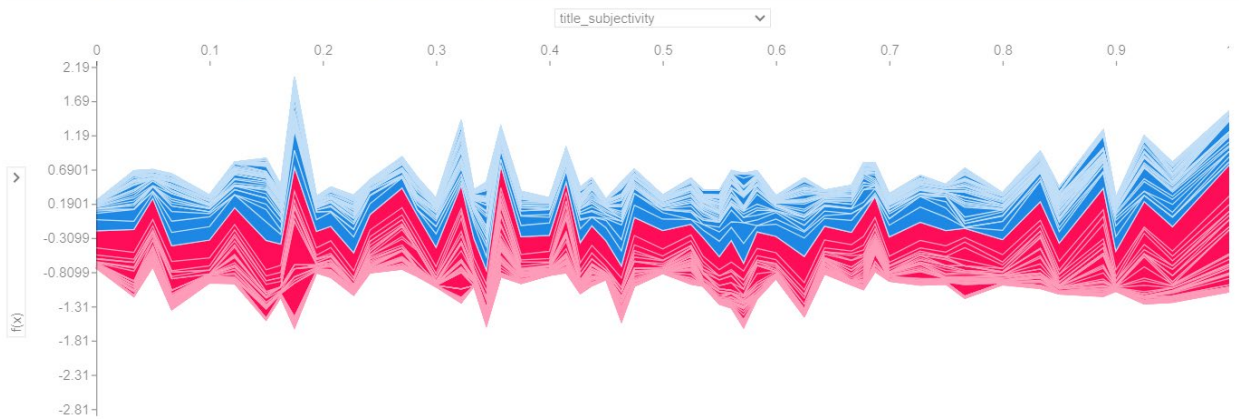


Now, feature importance will be extracted to determine the factors that contribute to an online news article's popularity. For this purpose, the author decided to use the SHAP package to extract feature importance because those ranked by XGBoost were inconsistent. In "Interpretable Machine Learning with XGBoost," Lundberg found that "feature importance orderings are very different for each of the three options provided by XGBoost" (2018). Therefore, SHAP method will be used to extract feature importance.

Feature Importance

```
explainer = shap.TreeExplainer(xgb2)
shap_values = explainer.shap_values(X)
```

