# IE 7300 Statistical Learning Project Report

## <u>Online News Popularity Prediction</u>

### Group 1

| | |
|---|---|
| Vaishnavee Manivannan | manivannan.va@northeastern.edu |
| Ameya Deshmukh | deshmukh.amey@northeastern.edu |
| Yash Nikhare | nikhare.y@northeastern.edu |
| Anjali Ingle | ingle.a@northeastern.edu |

## TABLE OF CONTENTS

## ABSTRACT

This project aims to analyze a dataset of online news articles and develop a classification model to predict their popularity based on various input features. The dataset includes traits and characteristics of the articles, and through the utilization of machine learning techniques such as Support Vector Machines (SVM), Naive Bayes, and logistic regression, predictive models are constructed. The objective is to provide insights into the factors that influence article popularity, enabling informed decision-making and content optimization strategies. By employing advanced statistical methods, this study seeks to contribute to a deeper understanding of online content dynamics.

## INTRODUCTION

The popularity of online news articles is a subject of keen interest for content creators and publishers alike. Understanding the factors that contribute to an article's popularity can inform decisions regarding content creation and distribution strategies. In this study, we address this challenge by analyzing a dataset of online news articles and developing a classification model to predict their popularity. By leveraging machine learning techniques such as Support Vector Machines (SVM), Naive Bayes, and logistic regression, we aim to identify the key features that influence article popularity. Through comprehensive analysis and modeling, we seek to provide valuable insights that can guide content optimization efforts and enhance decision-making in the digital publishing domain.

The process undertaken is outlined as follows:

- Data Cleaning - Identifying and eliminating noise from the dataset.
- Feature Selection and Evaluation.
- Implementation of Machine Learning Classification.
- Summary and Conclusion.

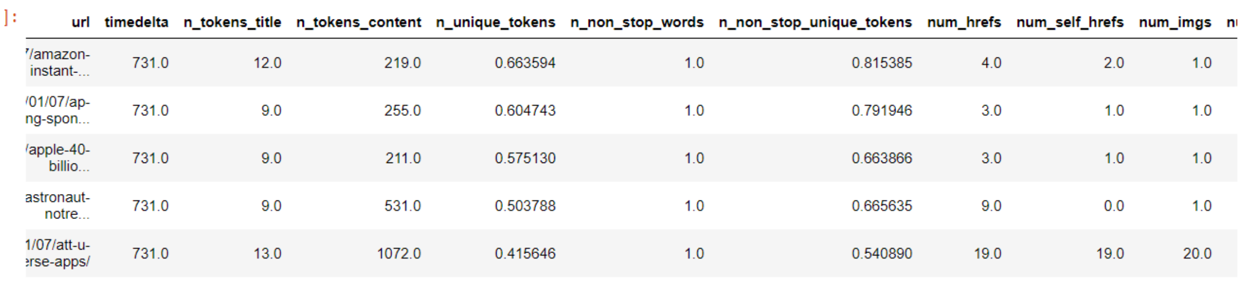
## DATA DESCRIPTION

The dataset has been downloaded from the UCI Machine Learning Repository. It offers statistics on 39797 articles published by Mashable over a period of two years. The dataset includes quantitative content elements and metadata for online news stories, with the target variable 'shares' representing the number of times the piece was shared on social media. Because the article's original content cannot be shared outside of Mashable, the dataset provides information such as the number of words, links, photos, polarity and subjectivity measures, and so on that accurately represent the articles.

The dataset link is: https://archive.ics.uci.edu/dataset/332/online+news+popularity

Alternatively, it can be downloaded from our google drive folder - link

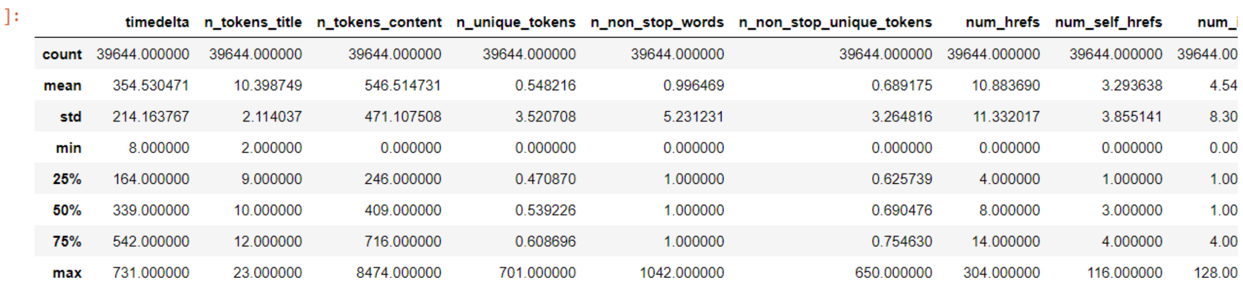
Data Overview:



]:

| url | timedelta | n_tokens_title | n_tokens_content | n_unique_tokens | n_non_stop_words | n_non_stop_unique_tokens | num_hrefs | num_self_hrefs | num_imgs | n |
|---|---|---|---|---|---|---|---|---|---|---|
| '/amazon-instant-... | 731.0 | 12.0 | 219.0 | 0.663594 | 1.0 | 0.815385 | 4.0 | 2.0 | 1.0 | |
| '01/07/ap-ng-spon... | 731.0 | 9.0 | 255.0 | 0.604743 | 1.0 | 0.791946 | 3.0 | 1.0 | 1.0 | |
| 'apple-40-billio... | 731.0 | 9.0 | 211.0 | 0.575130 | 1.0 | 0.663866 | 3.0 | 1.0 | 1.0 | |
| astronaut-notre... | 731.0 | 9.0 | 531.0 | 0.503788 | 1.0 | 0.665635 | 9.0 | 0.0 | 1.0 | |
| 1/07/att-u-erse-apps/ | 731.0 | 13.0 | 1072.0 | 0.415646 | 1.0 | 0.540890 | 19.0 | 19.0 | 20.0 | |

