Pattern Recognition

Project Report

Team 5

2176045 Kim Doyoon

2176222 Woo Minha

2491007 Kim Yeeun

2491025 Lee Jiwon

Index

1. Data Exploration

1.1. Purpose

The purpose of this EDA is to numerically identify the distribution of all variables, the presence of missing values, relationships between variables, and the association with the target variable (y) based on the train.csv data. Through this, the goal was to establish a foundation for selecting meaningful variables in subsequent modeling processes and to preemptively recognize unnecessary variables or those with multicollinearity issues.

1.2. Data Visualization

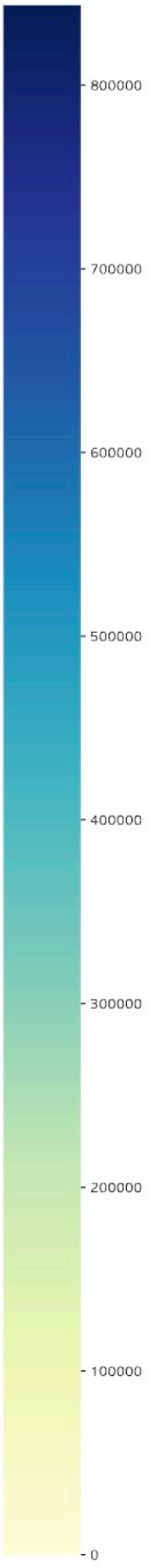
1.2.1. Statistics of All Variables

Using the describe() function, the basic statistics of all variables were checked. For numerical variables, the mean, median, standard deviation, minimum, maximum, and quartile values were calculated. Each categorical variable takes a value of 0 or 1, and the distribution of each category was checked through frequency counts.

Through this, the overall scale and distribution of the data could be understood, serving as a basis for determining variable selection and processing methods in subsequent preprocessing and modeling stages.

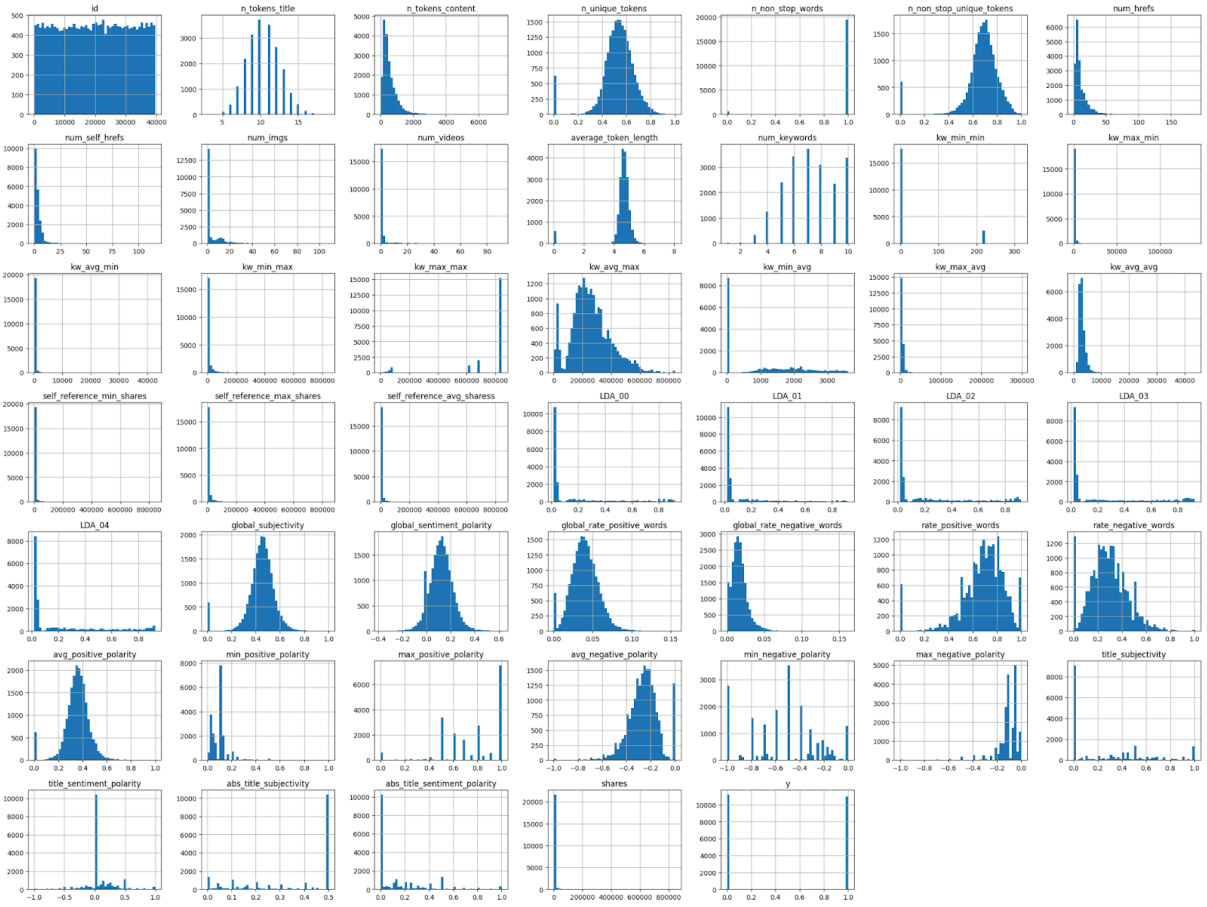
수치형 변수 기초 통계 요약

	count	mean	std	min	25%	median	75%	max
n_tokens_title	20000.00	10.39	2.11	3.00	9.00	10.00	12.00	19.00
n_tokens_content	20054.00	543.92	460.69	0.00	246.00	409.00	715.75	7413.00
n_unique_tokens	19942.00	0.53	0.14	0.00	0.47	0.54	0.61	1.00
n_non_stop_words	20017.00	0.97	0.17	0.00	1.00	1.00	1.00	1.00
n_non_stop_unique_tokens	19944.00	0.67	0.16	0.00	0.63	0.69	0.75	1.00
num_hrefs	19967.00	10.84	11.02	0.00	4.00	7.00	14.00	186.00
num_self_hrefs	19920.00	3.30	3.94	0.00	1.00	3.00	4.00	116.00
num_imgs	19986.00	4.47	8.09	0.00	1.00	1.00	4.00	108.00
num_videos	19906.00	1.25	4.09	0.00	0.00	0.00	1.00	91.00
average_token_length	19987.00	4.55	0.84	0.00	4.48	4.67	4.86	8.04
num_keywords	19955.00	7.21	1.91	1.00	6.00	7.00	9.00	10.00
kw_min_min	19984.00	26.42	69.96	-1.00	-1.00	-1.00	4.00	318.00
kw_max_min	20021.00	1132.88	3228.51	0.00	445.00	657.00	1000.00	139600.00
kw_avg_min	19971.00	315.63	679.37	-1.00	142.33	236.20	358.13	42827.86
kw_min_max	19985.00	13360.32	55609.26	0.00	0.00	1400.00	7900.00	843300.00
kw_max_max	19968.00	751711.29	215052.17	0.00	843300.00	843300.00	843300.00	843300.00
kw_avg_max	20040.00	259430.57	135661.35	0.00	172142.38	244130.95	331144.73	843300.00
kw_min_avg	20049.00	1118.02	1139.02	-1.00	0.00	1031.40	2059.98	3610.12
kw_max_avg	20018.00	5678.84	6214.38	0.00	3566.30	4355.57	6015.44	298400.00
kw_avg_avg	20023.00	3139.53	1335.08	0.00	2377.72	2868.34	3604.77	43567.66
self_reference_min_shares	19994.00	3857.64	18332.25	0.00	633.00	1200.00	2600.00	843300.00
self_reference_max_shares	20026.00	9954.07	38562.95	0.00	1100.00	2900.00	7900.00	843300.00
self_reference_avg_sharess	19954.00	6160.07	22153.93	0.00	986.00	2201.21	5200.00	843300.00
LDA_00	19976.00	0.19	0.26	0.02	0.03	0.03	0.24	0.92
LDA_01	19998.00	0.14	0.22	0.02	0.03	0.03	0.15	0.92
LDA_02	20047.00	0.22	0.28	0.02	0.03	0.04	0.34	0.92
LDA_03	19957.00	0.23	0.30	0.02	0.03	0.04	0.38	0.93
LDA_04	20013.00	0.23	0.29	0.02	0.03	0.04	0.39	0.93
global_subjectivity	19946.00	0.44	0.12	0.00	0.40	0.45	0.51	1.00
global_sentiment_polarity	19969.00	0.12	0.10	-0.38	0.06	0.12	0.18	0.62
global_rate_positive_words	19966.00	0.04	0.02	0.00	0.03	0.04	0.05	0.16
global_rate_negative_words	19951.00	0.02	0.01	0.00	0.01	0.02	0.02	0.16
rate_positive_words	19981.00	0.68	0.19	0.00	0.60	0.71	0.80	1.00
rate_negative_words	19958.00	0.29	0.16	0.00	0.19	0.28	0.38	1.00
avg_positive_polarity	20050.00	0.35	0.10	0.00	0.31	0.36	0.41	1.00
min_positive_polarity	19958.00	0.10	0.07	0.00	0.05	0.10	0.10	1.00
max_positive_polarity	20033.00	0.76	0.25	0.00	0.60	0.80	1.00	1.00
avg_negative_polarity	19916.00	-0.26	0.13	-1.00	-0.33	-0.25	-0.19	0.00
min_negative_polarity	19926.00	-0.52	0.29	-1.00	-0.70	-0.50	-0.30	0.00
max_negative_polarity	19997.00	-0.11	0.10	-1.00	-0.12	-0.10	-0.05	0.00
title_subjectivity	19928.00	0.28	0.33	0.00	0.00	0.14	0.50	1.00
title_sentiment_polarity	19967.00	0.07	0.26	-1.00	0.00	0.00	0.14	1.00
abs_title_subjectivity	19934.00	0.34	0.19	0.00	0.17	0.50	0.50	0.50
abs_title_sentiment_polarity	19951.00	0.16	0.23	0.00	0.00	0.00	0.25	1.00



1.2.2. Histograms of Numerical Variables

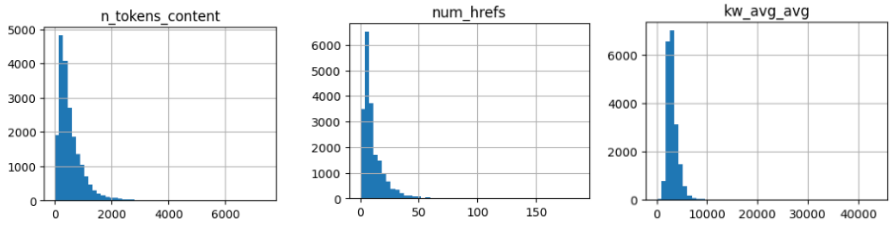
To visually compare the distribution of data before preprocessing, and to check the distribution shape of each variable and the presence of outliers or extreme values, 49 numerical variables such as n_tokens_title, n_tokens_content, etc., were plotted as histograms.



Upon examining the histograms, the distributions of numerical variables can be largely classified into four types:

1) Long-tail distribution

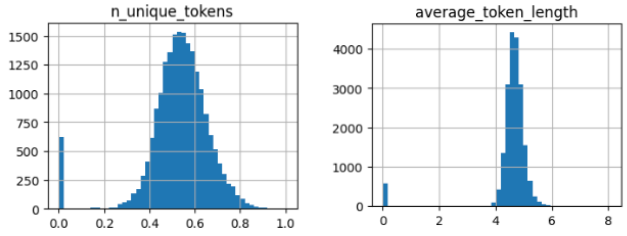
: n_tokens_content, num_hrefs, num_imgs, kw_* series, etc.



Most values are clustered near 0, with a long tail extending towards larger values. It was concluded that log transformation would be advantageous for model stability when handling these variables.