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

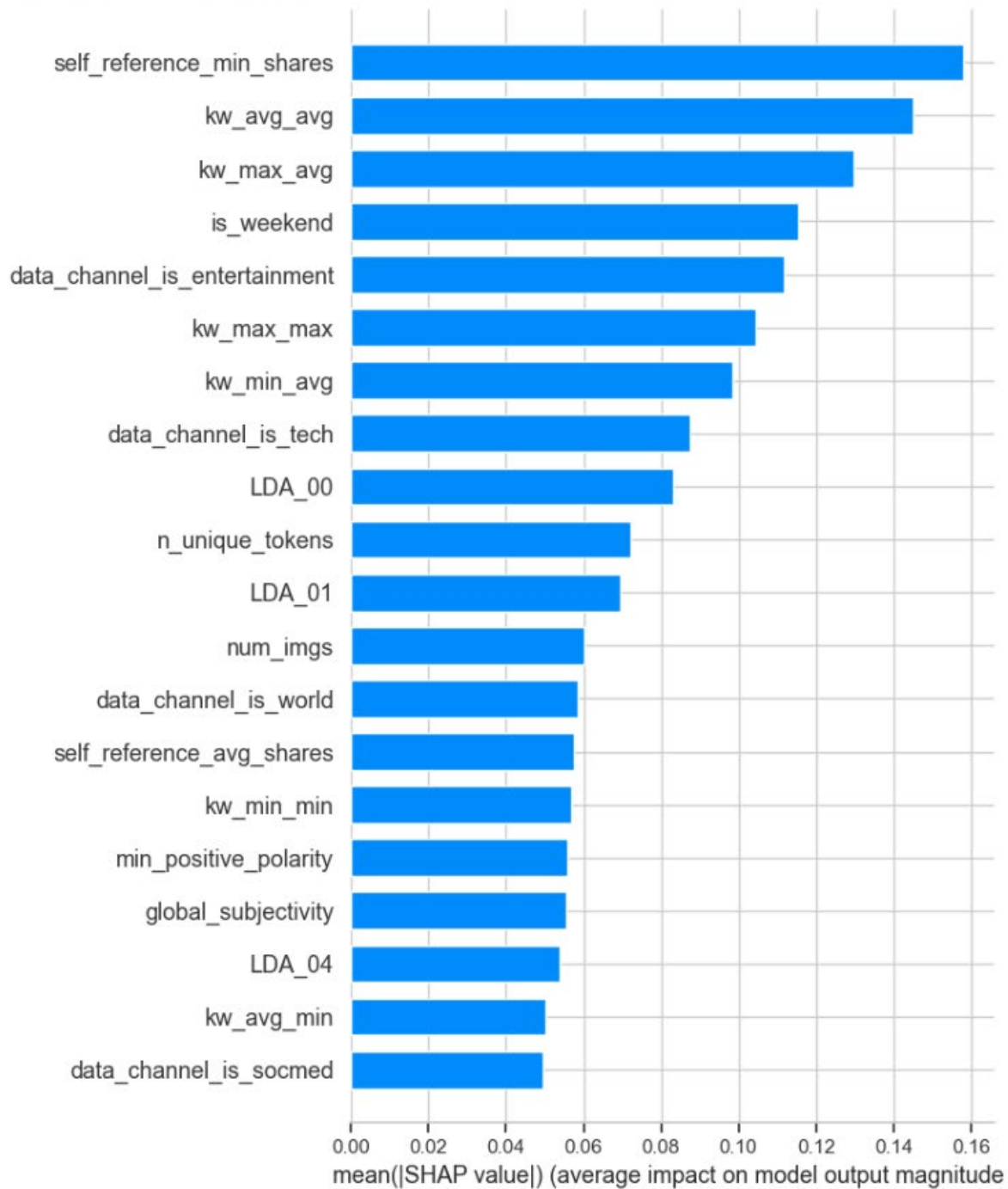


Above, we use the shap package to extract feature importance. Below, the shap package helps in extracting the most important features of our dataset. In relation to the final XGBoost model, the most important features are:

1. self_reference_min_shares
2. kw_avg_avg
3. kw_max_avg
4. is_weekend
5. data_channel_is_entertainment