2) Approximate normal distribution

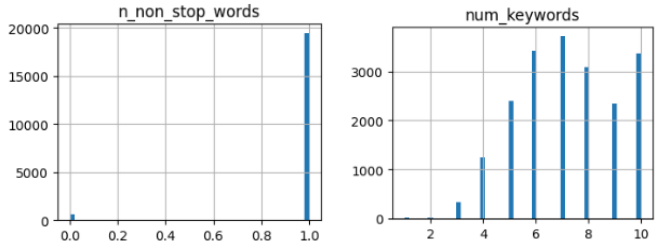
: n_unique_tokens, average_token_length, etc.



These show a bell-shaped normal distribution centered in the middle. Apart from occasional outliers, it was concluded that these variables could be handled without much concern.

3) Discrete value spike (distribution concentrated at specific values)

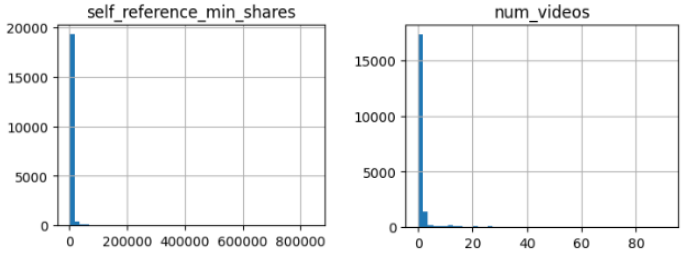
: n_non_stop_words, num_keywords, etc.



These show a distribution where frequencies are concentrated at specific values. After checking the correlation with y, it was concluded that binarization or binning followed by one-hot encoding could be attempted.

4) 0 spike + Outlier distribution

: self_reference_*, num_videos, etc.

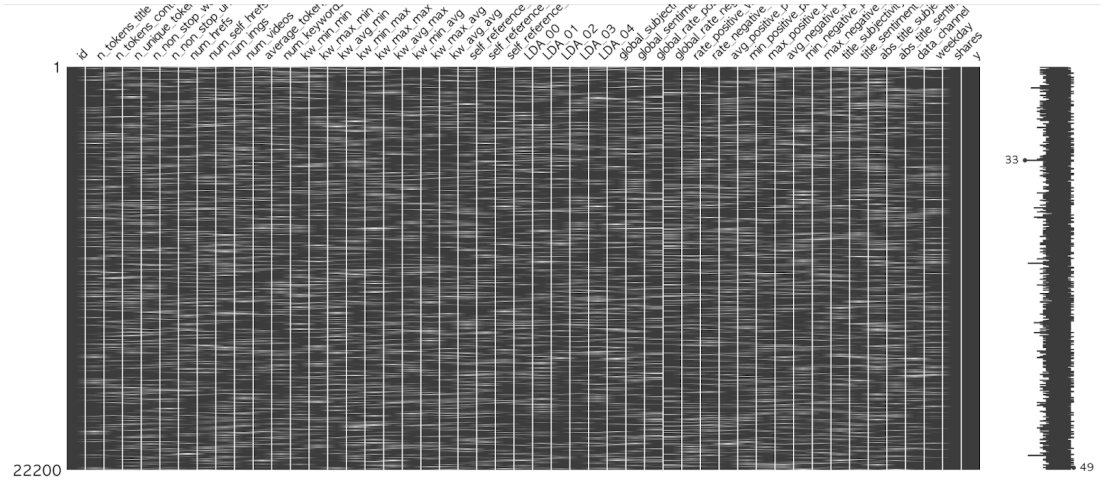


Most of the dataset consists of 0s, with only a few large outlier values. After analyzing correlations between variables, it was concluded that binarizing into 0/non-0 or applying log transformation would be suitable for handling this type of data.

1.2.3. Missing Value Analysis and Visualization

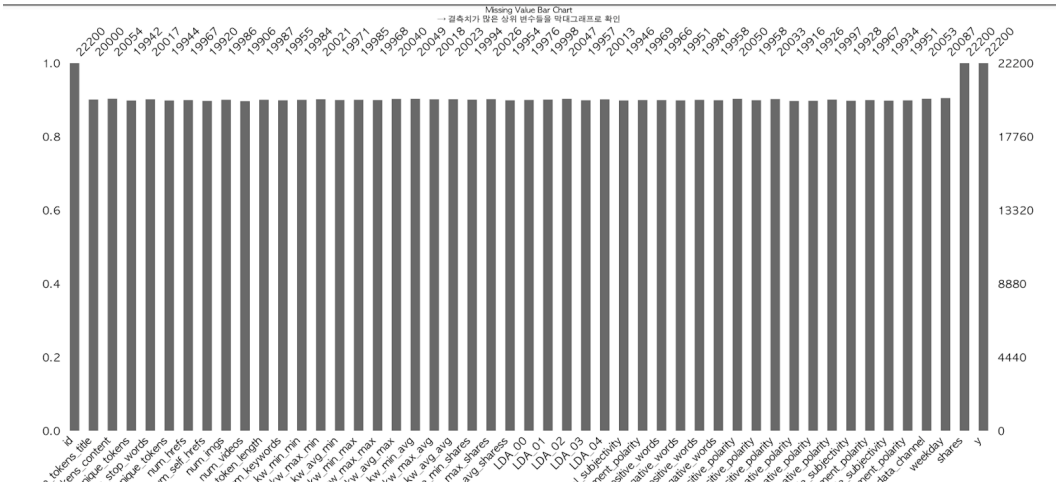
The number and ratio of missing values for all variables were calculated, and the overall distribution of missing values was visualized using the missingno library. The figure below shows the missing pattern across the entire dataset.

1) Missingno Matrix



The missing status of the entire dataset was visualized. In the figure below, horizontal breaks indicate missing values, allowing intuitive identification of which variables have missing values and the extent of their presence. Notably, this visualization showed that there were no identical row-wise missing patterns across multiple variables. This indicates that missing values occurred independently, rather than in relation to each other, suggesting that individual variable-level imputation strategies are more appropriate than joint variable-based imputation.

2) Missingno Bar Chart

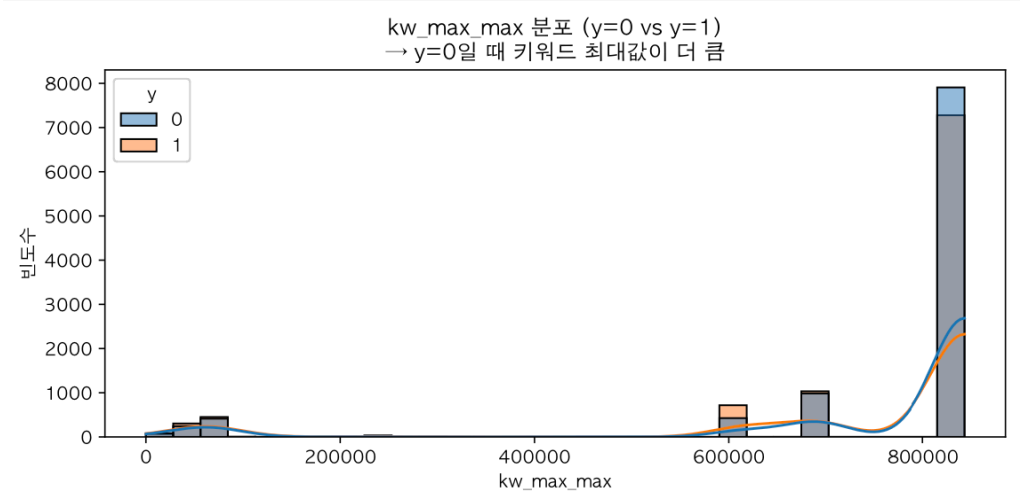


This is a graph visualizing the number of missing values per variable. Overall, missing values are not concentrated in specific variables, and most variables show a similar missing rate of around 10%. This suggests that missing values are evenly distributed throughout the data, indicating that a consistent missing value handling strategy across all variables is needed, rather than targeting only a small number of specific variables.

1.2.4. Visualization of Representative Variables

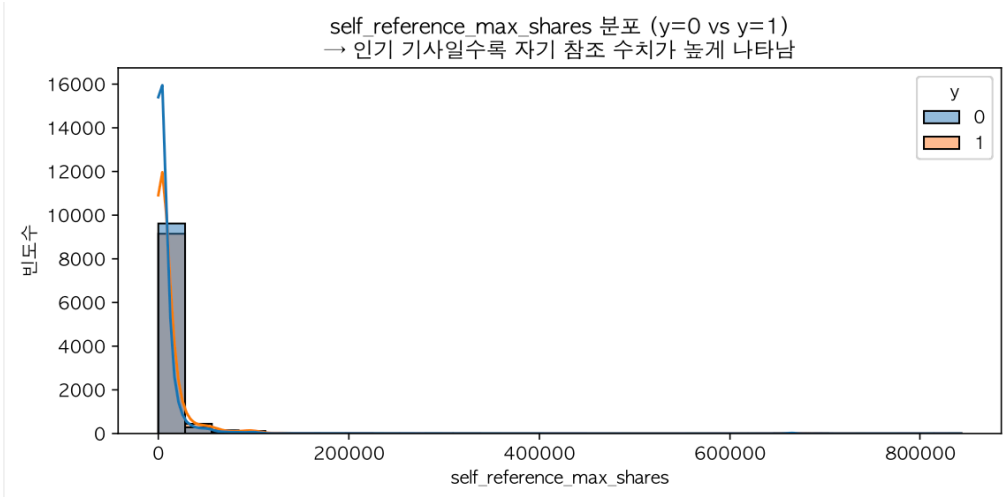
Three representative variables with large average differences between y=0 and y=1 classes were selected, and the distributions of y=0 and y=1 were visualized using histplot(). The KDE curve was also plotted to emphasize density differences.

1) kw_max_max distribution



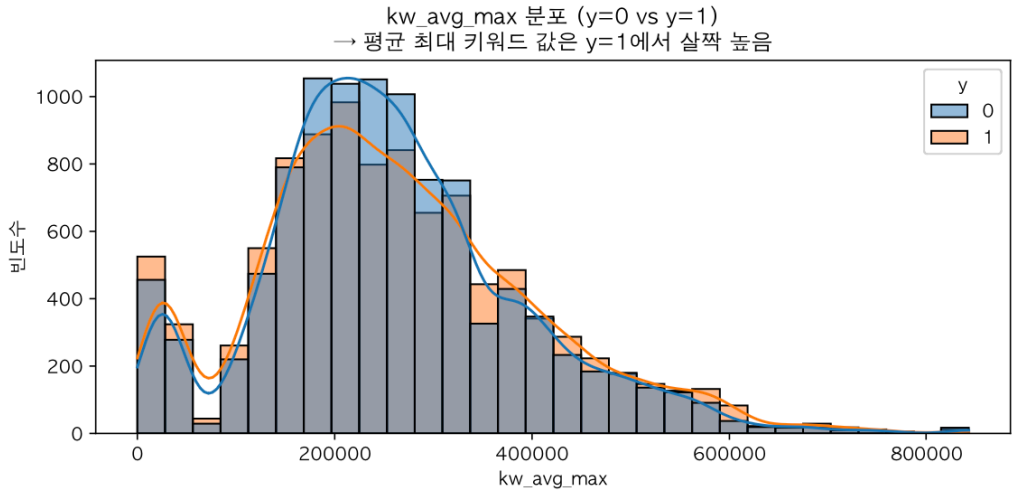
Comparing the distributions for y=0 and y=1, kw_max_max tends to have relatively larger values in y=0. This suggests that even in non-viral articles, there are cases where the frequency of specific keywords is overwhelmingly high. It is an interesting characteristic that the higher the concentration of specific keywords, the more likely the article is classified as non-viral.

2) self_reference_max_shares distribution



For the self-reference share variable, most of the data is concentrated at low values, but the y=1 (viral article) class shows a rare long-tail distribution with high values. This may imply that the more popular an article is, the more inbound links it receives from its own articles.

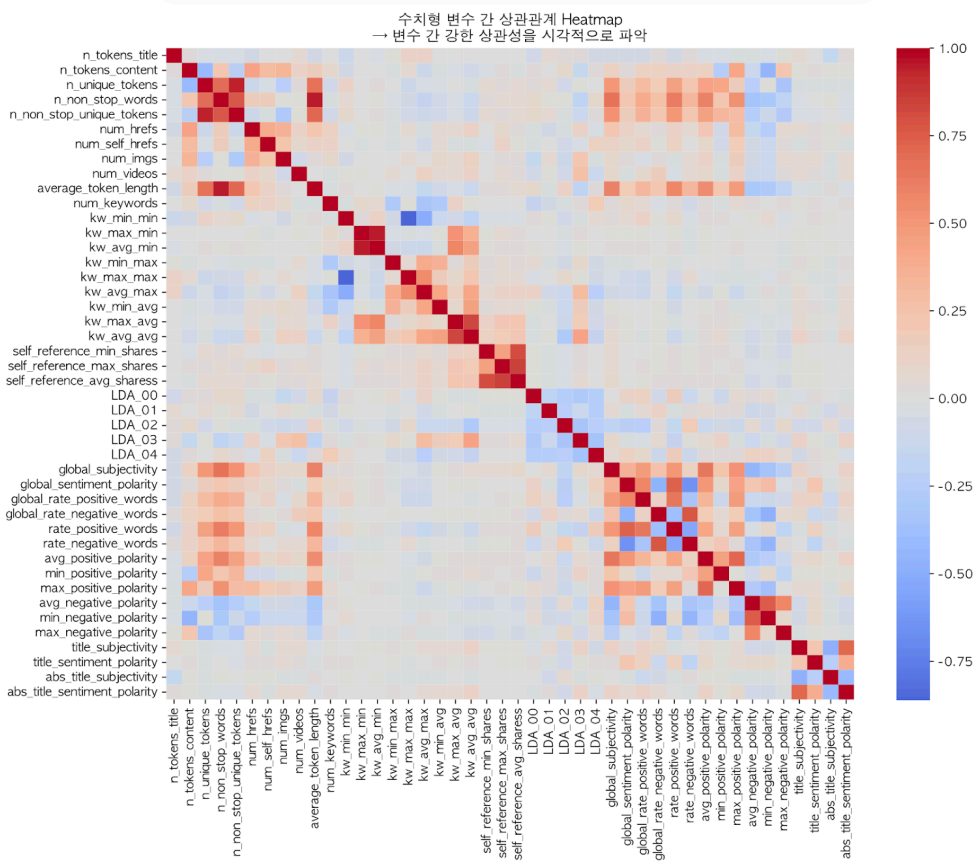
3) kw_avg_max distribution



The average maximum keyword value (kw_avg_max) shows a high peak distribution in y=0, but in y=1, there is a stronger tendency for values to spread into higher ranges. This can be interpreted as reflecting the characteristic of viral articles, where keywords are more widely distributed overall.

1.2.5. Correlation Analysis Between Variables

Pearson correlation coefficients between numerical variables were calculated and visualized as a heatmap.



In this process, it was found that keyword-related variables and self_reference-related variables had high correlations, with some variable pairs showing correlations above 0.9. In later sections, the correlations between variables were analyzed in more detail and used to supplement missing values. This also allowed for the early identification of potential multicollinearity issues during model training.

2. Data Preprocessing

2.1. Data Cleaning

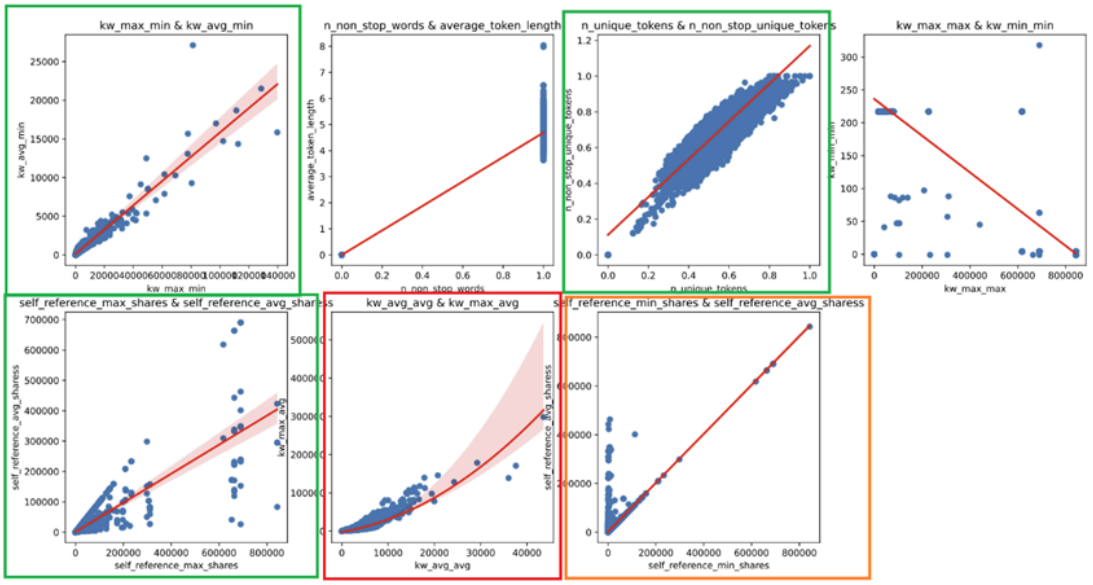
2.1.1. Missing Value Strategy

1) First Analysis

1-1) Numerical Variables

(a) Correlation Analysis

To handle missing values through correlations between numerical variables, pairs of variables with absolute Pearson correlation coefficients greater than 0.8 in the train set were selected and visualized. There were a total of 7 variable pairs with correlation coefficients above 0.8, and upon plotting, 4 pairs showed actual linear or curved relationships. Among them, 3 pairs (variables in the green box in the figure below) showed simple linear relationships, and 1 pair (red box) showed a curved relationship. Based on this analysis, it was concluded that these 4 pairs of variables should have their missing values filled using linear regression, considering their relationships. The self_reference_min_shares & self_reference_avg_shares pair (orange box) showed some degree of linear relationship, but the relationship was weak, and both variables had high skewness (27.712, 18.769, respectively), therefore, it was decided not to use the linear relationship for imputation, and instead, to handle them as described in section (b) below.



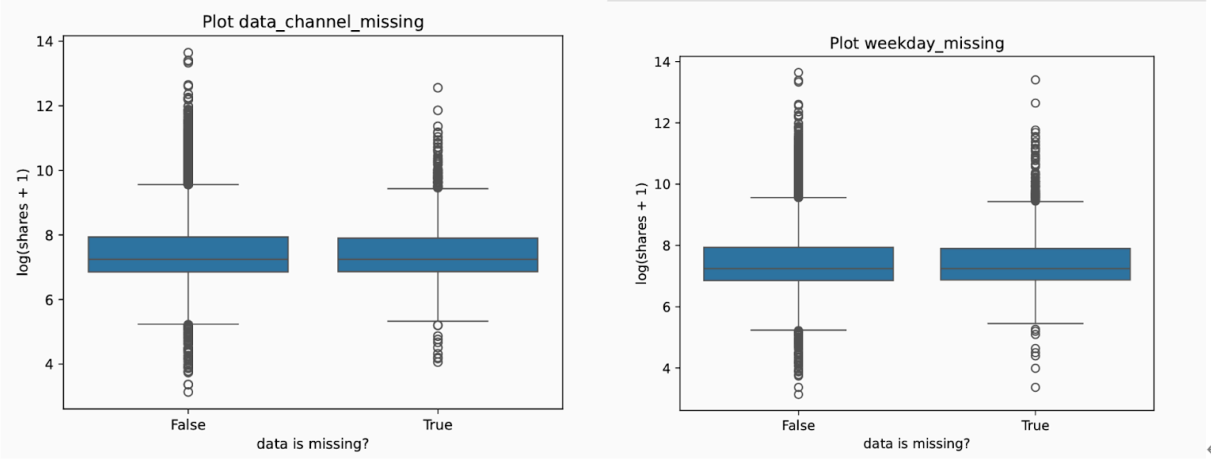
(b) Skewness Analysis

To fill missing values for variables without correlations, skewness was calculated. Also, even for correlated variables, if both variables are missing, the linear relationship cannot be used, so the imputation method was

determined based on the skewness of the variable. If the skewness is less than 1.0 (indicating a distribution relatively close to normal or symmetric), missing values were imputed with the mean. Conversely, if the skewness is 1.0 or higher, the median was used for imputation.

1-2) Categorical Variables

Our dataset contains two categorical variables ('data_channel', 'weekday'). To determine if missingness in categorical variables has any special meaning, the distribution of shares was plotted and compared for samples with and without missing values in each categorical variable. Since there was no difference in the shares distribution between missing and non-missing cases, it was concluded that it is more appropriate to fill missing values with the mode, rather than creating new values to assign meaning to the missingness.

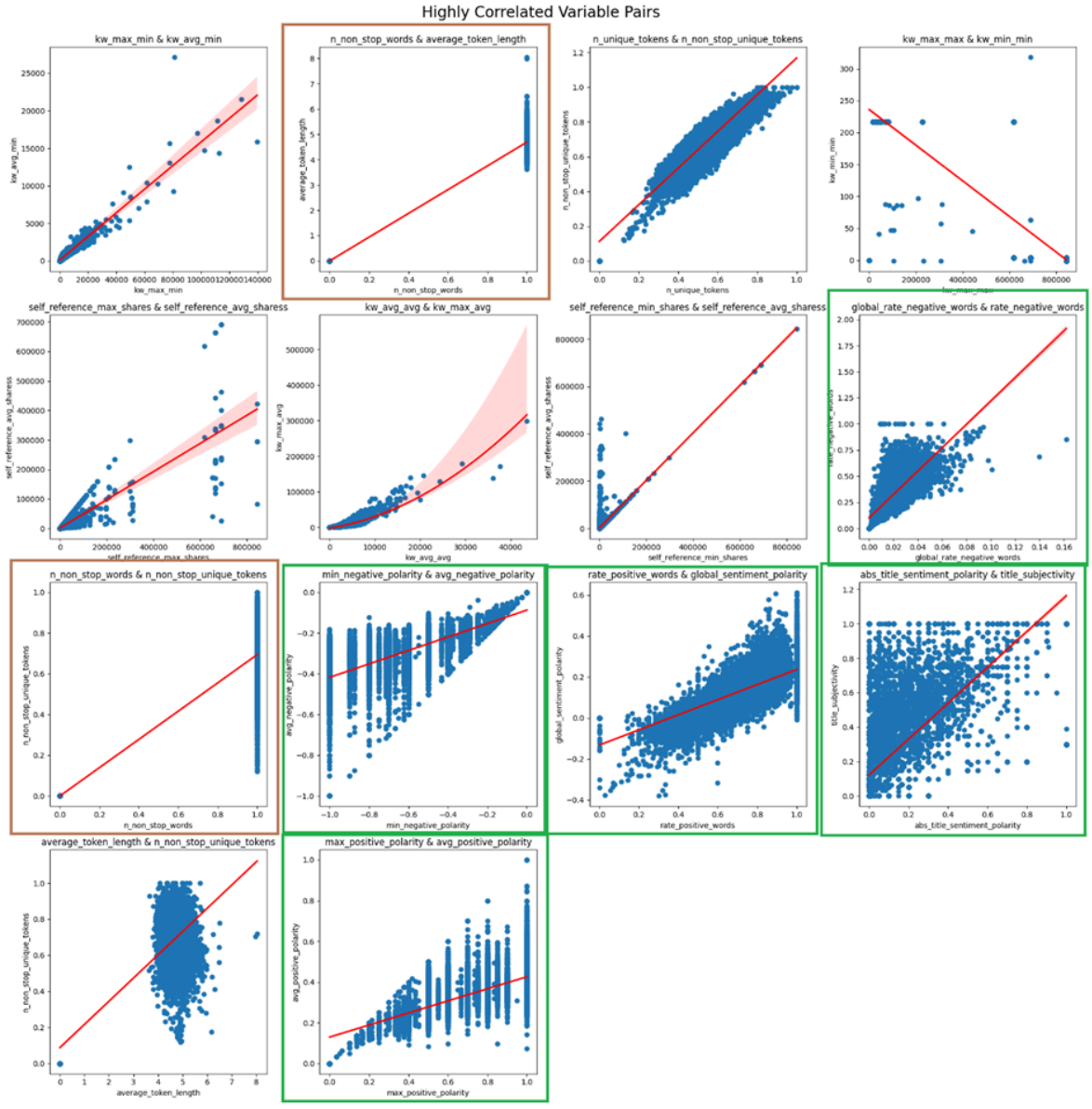


1-3) Rows with Too Many Missing Values

Rows containing too many missing values may not reflect overall data patterns and may hinder model training. Therefore, we decided to delete rows with missing values in 10 or more out of 46 variables (a total of 299 rows). This accounts for about 1% of all samples, so data loss is minimal.