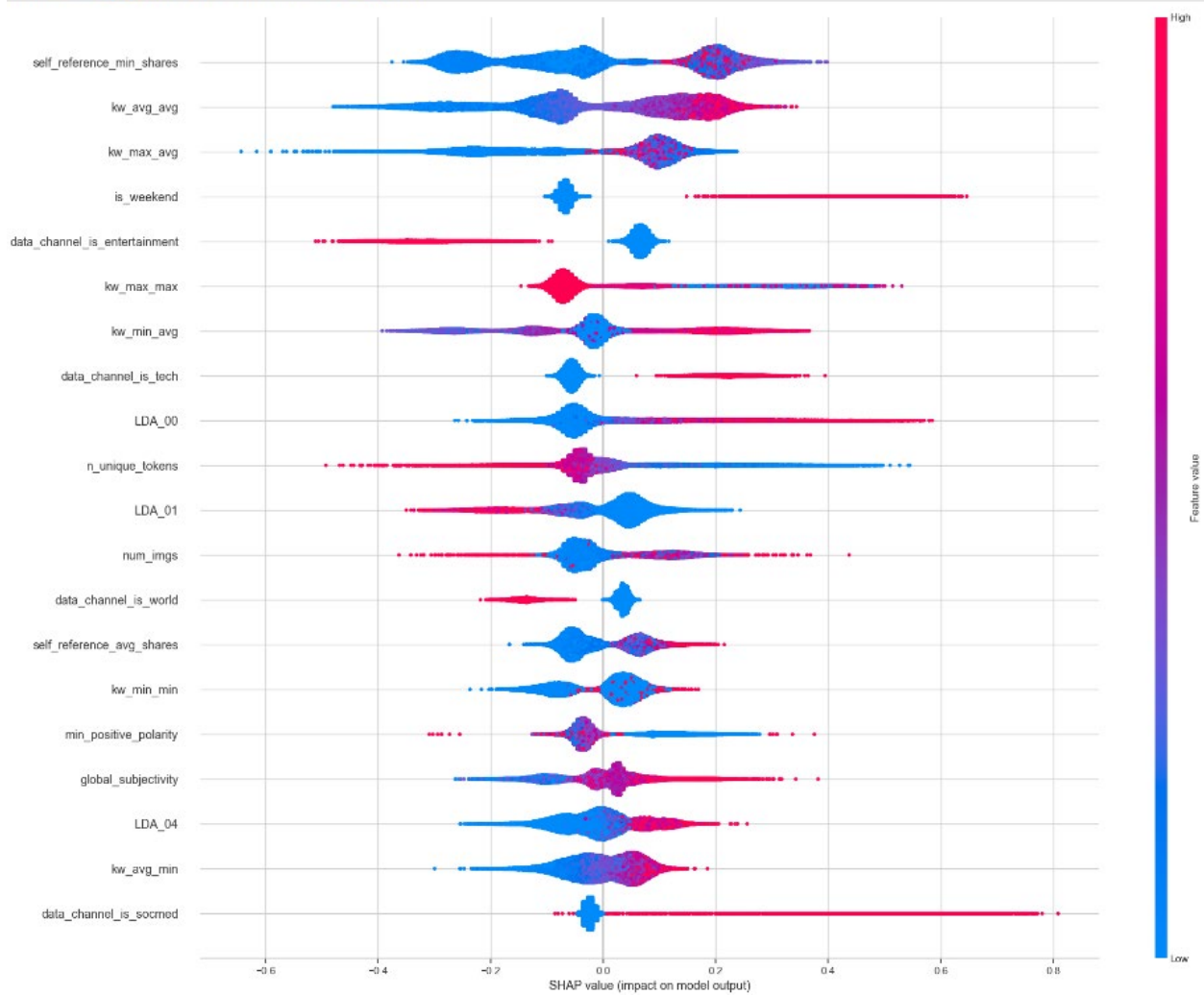
```
plt.figure(figsize = (20,20))  
shap.summary_plot(shap_values, X_train, plot_type="bar",)
```

<Figure size 2000x2000 with 0 Axes>



In the graph below, every article has one dot on each row. The x position of the dot is the impact of that feature on the model's prediction for the article, and the color of the dot represents the value of that feature for the article. Dots that do not fit on the row pile up to show density (Lundberg, 2018).

```
shap.summary_plot(shap_values, X, plot_size=(20,15))
```



E – Data Summary and Implications

The research question of this study is “Do certain factors affect the popularity of online news articles?” Based on the analysis, it is clear that certain factors affect the popularity of online news articles. Some variables seem to have a stronger influence on the target variable when it comes to statistical significance.

One limitation of this study is that the original dataset only included articles from one website (Mashable.com). The prevalence of popular articles in tech, business, and entertainment reflects the niched demographic of Mashable’s distribution. A sample containing articles from all sorts of publications would make a better dataset that could generalize better in predicting previously unseen articles.

Based on this study, one recommendation that can be made is to pay particular attention to the kind of articles that are published during the weekend. Even though the number of articles published on the weekend is lesser than those published during the week, the study shows that articles published during the weekend are more likely to be popular than not. This phenomenon surely warrants more investigation to determine the reason why.