2) Second Analysis

To further improve model performance, another round of data analysis and processing related to missing value imputation was conducted. In this analysis, pairs of variables with correlation coefficients above 0.7 were plotted to explore relationships more closely (previously, only pairs above 0.8 were considered). Lowering the threshold to 0.7 added 7 more highly correlated variable pairs. After analyzing the newly plotted graphs, it was decided to additionally reflect two aspects in missing value imputation, as discussed in the sections below.



2-1) Additional Variable Pairs for Imputation Using Linear Regression

In the new graphs, 5 variable pairs (green box) showed linear relationships, so it was decided to handle the missing values for these pairs using linear regression. Even if the linear relationship was not very strong, it was judged that using regression would be better than naively filling missing values with the mean or median.

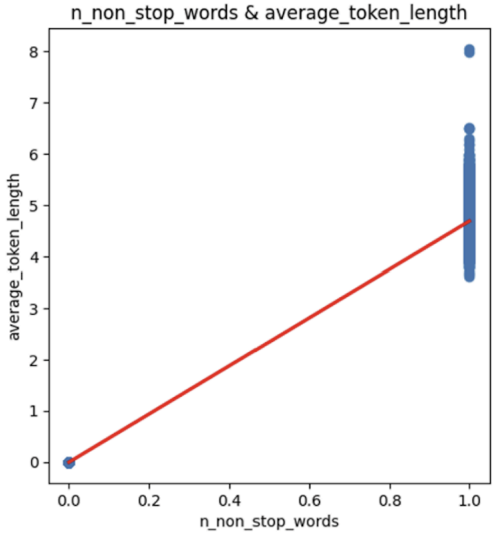
2-2) Considering Relationships Among Three Variables

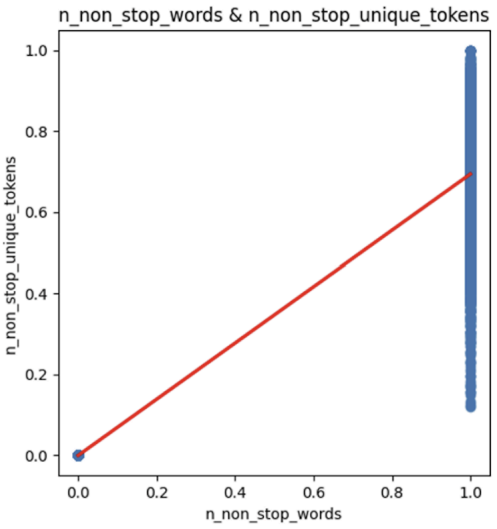
Through two graphs (brown box), the relationships among three variables (n_non_stop_words, n_non_stop_unique_tokens, average_token_length) were identified. As shown in the enlarged graphs below, n_non_stop_words only takes values very close to 0 or 1. Upon checking, Max: 0.999999999746, Median: 0.999999995984, so it can be considered almost a binary variable.

Looking at the relationship between n_non_stop_words and average_token_length (left graph below), when n_non_stop_words is 0, average_token_length is also 0, and when n_non_stop_words is positive, average_token_length takes positive values between about 3 and 8. A similar relationship is found between n_non_stop_words and n_non_stop_unique_tokens (right graph below): when n_non_stop_words is 0, n_non_stop_unique_tokens is also 0,

and only when n_non_stop_words is positive does n_non_stop_unique_tokens take positive values between about 0.1 and 1.0.

Based on these relationships, the following conditional logic can be derived: if n_non_stop_words is 0, the other two variables are also 0. Conversely, if average_token_length or n_non_stop_unique_tokens is greater than 0, n_non_stop_words is a positive value very close to 1. Thus, these three variables have strong conditional dependencies, and when missing values exist, these relationships can be used to predict or supplement the value of one variable from the others.





2.1.2. Process to Fill Missing Values

1) First Round of Missing Value Imputation

Missing values were filled according to the strategy summarized in strategy_df.csv, which was determined from the missing value analysis. First, missing values in variables not designated for regression (i.e., those without correlation coefficients above 0.8 with a paired variable) were imputed using central tendency measures. Subsequently, regression was used to handle the missing values for the variable pairs that were judged suitable for regression (due to having correlation coefficients above 0.8).

1-1) Central Tendency Measures

```
for col, strategy in strategy_map.items():
    if col not in train.columns:
        continue
    if strategy == "mean":
        train[col].fillna(train[col].mean(), inplace=True)
        test[col].fillna(test[col].mean(), inplace=True)
    elif strategy == "median":
        train[col].fillna(train[col].median(), inplace=True)
        test[col].fillna(test[col].median(), inplace=True)
    elif strategy == "mode":
        train[col].fillna(train[col].mode().iloc[0], inplace=True)
        test[col].fillna(test[col].mode().iloc[0], inplace=True)
```

Change in the number of missing values in the train set when only central tendency measures were used

: 95,275 → 37,782

1-2) Regression

To train a regression model capable of predicting a missing value in one variable (A or B) from the other, only rows without missing values for both A and B were initially selected. Given that the specific rows with missing values could differ between variable A and variable B, regression was performed twice: once using A as the independent variable to predict B (dependent), and then vice versa. For rows where both variables were missing and thus could not be imputed via this regression approach, missing values were subsequently filled using the mean or median based on the variable's skewness.

Since missing values in the train and test sets must be handled in the same way, the regression model trained on the train set was used to fill missing values in the corresponding columns of the test set.

a) Linear regression

model_avg: trained to predict var_avg from var_max

model_max: trained to predict var_max from var_avg

```
# 1) 둘 다 결측치 없는 행으로 regression 학습용 데이터 준비
  mask_both_train = train[var_avg].notna() & train[var_max].notna()
  train_reg = train.loc[mask_both_train, [var_max, var_avg]]

# 모델1: var_max → var_avg
  model_avg = LinearRegression()
  model_avg.fit(train_reg[[var_max]], train_reg[var_avg])

# 모델2: var_avg → var_max
  model_max = LinearRegression()
  model_max.fit(train_reg[[var_avg]], train_reg[var_max])
```

The model trained on the train set was used to fill missing values in the test set.

```
# test에도 동일하게 적용
  mask_both_test = test[var_avg].notna() & test[var_max].notna()

# 학습은 train 데이터만 사용
  mask_avg_miss_test = test[var_avg].isna() & test[var_max].notna()
  if mask_avg_miss_test.any():
    Xt_pred_avg = test.loc[mask_avg_miss_test, [var_max]]
    test.loc[mask_avg_miss_test, var_avg] = model_avg.predict(Xt_pred_avg)

  mask_max_miss_test = test[var_max].isna() & test[var_avg].notna()
  if mask_max_miss_test.any():
    Xt_pred_max = test.loc[mask_max_miss_test, [var_avg]]
    test.loc[mask_max_miss_test, var_max] = model_max.predict(Xt_pred_max)
```

b) Polynomial regression

Model form: $kw\_max\_avg = w_1(kw\_avg\_avg) + w_2(kw\_avg\_avg)^2 + b$

```
# 2) 둘 다 값이 있는 행을 골라 학습용 데이터로 선별
mask_train = train["kw_avg_avg"].notna() & train["kw_max_avg"].notna()
df_reg_train = train.loc[mask_train, ["kw_avg_avg", "kw_max_avg"]]

X_train = df_reg_train["kw_avg_avg"]
y_train = df_reg_train["kw_max_avg"]

poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train)

model = LinearRegression()
model.fit(X_train_poly, y_train)

# 3) 하나만 결측치인 경우에 kw_max_avg 채우기
mask_fill_train = train["kw_avg_avg"].notna() & train["kw_max_avg"].isna()
X_pred_train = poly.transform(train.loc[mask_fill_train, ["kw_avg_avg"]])
train.loc[mask_fill_train, "kw_max_avg"] = model.predict(X_pred_train)

mask_fill_test = test["kw_avg_avg"].notna() & test["kw_max_avg"].isna()
X_pred_test = poly.transform(test.loc[mask_fill_test, ["kw_avg_avg"]])
test.loc[mask_fill_test, "kw_max_avg"] = model.predict(X_pred_test)
```

Afterward, polynomial regression was performed again with the positions of kw_avg_avg and kw_max_avg swapped.

c) When both variables are missing

For cases where both variables were missing and could not be filled by regression, missing values were filled using central tendency measures, as described in strategy_df(1).csv. The code is the same as in 1-1).

2) Second Round of Missing Value Imputation

After completing the second round of missing value analysis, the train.csv was processed again, reflecting additional considerations related to missing value handling.

2-1) Missing Value Imputation Considering Relationships Among Three Variables

Imputation considering the relationships among n_non_stop_words, _non_stop_unique_tokens, and average_token_length was carried out in three main steps. First, if any of the three variables was 0, the missing values for the other variables were also filled with 0. In train.csv, there were no cases where one of the three variables was 0 and the others were not, so filling missing values in this way did not cause issues.

```
### Step 1. 세 변수 중 하나라도 0이면 나머지도 0
zero_mask = (
    (df['n_non_stop_words'] == 0) |
    (df['n_non_stop_unique_tokens'] == 0) |
    (df['average_token_length'] == 0)
)
df.loc[zero_mask, ['n_non_stop_words', 'n_non_stop_unique_tokens', 'average_token_length']] = 0
```

Second, considering the relationship between n_non_stop_words and average_token_length, missing values for these two variables were filled. Since the correlation coefficient between n_non_stop_words and average_token_length was higher than that between n_non_stop_words and n_non_stop_unique_tokens, if only average_token_length was missing and n_non_stop_words was not 0, missing values were filled with the median of non-zero average_token_length values. Since there were noticeable outliers with large values in the distribution, the median was used instead of the mean. If only n_non_stop_words was missing and average_token_length was not 0, n_non_stop_words was filled with 1. If both were missing, n_non_stop_words was filled with 1, and average_token_length was filled with the median of non-zero values. Although filling both with 0 was considered, it was thought that this would cause bias due to too many 0s, so the above method was chosen. Additionally, since the median of n_non_stop_words was 0.999999995984, and the majority of n_non_stop_words values are positive and close to 1, it was judged appropriate to fill missing values with 1 rather than 0.

```
# 대표값 계산
median_avg_token_length = df.loc[
    (df['average_token_length'].notnull()) & (df['average_token_length'] != 0),
    'average_token_length'
].median()
```

```
### Step 2. n_non_stop_words 와 average_token_length 관계

# 2-1. average_token_length만 결측이고, n_non_stop_words가 0이 아닌 경우
mask_atl_null = df['average_token_length'].isnull() & (df['n_non_stop_words'] != 0)
df.loc[mask_atl_null, 'average_token_length'] = median_avg_token_length

# 2-2. n_non_stop_words만 결측이고, average_token_length가 0이 아닌 경우
mask_nsw_null = df['n_non_stop_words'].isnull() & (df['average_token_length'] != 0)
df.loc[mask_nsw_null, 'n_non_stop_words'] = 1

# 2-3. 둘 다 결측인 경우
both_null = df['n_non_stop_words'].isnull() & df['average_token_length'].isnull()
df.loc[both_null, 'n_non_stop_words'] = 1
df.loc[both_null, 'average_token_length'] = median_avg_token_length
```

Third, missing values for n_non_stop_unique_tokens were filled considering its relationship with n_non_stop_words. Since all missing values for n_non_stop_words had already been filled in the previous step, only the missing values for n_non_stop_unique_tokens needed to be handled. If n_non_stop_words was 0, n_non_stop_unique_tokens was also filled with 0; if n_non_stop_words was not 0, missing values for n_non_stop_unique_tokens were filled with the mean of non-zero n_non_stop_unique_tokens values. Since there were no noticeable outliers in the distribution of non-zero n_non_stop_unique_tokens, the mean was used instead of the median.

```
mean_unique_tokens = df.loc[
    (df['n_non_stop_unique_tokens'].notnull()) & (df['n_non_stop_unique_tokens'] != 0),
    'n_non_stop_unique_tokens'
].mean()
```

```
### Step 3. n_non_stop_words 와 n_non_stop_unique_tokens 관계

# 3-1. n_non_stop_words == 0이면 나머지도 0
mask_nsw_zero = df['n_non_stop_words'] == 0
df.loc[mask_nsw_zero, ['n_non_stop_unique_tokens', 'average_token_length']] = 0

# 3-2. n_non_stop_unique_tokens만 결측이고, n_non_stop_words가 0이 아닌 경우
mask_unt_null = df['n_non_stop_unique_tokens'].isnull() & (df['n_non_stop_words'] != 0)
df.loc[mask_unt_null, 'n_non_stop_unique_tokens'] = mean_unique_tokens
```

2-2) Previously, only variable pairs with correlation coefficients above 0.8 were handled by regression, but it was decided that it is reasonable to also fill missing values for pairs with correlation coefficients above 0.7 using regression. (See additional variable pairs in 2.1.1. 2-1)

The rest of the process was carried out the same as in the first round.