The author of this study submits the following recommendation course of actions for future studies:

- Use of XGBoost regression to predict the number of social media shares instead of using a threshold value
- Clustering the articles based on channels or topic

Initially converting the number of social media shares into the categorical number of 0 or 1 presented the possibility of information loss. Predicting the number of social media shares

could possibly yield better results. In addition, leveraging clustering algorithms could also yield more insights about the different segments within the publication's reader base.

F – Sources

- Chen, Tianqi. (n.d.). Tianqi Chen. Retrieved August 31, 2023, from <https://tqchen.com>.
- Fernandes, Kelwin, Vinagre, Pedro, Cortez, Paulo, and Sernadela, Pedro. (2015). Online News Popularity. UCI Machine Learning Repository. <https://doi.org/10.24432/C5NS3V>
- How XGBoost Works. (n.d.). Amazon Sagemaker. Retrieved August 31, 2023, from <https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html>
- Lundberg, Scott. (2018, April 17). Interpretable Machine Learning with XGBoost. Towards Data Science. Retrieved from <https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27>.
- Mueller, Alex. (2018, March 24). 5 reasons why jupyter notebooks suck. Towards Data Science. Retrieved September 1, 2023, from <https://towardsdatascience.com/5-reasons-why-jupyter-notebooks-suck-4dc201e27086>
- Sugino, Michio. (2022, October 20). XGBoost: its Genealogy, its Architectural Features, and its Innovation. Towards Data Science. Retrieved August 31, 2023, from <https://towardsdatascience.com/xgboost-its-genealogy-its-architectural-features-and-its-innovation-bf32b15b45d2>.
- Uddin, Md. Taufeeq (2018). Predicting the Popularity of Online News from Content Metadata. Retrieved August 29, 2023, from <https://github.com/krishnakartik1/onlineNewsPopularity/blob/master/Paper2/Predicting%20the%20Popularity%20of%20Online%20News%20from%20Content%20Metadata.pdf>

G – Appendix

```
# 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()

THRESHOLD = 1400
ALPHA = 0.05

# Read a csv file
df = pd.read_csv('../data/in/OnlineNewsPopularity.csv')

df.head()
df.info()
df.shape

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_percentage,
                    'num_empty': empty_count, 'empty_percentage': empty_percentage,
                    'nan_count': nan_count, 'nan_percentage': nan_percentage})

return dfx

show_missing(df)

df.columns

for col in df.columns:
    df = df.rename(columns={col:(col.strip(' '))})

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

df.shape
df = df.drop_duplicates(keep = False)
df.shape

# 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'])

df.to_csv('../data/out/online_news_popularity_clean.csv', index=False)

df.head()
df.info()
df.describe()

sns.countplot(x='target', data=df)

token_cols = ['n_tokens_title', 'n_tokens_content', 'n_unique_tokens', 'n_non_stop_words',
              'n_non_stop_unique_tokens', 'average_token_length', 'num_keywords']
links_cols = ['num_hrefs', 'num_self_hrefs']

```

```

media_cols = ['num_imgs', 'num_videos']

channel_cols = ['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_cols = ['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_ref_cols = ['self_reference_min_shares', 'self_reference_max_shares',
'self_reference_avg_shares']

week_cols = ['weekday_is_monday', 'weekday_is_tuesday', 'weekday_is_wednesday',
'weekday_is_thursday', 'weekday_is_friday', 'weekday_is_saturday', 'weekday_is_sunday']

topic_cols = ['LDA_00', 'LDA_01', 'LDA_02', 'LDA_03', 'LDA_04']

global_cols = ['global_subjectivity', 'global_sentiment_polarity', 'global_rate_positive_words',
'global_rate_negative_words']

local_cols = ['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_cols = ['title_subjectivity', 'title_sentiment_polarity', 'abs_title_subjectivity',
'abs_title_sentiment_polarity']

all_columns = ['token_cols', 'links_cols', 'media_cols', 'channel_cols', 'kw_cols',
'self_ref_cols',
                'weekday_cols', 'weekend_cols', 'topic_cols', 'global_cols', 'local_cols',
'title_cols']

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

viz_box(df, 'shares')

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))

viz_box(df, 'shares')

continuous_cols = token_cols + links_cols + kw_cols + self_ref_cols + global_cols + local_cols +
title_cols

for col in continuous_cols:
    viz_box(df, col)

unpopular_df = df[df['shares'] < THRESHOLD ]
popular_df = df[df['shares'] >= THRESHOLD ]

popular_df[continuous_cols].hist(figsize=(20,20))
plt.show()

unpopular_df[continuous_cols].hist(figsize=(20,20))
plt.show()

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

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

```
plt.show()

# Adapted from
# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-coordinates-side-
# by-side-dodged

# 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', 'Saturday',
'Sunday'))

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

# Adapted from
# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-coordinates-side-by-side-dodged

# 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', 'Saturday',
'Sunday'))

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

# Adapted from
# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-coordinates-side-by-side-dodged

# 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 Media', 'Tech',
'World'))

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

# Adapted from
# https://stackoverflow.com/questions/10369681/how-to-plot-bar-graphs-with-same-x-coordinates-side-by-side-dodged

# 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()

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)

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)

final_df = df.drop(columns=['shares'])

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

# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=493)

logreg0 = LogisticRegression()
logreg0.fit(X_train, y_train)

print('Accuracy of logistic regression on train set: {:.2f}'.format(logreg0.score(X_train,
y_train)))
```

```
print('Accuracy of logistic regression on test set: {:.2f}'.format(logreg0.score(X_test, y_test)))
```

```
y_pred = logreg0.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
# parameter grid
```

```
parameters = {
```

```
    'penalty' : ['l1', 'l2'],
```

```
    'C'       : np.logspace(-3, 3, 7),
```

```
    'solver'  : ['newton-cg', 'lbfgs', 'liblinear'],
```

```
}
```

```
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)
```

```
print("Tuned Hyperparameters :", clf.best_params_)
```

```
print("Logistic Regression) Accuracy :", clf.best_score_)
```

```
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)))
```

```
y_pred = logreg2.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
```

```
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")

# initial XGBOOST model
xgb0 = XGBClassifier(tree_method = 'gpu_hist')
xgb0.fit(X_train, y_train)

print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb0.score(X_train, y_train)))
print('Accuracy of xgboost classifier classifier on test set: {:.2f}'.format(xgb0.score(X_test,
y_test)))

y_pred = xgb0.predict(X_test)
print(classification_report(y_test, y_pred))

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

# 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],
}
]

# Define cross validation
kfold = KFold(n_splits=10)

# AUC and accuracy as score
scoring = {'AUC': 'roc_auc', 'Accuracy': make_scorer(accuracy_score)}

# 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)

print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb1.score(X_train, y_train)))
print('Accuracy of xgboost regression classifier on test set: {:.2f}'.format(xgb1.score(X_test,
y_test)))

y_pred = xgb1.predict(X_test)
print(classification_report(y_test, y_pred))

print(xgb1.best_params_)
```



```

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)

print('Accuracy of xgboost classifier on train set: {:.2f}'.format(xgb2.score(X_train, y_train)))
print('Accuracy of xgboost classifier classifier on test set: {:.2f}'.format(xgb2.score(X_test,
y_test)))

y_pred = xgb2.predict(X_test)
print(classification_report(y_test, y_pred))

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")

#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")

explainer = shap.TreeExplainer(xgb2)
shap_values = explainer.shap_values(X)

shap.force_plot(explainer.expected_value, shap_values[:1000,:], X_train.iloc[:1000,:])

plt.figure(figsize = (20,20))
shap.summary_plot(shap_values, X_train, plot_type="bar",)

shap.summary_plot(shap_values, X, plot_size=(20,20))

for col in X_train.columns:
    shap.dependence_plot(col, shap_values, X)

print('Successful run!')
```