2-3) Verification of Completed Missing Value Imputation

To check if missing value imputation was completed perfectly, data.isnull().sum().sum() was used to confirm the number of remaining missing values after all the above processes.

```
print("결측치 남은 개수:", train.isnull().sum().sum())
train.to_csv('train_clean_reg.csv', index = False)
print("결측치 남은 개수:", test.isnull().sum().sum())
test.to_csv('test_clean_reg.csv', index = False)
```

```
결측치 남은 개수: 0
결측치 남은 개수: 0
```

As a result, it was confirmed that missing values were completely handled, and the train set with completed missing value imputation was saved as train_clean_reg.csv, and the test set as test_clean_reg.csv.

2.1.3. Outlier Handling

Some numerical variables in the data contain extreme outliers, which could negatively affect model training stability and performance. For example, the num_imgs variable contains values exceeding 100, with samples having values over 60 concentrated in the tail, distorting the overall distribution.

Before deciding on an outlier handling method, the distribution characteristics of the variables were analyzed. As a result, many numerical variables showed high skewness, indicating that the data does not follow a normal distribution. Therefore, it was judged that the Z-score-based outlier detection method, which assumes normality, is not suitable.

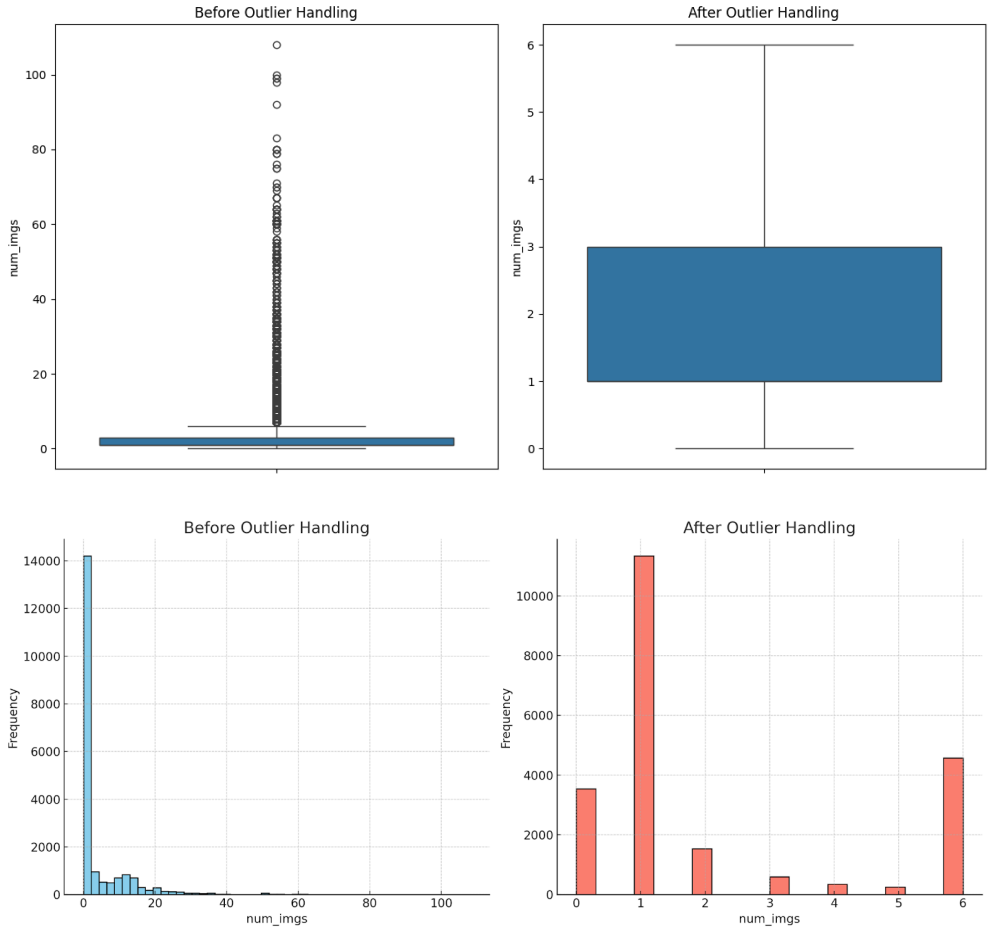
Accordingly, the IQR (Interquartile Range) method, which can robustly detect extreme values regardless of the data distribution, was adopted. The IQR method works particularly well for skewed distributions, which matches the characteristics of this data. Using the commonly used criterion of 1.5 × IQR, about 20% of outliers were detected in this data, which is a natural result. It was judged that relaxing this criterion would risk missing true outliers, so it was not appropriate.

Outliers were detected by defining the lower and upper bounds as follows:

Lower Bound = Q1 - 1.5 × IQR , Upper Bound = Q3 + 1.5 × IQR

Values outside this range were handled by applying a clipping method, replacing them with the lower or upper bound of the variable. This was applied to all numerical variables except categorical variables.

The distributions before and after outlier handling were compared using boxplots and histograms, and it was confirmed that the data distribution became more normalized and stable after processing.



To verify the impact of outlier handling on the model, the same model (LightGBM) was trained using train_clean_reg.csv and train_outlier_handled.csv, and performance metrics were compared.

이상치 처리 전후 비교

Dataset	Accuracy	F1-score	AUC	MeanScore
이상치 처리 전	0.659685	0.654154	0.7208	0.678213
이상치 처리 후	0.663964	0.661063	0.721745	0.682257

Accuracy, F1-score, AUC, and the overall mean score (MeanScore) all improved after outlier handling, with the F1-score increasing by about 0.0069, showing the largest improvement. This indicates that outlier clipping stabilized the distribution of the training data, allowing the model to learn more generalized patterns. In other words, after outlier handling, the tendency for model overfitting decreased, and performance metrics improved overall. The final dataset with outliers handled was saved as train_outlier_handled.csv and used in subsequent steps of the preprocessing pipeline.

2.2. Data Transformation

2.2.1. Categorical Encoding

After handling outliers, we converted the two categorical variables, data_channel and weekday, into numerical variables so that the model could learn smoothly. Since both variables are nominal data and there was no need to preserve order or grade differences, we used One-Hot Encoding to separate each categorical variable into binary columns.

```
categorical_cols = ["data_channel", "weekday"]

#One-Hot Encoder 설정
ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)

train_ohe = pd.DataFrame(
    ohe.fit_transform(test[categorical_cols]),
    columns=ohe.get_feature_names_out(categorical_cols),
    index=test.index)

test_ohe = pd.DataFrame(
    ohe.transform(test[categorical_cols]), #train set과 동일하게 정리
    columns=ohe.get_feature_names_out(categorical_cols),
    index=test.index)
```

“data_channel” was transformed into a total of 6 binary columns in the format of “data_channel_channelname”, and “weekday” was transformed into 7 binary columns in the format of “weekday_dayname”. The encoded file was saved as train_encoded.csv.

2.2.2. Feature Selection

Before training the model, we performed feature selection based on train_encoded.csv to remove unnecessary variables and improve prediction performance.

First, we removed the id column because it was unrelated to learning, and the shares column because it only existed in the training set and not in the test set. We also removed columns with more than 10 missing values.

Variables with very low variance, excessively high correlation with other variables, and multicollinearity issues were considered as candidates for removal.

We first removed variables with a variance less than 0.0005, and for variable pairs with a Pearson correlation coefficient greater than 0.98, one of the variables was removed. We also removed variables with a Variance Inflation Factor (VIF) greater than 20, based on multicollinearity concerns.

By combining these three criteria, we extracted the variables to be removed. The variables removed are as follows: ['kw_avg_avg', 'self_reference_avg_shares', 'abs_title_sentiment_polarity']

Even if variables met the removal criteria, if they were considered important based on domain knowledge, they were retained as exceptions. Specifically, we preserved kw_avg_min and global_sentiment_polarity.

After applying the criteria, we created train_selected.csv and test_selected.csv based on the selected variables.

2.2.3. Standardization

We performed scaling on train_selected.csv and test_selected.csv, where missing value treatment, outlier handling, and feature selection were already completed.

Before standardizing all numeric variables, we first checked the skewness of each variable.

The highest absolute skewness value was 1.448450, and there were 13 variables whose absolute skewness exceeded 1.0. Variables with skewness greater than 1.0 are likely to have a highly asymmetric distribution, so we applied log

transformation to normalize them. For stability, we used the form log(1 + x) for transformation, and when there were values less than or equal to 0, all values were shifted to be positive before applying the transformation. After log transformation, we performed standardization using StandardScaler on all numeric variables so that the mean became 0 and variance became 1. Although MinMaxScaler was considered, we determined that standardization was more suitable because it preserves the relative distribution between variables while reducing the impact of outliers. Through this process, we eliminated unit differences among variables, enabling machine learning models used in the next steps to train without being affected by the scale of the variables.
The final files after scaling were saved as train_real_final.csv and test_real_final.csv, which were used as the final datasets for training.

3. Model Training and Validation

To achieve better performance, we aimed to tune as many models as possible and compare their performance to select the best-performing one. A total of nine models were used: Logistic Regression, Neural Network, Decision Tree, Random Forest, Gradient Boosting, XGBoost, CatBoost, LightGBM, and Stacking. First, we split the full dataset (train_real_final.csv) into a training set (train) and a validation set (val), with a ratio of 8:2.

Since the dataset had nearly 20,000 samples and each model had to be trained on this large volume of data, it was deemed inefficient for one person to handle all models. Therefore, we divided the models among team members, but to ensure reliability in performance comparison, we unified the evaluation criteria. As a result, we defined a custom scoring metric called mean score, calculated as the average of Accuracy, F1 Score, and ROC AUC. A custom scoring function was implemented to return this value, and it was consistently applied across all model tuning processes. Model tuning was performed only within the training set (train), and hyperparameter optimization was conducted using 5-fold cross-validation (Stratified K-Fold).

3.1. Model Tuning

3.1.1. Logistic Regression

```
model = LogisticRegression(max_iter=1000)
```

The logistic regression model was tuned using the liblinear solver based on the L2 penalty.

Logistic regression has only one major hyperparameter, C, and when using the liblinear solver, the search space is relatively small and simple. Therefore, we judged that two rounds of Grid Search would be sufficient to achieve optimal results for this model. The model's performance was compared using the mean score, which is the average of accuracy, F1 score, and ROC AUC. Tuning was conducted within the training set (80% of train_real_final.csv), excluding the validation set, using Stratified 5-Fold Cross-Validation. The mean score of the default model (C=1, penalty='l2', solver='liblinear') was approximately 0.6452. We then proceeded with two rounds of GridSearchCV.

1) 1st Grid Search

The search range for the first Grid Search was set as follows.

```
param_grid1 = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
    'solver': ['liblinear']
}
```

The result of the first Grid Search showed the best parameters as { 'C': 10, 'penalty': 'l2', 'solver': 'liblinear' }, with a best mean score of 0.64745.

2) 2nd Grid Search

```
param_grid2 = {
    'C': [0.5, 1, 2],
    'penalty': ['l2'],
    'solver': ['liblinear']
}
```

Based on the results of the first round, we narrowed the C value range and conducted a second Grid Search. The best parameters were { 'C': 2, 'penalty': 'l2', 'solver': 'liblinear' }, and the best mean score slightly improved to 0.6475.

3.1.2. Neural Network

```
model = MLPClassifier(max_iter=300, random_state=42)
```

The neural network model was based on MLPClassifier, with max_iter=300 and random_state=42 set for training stability. As with logistic regression, we judged that a limited hyperparameter search space was sufficient due to the relatively simple structure, so we only performed two rounds of Grid Search.

The default model's mean score was approximately 0.6233. Tuning was conducted through two rounds of GridSearchCV, and the model performance was evaluated using the mean score, calculated as the average of accuracy, F1 score, and ROC AUC.

1) 1st Grid Search

The search range for the first Grid Search was set as follows.

```
param_grid1 = {
    'hidden_layer_sizes': [(32,), (64,), (128,)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001]
}
```

The result of the first Grid Search showed the best parameters as { 'hidden_layer_sizes': (64,), 'activation': 'relu', 'alpha': 0.001 }, with a best mean score of 0.6243.

2) 2nd Grid Search

```
param_grid2 = {
    'hidden_layer_sizes': [(64,), (128,), (64, 32)],
    'alpha': [0.0001, 0.001, 0.01]
}
```

A second Grid Search was conducted within a narrower range. The best parameters were { 'hidden_layer_sizes': (64,), 'alpha': 0.001 }, and the best mean score slightly decreased to 0.6241.

Although both Logistic Regression and Neural Network models showed slight performance improvements through tuning, their overall performance was relatively lower compared to other tree-based models. Therefore, we concluded the tuning process at this point and did not conduct further experiments.

3.1.3. Decision Tree

```
# 기본 디시전트리 모델 정의
dt = DecisionTreeClassifier(random_state=42)
```

When the default Decision Tree model was defined as above and 5-fold cross-validation was performed on the training set (train), the mean score of accuracy, F1 score, and ROC AUC was 0.5630255279446106, indicating low performance. To improve performance, we conducted two rounds of Grid Search. Since the model volume is small, we considered two rounds of Grid Search to be sufficient for finding the optimal parameters.

1) 1st Grid Search

The search range for the first Grid Search was set as follows.

```
param_grid = {
    'max_depth': [3,5,7,9,11],
    'min_samples_split': [2,5,7,9],
    'min_samples_leaf': [1,3,5,7],
    'criterion': ['gini','entropy']
}
```

The results were as follows: Best parameters: { 'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 7, 'min_samples_split': 2 }, and the best cross-validation mean score was 0.635389186545787. Compared to the score before tuning, the performance improved by 0.0723636586. Based on the parameters found in this round, we defined the parameter range for the second Grid Search.

2) 2nd Grid Search

```
param_grid = {
    'max_depth': [3,4,5],            # 1차 서치 결과: 5
    'min_samples_split': [2,3,4],    # 1차 서치 결과: 2
    'min_samples_leaf': [6,7,8,9],   # 1차 서치 결과: 7
    'criterion': ['gini','entropy']  # 1차 서치 결과: 'entropy'
}
```

The results were as follows: Best parameters: { 'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 9, 'min_samples_split': 2 }, and the best cross-validation mean score was 0.6354823892107963. There was a change

in the 'min_samples_leaf' parameter, and the performance slightly improved compared to the first Grid Search. Compared to the default model, the performance improved by 0.07245686127.

3.1.4. Random Forest

```
# 기본 랜덤포레스트 모델 정의
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

As shown in the figure above, when the default Random Forest model was defined and 5-fold cross-validation was performed on the training set, the mean score of accuracy, F1 score, and ROC AUC was 0.6618500425914656. To improve performance, we proceeded with the following tuning process. The Random Forest model was tuned in three stages: Randomized Search, Grid Search, and Bayesian Optimization.

1) Randomized Search

The search space for the Randomized Search was set as follows. The number of iterations was set to 50, and the Randomized Search was executed.

```
# 5. 1차 Random Search
param_dist = {
    'n_estimators': np.arange(100, 501, 100),
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}
```

As a result, the best parameters were: { 'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 'log2', 'max_depth': None }, and the best mean score was 0.6737480231057686. Based on these best parameters, we defined the parameter range for Grid Search and proceeded as shown in the figure below.

2) Grid Search

```
# 1. 이전 랜덤서치 결과 기반의 그리드 탐색 범위 설정
param_grid = {
    'n_estimators': [400, 500, 600],              # 500 중심으로 조금 위아래
    'max_depth': [None, 30, 50],                  # None 근처의 깊이 추가
    'min_samples_split': [3, 5, 7],               # 5 주변의 값
    'min_samples_leaf': [3, 4, 5],                # 4 주변의 값
    'max_features': ['log2']                      # 랜덤서치 결과 이 옵션이 우수
}
```

The results were as follows:

Best Parameters (GridSearch): { 'max_depth': 30, 'max_features': 'log2', 'min_samples_leaf': 3, 'min_samples_split': 7, 'n_estimators': 600 },

Best Mean Score (GridSearch): 0.6737651164117422. The best parameters changed, and the cross-validation score slightly increased (by 0.000017093306).

Based on these best parameters, we defined the search space for Bayesian Optimization as shown below and set the number of iterations to 30.

3) Bayesian Optimization

```
# 베이지안 탐색 범위 설정 (GridSearch 결과 기준으로 좁은 범위 탐색)
param_space = {
    'n_estimators': (500, 800),                # GridSearch 결과 600
    'max_depth': (20, 40),                     # 결과 30
    'min_samples_split': (5, 10),              # 결과 7
    'min_samples_leaf': (1, 5),                # 결과 3
    'max_features': ['log2', 'sqrt']           # 결과 log2
}
```

As a result, we obtained the following:
Best Parameters (Bayesian): OrderedDict([('max_depth', 24), ('max_features', 'sqrt'), ('min_samples_leaf', 1), ('min_samples_split', 7), ('n_estimators', 799)]),
Best Mean CV Score: 0.6746550478159087.
Since the CV score improved by 0.000889931404 compared to the previous step, we selected these best parameters as the final parameters.
Compared to the default model, the mean score improved by a total of 0.01280500522 through tuning.

3.1.5. Gradient Boosting

```
GradientBoostingClassifier(n_estimators=100, random_state=42)
```

When the Gradient Boosting model was defined as above and 5-fold cross-validation was performed on the training set, the mean score of accuracy, F1-score, and AUC was 0.674756. To improve performance, we conducted multiple rounds of tuning.

1) Randomized Search

Randomized Search was repeated 30 times, and the search space was as follows.

```
'GradientBoost': {
    'n_estimators': randint(50, 300),
    'learning_rate': uniform(0.01, 0.5),
    'max_depth': randint(2, 8),
    'subsample': uniform(0.6, 0.4),},
```

The best parameters obtained from Randomized Search for the Gradient Boosting model were:

{ 'learning_rate': 0.033332831606807715, 'max_depth': 5, 'n_estimators': 257, 'subsample': 0.6931085361721216 }, and the best cross-validation score was 0.6754.

2) Grid Search

The parameter range for Grid Search was set to ±20% based on the best parameters found from Randomized Search. The best parameters after Grid Search were:
{ 'learning_rate': 0.026666265285446175, 'max_depth': 5, 'n_estimators': 257, 'subsample': 0.831730243406546 }. There was a slight decrease in the learning_rate and an increase in the subsample parameter. After Grid Search, the best CV score was 0.6797117939569308, which was about a 0.005 improvement compared to the untuned model. However, despite two rounds of tuning, the model's performance was significantly lower compared to other models. Furthermore, since Gradient Boosting is Python-based, tuning required a considerable amount of time, resulting in low efficiency. Considering these factors, we decided not to proceed with Bayesian Optimization and concluded the tuning of the Gradient Boosting model.

3.1.6. XGBoost

```
XGBClassifier(use_label_encoder=False, eval_metric="logloss", random_state=42)
```

When the XGBoost model was defined as above and 5-fold cross-validation was performed on the training set, the mean score of accuracy, F1-score, and AUC was 0.646771. To improve performance, several rounds of tuning were conducted.

1) Randomized Search

Randomized Search was repeated 30 times, and the search space was as follows.

```
'XGBoost': {
    'n_estimators': randint(50, 300),
    'learning_rate': uniform(0.01, 0.5),
    'max_depth': randint(2, 8),
    'colsample_bytree': uniform(0.5, 0.5),
    'subsample': uniform(0.6, 0.4),
},
```

The best parameters obtained from Randomized Search for the XGBoost model were:

{ 'colsample_bytree': 0.6999304858576277, 'learning_rate': 0.033332831606807715, 'max_depth': 5, 'n_estimators': 257, 'subsample': 0.6931085361721216 }, and the best CV score was 0.6810.

2) Grid Search

The parameter range for Grid Search was set to ±20% based on the best parameters found from Randomized Search. The best parameters after Grid Search for the XGBoost model were:
{ 'colsample_bytree': 0.8399165830291533, 'learning_rate': 0.026666265285446175, 'max_depth': 5, 'n_estimators': 257, 'subsample': 0.831730243406546 }. There was an approximately 0.14 increase in both 'subsample' and 'colsample_bytree', and a slight decrease in 'learning_rate'. After Grid Search, the best CV score was 0.681430327842691, showing a performance improvement of about 0.0004 compared to Randomized Search.

3) Bayesian Optimization

The search space for Bayesian Optimization was as follows.

```
# 베이지안 탐색 범위 설정 (GridSearch 결과 기준으로 좁은 범위 탐색)
param_space = {
    'n_estimators': (100, 350), #257
          'learning_rate': (0.1, 0.4), #0.026
          'max_depth': (3, 8),#5
          'subsample':  (0.6, 1.0), #0.8317
          'colsample_bytree': (0.6, 1.0),#0.8399
}
```

After Bayesian Optimization, the best parameters for the XGBoost model were:
{ 'colsample_bytree': 0.7372828076512649, 'learning_rate': 0.1046906412570721, 'max_depth': 3, 'n_estimators': 105, 'subsample': 0.7947757687796999 }, and the best mean CV score was 0.6778094168807. However, the parameter combination from Grid Search yielded a higher cross-validation score (0.6814) than the result from Bayesian Optimization (0.6778). Therefore, we adopted the Grid Search configuration as the final XGBoost model. This decision was based on prioritizing the model's generalization performance, regardless of the tuning method used.

3.1.7. CatBoost

```
CatBoostClassifier(verbose=0, random_state=42)
```

When the CatBoost model was defined as above and 5-fold cross-validation was performed on the training set, the mean score of accuracy, F1-score, and AUC was 0.676991. To improve performance, several rounds of tuning were conducted.

1) Randomized Search

Randomized Search was repeated 30 times, and the search space was as follows.

```
'CatBoost': {
    'iterations': randint(100, 1000),
    'learning_rate': uniform(0.01, 0.5),
    'max_depth': randint(2, 8),
    'l2_leaf_reg': uniform(1, 10),
    'border_count': randint(32, 255),
    'bagging_temperature': uniform(0, 1),
},
```

The best parameters obtained from Randomized Search for the CatBoost model were:
{ 'bagging_temperature': 0.7712703466859457, 'border_count': 36, 'iterations': 589, 'l2_leaf_reg': 4.584657285442726, 'learning_rate': 0.06793452976256485, 'max_depth': 2 }, with a best CV score of 0.6761.

2) Grid Search

The parameter range for Grid Search was set to ±20% based on the best parameters from Randomized Search. After Grid Search, the best parameters for the CatBoost model were:
{ 'bagging_temperature': 0.39503647709151263, 'border_count': 201, 'iterations': 367, 'l2_leaf_reg': 5.275410183585496, 'learning_rate': 0.027251476046457116, 'max_depth': 7 }.

All parameters changed significantly. The best CV score after Grid Search was 0.6788609652232847, reflecting an improvement of about 0.003 compared to the result from Randomized Search.

3) Bayesian Optimization

The search space for Bayesian Optimization was as follows.

```
# 베이지안 탐색 범위 설정 (GridSearch 결과 기준으로 좁은 범위 탐색)
param_space = {
        'iterations': (250,450), #367
        'learning_rate': (0.01, 0.04), #0.02725
        'max_depth': (5, 10), #7
        'l2_leaf_reg': (4,6), #5.2754
        'border_count': (150, 250), #201
        'bagging_temperature': (0.3, 0.5), #0.3950
}
```

After Bayesian Optimization, the best parameters for the CatBoost model were:

{ 'bagging_temperature': 0.32792674145586054, 'border_count': 172, 'iterations': 442, 'l2_leaf_reg': 5, 'learning_rate': 0.03636392729936713, 'max_depth': 6 }. Most of the parameters changed slightly.

The best mean CV score was 0.6790261136831482, showing a performance improvement of about 0.002 compared to the untuned model.

3.1.8. LightGBM

```
LGBMClassifier(random_state=42)
```

When the LightGBM model was defined as above and 5-fold cross-validation was performed on the training set, the mean score of accuracy, F1-score, and AUC was 0.6707483. To improve performance, several rounds of tuning were conducted.

1) Randomized Search

Randomized Search was repeated 50 times, and the search space was as follows.

```
param_dists = {
        "n_estimators":    randint(50, 300),
        "learning_rate":   uniform(0.01, 0.3),
        "num_leaves":      randint(20, 150),
        "max_depth":       randint(3, 12),
        "subsample":       uniform(0.6, 0.4),
}
```

The best parameters obtained from Randomized Search for the LightGBM model were:

{ 'learning_rate': 0.0330939729486379, 'max_depth': 5, 'n_estimators': 135, 'num_leaves': 47, 'subsample': 0.9718790609370292 }, with a best CV score of 0.6767664926049749.

2) Grid Search

The parameter range for Grid Search was set to ±20% based on the best parameters found through Randomized Search. After Grid Search, the best parameters for the LightGBM model were:

{ 'learning_rate': 0.03971276753836547, 'max_depth': 5, 'n_estimators': 108, 'num_leaves': 37, 'subsample': 0.7775032487496234 }.

All parameters changed except for 'max_depth'. The best CV score after Grid Search was 0.6769098773798279, showing an improvement of about 0.0002 compared to Randomized Search.

3) Bayesian Optimization

The search space for Bayesian Optimization was as follows.

```
# 베이지안 탐색 범위 설정 (GridSearch 결과 기준으로 좁은 범위 탐색)
param_space = {
        "n_estimators":     (90,120),    #108
        "learning_rate":    (0.01, 0.05),  #0.03971
        "num_leaves":       (20,50),     #37
        "max_depth":        (3,7),    #5
        "subsample":        (0.5,1.0),  #0.7775
}
```

After Bayesian Optimization, the best parameters for the LightGBM model were:

{ 'learning_rate': 0.04198213766428693, 'max_depth': 5, 'n_estimators': 121, 'num_leaves': 41, 'subsample': 0.9519260362533433 }.

All parameters increased except for 'max_depth'. The best mean CV score was 0.6776729674608539, showing an improvement of approximately 0.007 compared to the untuned model.

3.1.9. Stacking

For the stacking model, we selected CatBoost, Random Forest, and XGBoost—models with different learning mechanisms and strong performance after tuning—as base models. This composition aimed to allow prediction errors of the individual models to complement one another. CatBoost has strengths in handling categorical variables and preventing overfitting, Random Forest offers stability and interpretability, and XGBoost provides high predictive performance through refined regularization and fast training speed. For the meta-model responsible for final predictions, we experimented with both Logistic Regression and MLP.

1) Logistic Regression

We used Logistic Regression due to its simplicity, interpretability of output probabilities, and the presence of regularization options that help prevent overfitting.

```
# 4. 메타 모델 정의
meta_model = LogisticRegression(random_state=42)
```

First, we reused the tuned base models from the previous sections. For the meta-model, we initially used the default Logistic Regression model without tuning, as shown in the figure above. When evaluated via cross-validation on the training set, the mean score was 0.6802517094262314. Due to constraints on time and

resources, we manually tuned the meta-model by comparing performance across 12 parameter combinations using 5-fold cross-validation within the training set.
The tuning parameter combinations are shown below.

```
# 1. 튜닝 대상 조합 설정 (penalty, solver, l1_ratio, C)
param_combinations = [
    {'penalty': 'l2', 'solver': 'lbfgs', 'l1_ratio': None, 'C': 0.01},
    {'penalty': 'l2', 'solver': 'lbfgs', 'l1_ratio': None, 'C': 0.1},
    {'penalty': 'l1', 'solver': 'liblinear', 'l1_ratio': None, 'C': 0.01}
    {'penalty': 'l1', 'solver': 'liblinear', 'l1_ratio': None, 'C': 0.1},
    {'penalty': 'elasticnet', 'solver': 'saga', 'l1_ratio': 0.5, 'C': 0.01},
    {'penalty': 'elasticnet', 'solver': 'saga', 'l1_ratio': 0.5, 'C': 0.1},
    {'penalty': 'l2', 'solver': 'lbfgs', 'l1_ratio': None, 'C': 1.0},
    {'penalty': 'l2', 'solver': 'lbfgs', 'l1_ratio': None, 'C': 10.0},
    {'penalty': 'l1', 'solver': 'liblinear', 'l1_ratio': None, 'C': 1.0},
    {'penalty': 'l1', 'solver': 'liblinear', 'l1_ratio': None, 'C': 10.0},
    {'penalty': 'elasticnet', 'solver': 'saga', 'l1_ratio': 0.2, 'C': 0.1},
    {'penalty': 'elasticnet', 'solver': 'saga', 'l1_ratio': 0.8, 'C': 0.1}]
```

As a result, the combination {penalty = l1, solver = liblinear, l1_ratio = None, C = 0.01} achieved the best mean score of 0.6812462795754222, representing a performance improvement of 0.000994570149 over the default meta-model.

2) MLP

While linear meta-models like Logistic Regression can only learn linear combinations of the base models' predictions, MLP was chosen because it can capture non-linear relationships among base model predictions.

```
mlp = MLPClassifier(random_state=42)
```

As with the previous case, we reused the tuned base models from earlier sections, and initially used the default MLP model as the meta-model, without any tuning, as shown in the figure above. Under this setting, the cross-validation score on the training set had a mean score of 0.6902.

Since the meta-model plays a less significant role in performance compared to the base models, we concluded that Randomized Search alone would be sufficient for tuning.
The Randomized Search parameter space was defined as follows.

```
param_dists = {
    "hidden_layer_sizes": [(5,), (10,), (20,), (10,10), (20,10)],
    "activation": ["relu", "tanh"],
    "alpha": uniform(1e-5, 1e-2),
    "learning_rate_init": uniform(1e-4, 1e-2),
    "solver": ["adam"],
    "early_stopping": [True],
    "validation_fraction": [0.1, 0.2],
    "n_iter_no_change": [10, 20],
    "max_iter": [500],
}
```

As a result of the Randomized Search, the best parameters for the MLP meta-model were:
{ 'activation': 'tanh', 'alpha': 8.066305219717406e-05, 'early_stopping': True, 'hidden_layer_sizes': (5,), 'learning_rate_init': 0.003012291401980419, 'max_iter': 500, 'n_iter_no_change': 10, 'solver': 'adam', 'validation_fraction': 0.1 }. The best cross-validation score was 0.6815311604135517.

3.2. Model Selection and Improving Method

3.2.1. Validation Set Performance Comparison

After completing the tuning process in the previous steps, the finalized models were trained on the training data (train), and their performance was evaluated on the validation data (val). Four evaluation metrics were used for performance comparison: Accuracy, F1 Score, ROC AUC, and Mean Score.
The validation results for each model are as follows.

Model	Accuracy	F1 Score	ROC AUC	Mean Score
Logistic Regression	0.6423	0.6280	0.6892	0.6532
Neural Network	0.6067	0.6016	0.6562	0.6215
Random Forest	0.6670	0.6699	0.7215	0.6862
Gradient Boosting	0.6589	0.6579	0.7264	0.6811
LightGBM	0.6615	0.6606	0.7222	0.6814
XGBoost	0.6651	0.6644	0.7266	0.6854
Decision Tree	0.6206	0.6338	0.6601	0.6382
Catboost	0.6692	0.6674	0.7304	0.6890
Stacking-Logistic	0.6683	0.6699	0.7305	0.6896
Stacking-MLP	0.6667	0.6642	0.7311	0.6874

Based on the validation results, the Stacking-Logistic model achieved the highest mean score and demonstrated the best performance, so it was selected as the final model.

3.2.2. Final Improving Methods

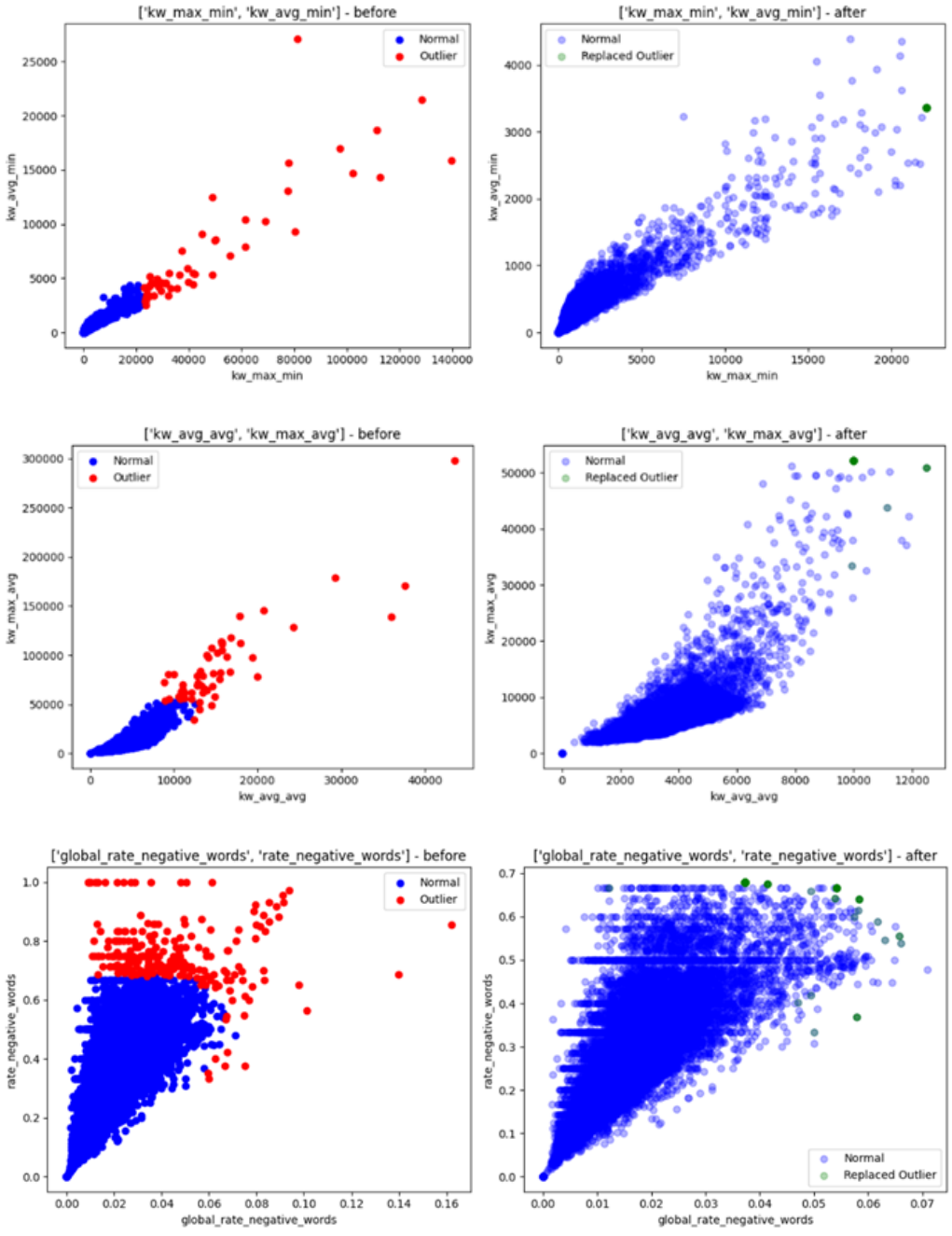
To further enhance the predictive performance of the final selected stacking model, two methods were applied. First, to address the fact that the original outlier handling approach was naive, we improved the outlier detection and treatment strategies. After reprocessing the data using this improved approach, we tuned the stacking model. Lastly, we performed threshold tuning to boost the model's performance.

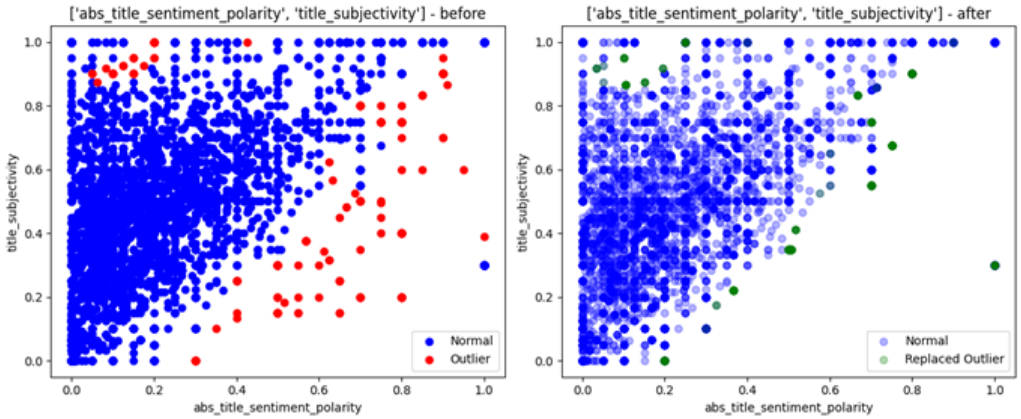
1) Outlier Detection and Handling

1-1) Using DBSCAN

To more precisely conduct outlier detection and treatment depending on the variable types, we plotted scatterplots for pairs of variables with high correlation coefficients. We then applied DBSCAN to these scatterplots to detect outliers and replaced detected outlier values with the nearest normal data point. Outlier detection using DBSCAN was performed for a total of four variable pairs (eight variables). For each variable pair, we explored optimal hyperparameters by adjusting eps and min_samples. During this process, we carefully considered factors such as the "number of points detected as outliers (1–2% of the entire dataset)," the "shape of the scatterplot," and

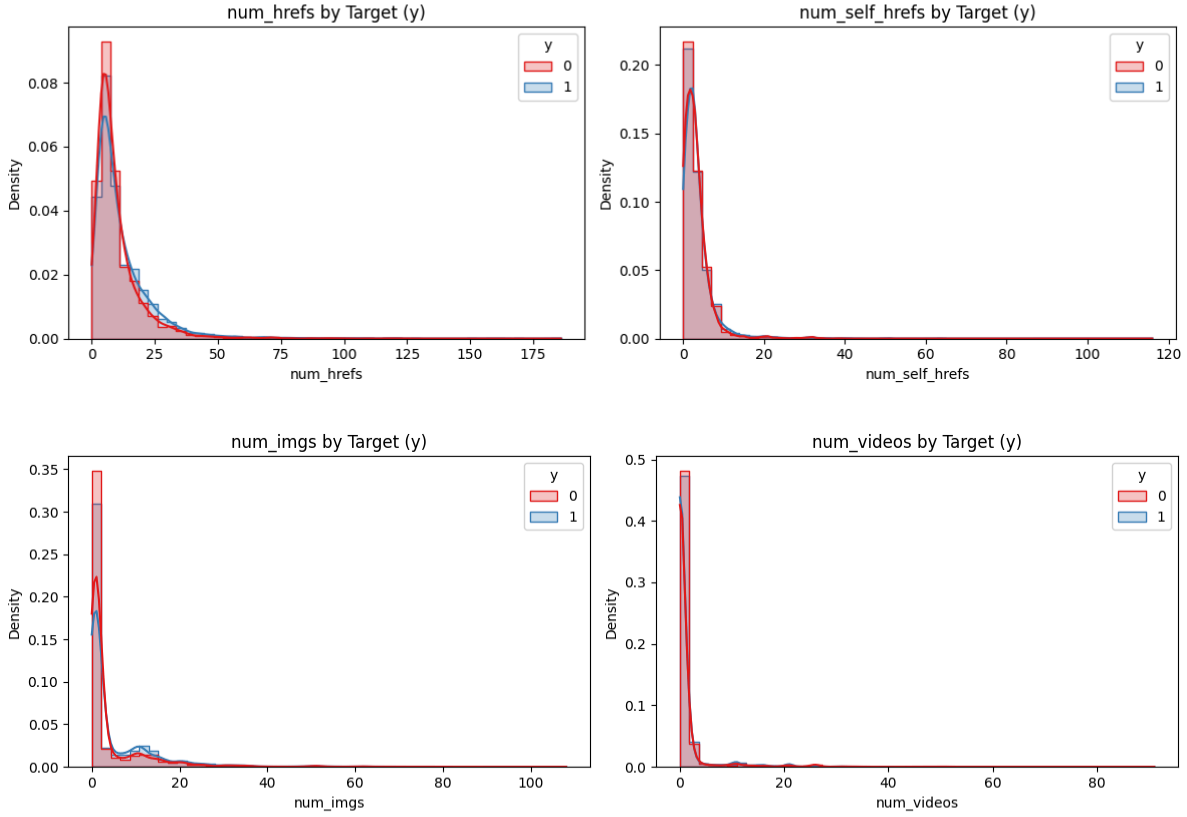
whether “important information was excessively removed as outliers.” The following graphs show the scatterplots before and after replacing outliers with the nearest normal point using DBSCAN. In the right scatterplot, the red dots indicate outliers, while in the left image, the green dots represent the outliers that have been processed and replaced.





1-2) Applying the Quantile Approach

For the remaining variables not processed with DBSCAN, we decided to continue applying the clipping approach as before. Initially, we used a clipping method based on the IQR approach. However, after examining the histograms of the variables, we realized that many variables had distributions skewed to one side, as shown in the figure below. To address this, we shifted our strategy to use the quantile approach, which can more effectively remove outliers concentrated on one side. By varying the quantiles, we generated multiple datasets and compared their performance using a baseline random forest model. Based on these comparisons, we determined that clipping at the lower 5% and upper 95% thresholds produced the highest model performance, so we adopted these thresholds as the final clipping criteria.



2) Meta-Model Tuning

After modifying only the outlier treatment in the preprocessing process, we created a new final dataset. Due to time and resource constraints, we did not retune the parameters of the stacking model's base models and used them as they were. For the meta-model, which is logistic regression, we manually tuned six parameter candidates. The candidates are listed below.

```
# 1. 튜닝 대상 조합 설정 (penalty, solver, l1_ratio, C)
param_combinations = [
    {'penalty': 'l2', 'solver': 'lbfgs', 'l1_ratio': None, 'C': 0.01},
    {'penalty': 'l2', 'solver': 'lbfgs', 'l1_ratio': None, 'C': 0.1},
    {'penalty': 'l1', 'solver': 'liblinear', 'l1_ratio': None, 'C': 0.01},
    {'penalty': 'l1', 'solver': 'liblinear', 'l1_ratio': None, 'C': 0.1},
    {'penalty': 'elasticnet', 'solver': 'saga', 'l1_ratio': 0.5, 'C': 0.01},
    {'penalty': 'elasticnet', 'solver': 'saga', 'l1_ratio': 0.5, 'C': 0.1}
]
```

As a result of the search, {penalty=elasticnet, solver=saga, l1_ratio=0.5, C=0.1} showed the highest cross-validation score (mean score of accuracy, F1, and ROC AUC), and thus was selected as the final parameter set. Using these parameters, we trained the stacking model on the training dataset and evaluated it on the validation set. The mean score achieved was 0.692487448869436, which was higher than that of the stacking model in section 3.2.1. This confirmed that the changes in outlier detection positively impacted the model's performance.

3) Threshold Tuning

To further improve the performance of the final tuned stacking model, we conducted threshold tuning. Considering the structural complexity of the stacking model and the computational burden, we did not use cross-validation. Instead, we once again split the training data into a mini-training set (mini_train) and a mini-validation set (mini_val), and used these to tune the threshold. We varied the threshold value from 0.40 to 0.60 in 0.01 increments, calculating the average of Accuracy, F1 Score, and ROC AUC for each threshold. The threshold with the highest average score was selected as the final threshold, which turned out to be 0.44. Finally, we retrained the stacking model on the training dataset(train) and evaluated its performance on the validation set(val) with the threshold set to 0.44. The results were as follows:

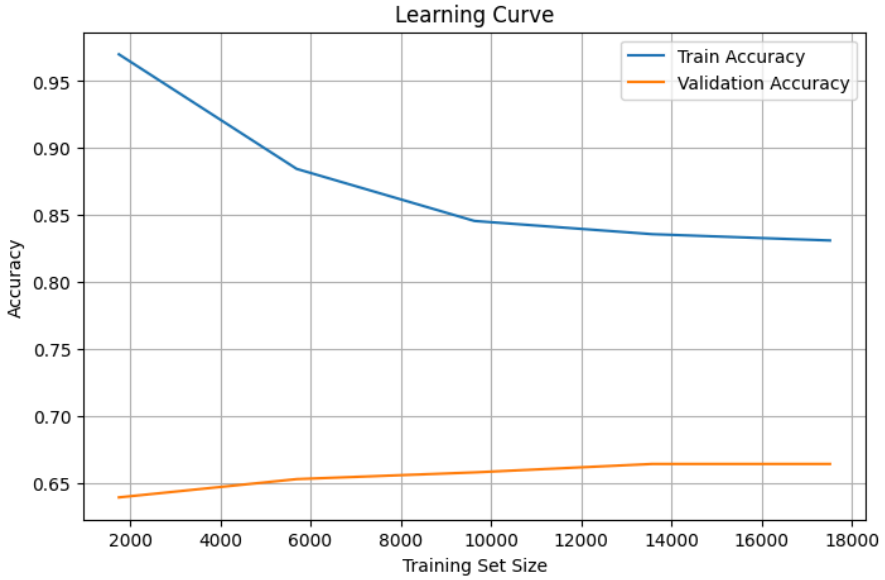
Accuracy: 0.6614928098607624
F1 Score: 0.6917480773228019
ROC AUC : 0.7333949573876959
Mean Score: 0.6955452815237534

Through the final improving methods, we achieved an improvement of approximately 0.0059 in the mean score compared to the stacking model in section 3.2.1. This procedure allowed us to fine-tune the model's classification criteria, leading to expectations of more stable and balanced performance in real-world prediction scenarios. The final model was trained on the entire dataset (the combined training and validation sets), and a tuned threshold was applied after training to make final predictions on the test set, with the results submitted accordingly.

4. Final Validation and Discussions

4.1. Overfitting

To verify whether the model has learned appropriately, we aimed to observe how performance changes during the training process by plotting the learning curve.



Looking at the training curve, it shows very high accuracy when the training data is small, and then gradually decreases. This is a very natural process that occurs as the model generalizes, and it shows that the model appropriately mitigates initial overfitting. Conversely, the validation curve starts low, but as the training data increases, the model increasingly demonstrates generalization ability, and its accuracy improves. The accuracy of the validation set increases and then gradually converges, suggesting a performance limit, which appears to be due to the inherent limitations of the given dataset.

4.2. Future Direction

To improve the project's performance, we propose the following directions:

4.2.1. Feature Engineering

In addition to the current columns, further feature engineering should be conducted, such as creating composite derived variables based on existing variables (e.g., word embedding) and integrating external data (e.g., highest word popularity in article titles, number of subscribers of the news publishing channel, etc.).

4.2.2. Diversification and Addition of Stacking Base Models

Due to time and resource constraints, the project's stacking base models include Random Forest (RF), CatBoost, and XGBoost. However, having four or more base models for stacking is generally better for performance improvement. Since the current base models do not include MLP (neural network) related models, adding neural network-related models to the base models could potentially lead to performance improvement.

5. References

McConnell, K. (2021). *Exploration of online news source data set using machine learning*. Whitman College. Retrieved from https://arminda.whitman.edu/_flysystem/fedora/2021-09/Exploration_of_Online_News_Source_Data_Set_using_Machine_Learning.pdf