Statistical Summary of Numerical Features:



]:

| | timedelta | n_tokens_title | n_tokens_content | n_unique_tokens | n_non_stop_words | n_non_stop_unique_tokens | num_hrefs | num_self_hrefs | num_ |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 39644.000000 | 39644.000000 | 39644.000000 | 39644.000000 | 39644.000000 | 39644.000000 | 39644.000000 | 39644.000000 | 39644.00 |
| **mean** | 354.530471 | 10.398749 | 546.514731 | 0.548216 | 0.996469 | 0.689175 | 10.883690 | 3.293638 | 4.54 |
| **std** | 214.163767 | 2.114037 | 471.107508 | 3.520708 | 5.231231 | 3.264816 | 11.332017 | 3.855141 | 8.30 |
| **min** | 8.000000 | 2.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00 |
| **25%** | 164.000000 | 9.000000 | 246.000000 | 0.470870 | 1.000000 | 0.625739 | 4.000000 | 1.000000 | 1.00 |
| **50%** | 339.000000 | 10.000000 | 409.000000 | 0.539226 | 1.000000 | 0.690476 | 8.000000 | 3.000000 | 1.00 |
| **75%** | 542.000000 | 12.000000 | 716.000000 | 0.608696 | 1.000000 | 0.754630 | 14.000000 | 4.000000 | 4.00 |
| **max** | 731.000000 | 23.000000 | 8474.000000 | 701.000000 | 1042.000000 | 650.000000 | 304.000000 | 116.000000 | 128.00 |

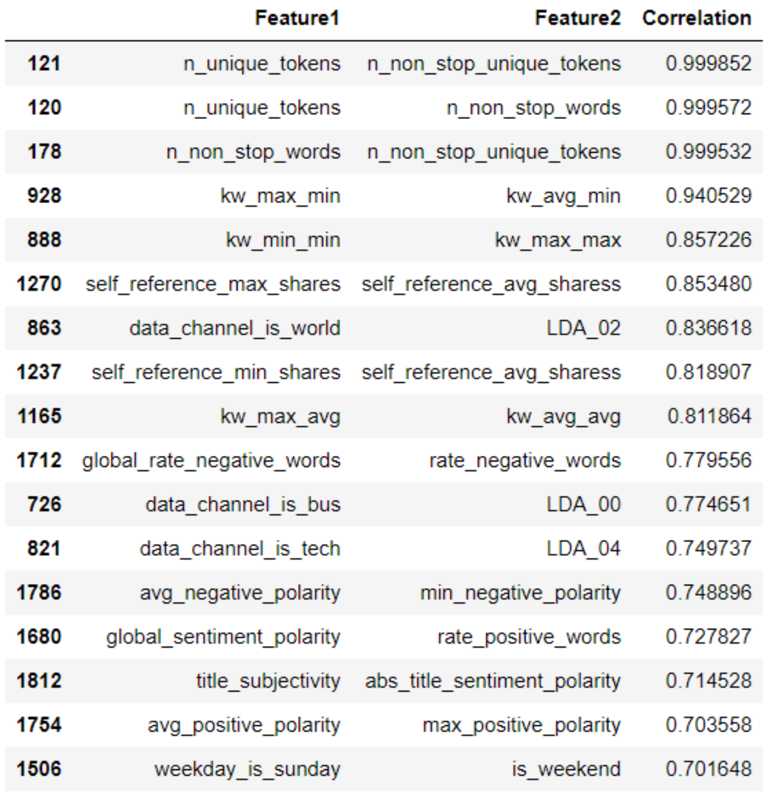
## DATA PRE-PROCESSING

In the data pre-processing stage, we conducted several steps to enhance the quality and relevance of the dataset.

***Null Value Check:*** Initially, we inspected the dataset for any null values. The data set seems to be clean with no null values in any of the columns.

***Dropping Irrelevant Columns:*** We identified and removed columns deemed irrelevant for analysis, such as 'URL' and 'timedelta', as they didn't contribute significantly to our objectives. Additionally, we consolidated multiple columns representing weekday numbers into a single column, which was subsequently ordinal encoded.

***Dropping highly correlated columns:*** Next, the pairs of highly correlated columns in the dataset were identified with a cut-off of 0.70. Below is the list of highly correlated columns.



| | Feature1 | Feature2 | Correlation |
|---|---|---|---|
| **121** | n_unique_tokens | n_non_stop_unique_tokens | 0.999852 |
| **120** | n_unique_tokens | n_non_stop_words | 0.999572 |
| **178** | n_non_stop_words | n_non_stop_unique_tokens | 0.999532 |
| **928** | kw_max_min | kw_avg_min | 0.940529 |
| **888** | kw_min_min | kw_max_max | 0.857226 |
| **1270** | self_reference_max_shares | self_reference_avg_sharess | 0.853480 |
| **863** | data_channel_is_world | LDA_02 | 0.836618 |
| **1237** | self_reference_min_shares | self_reference_avg_sharess | 0.818907 |
| **1165** | kw_max_avg | kw_avg_avg | 0.811864 |
| **1712** | global_rate_negative_words | rate_negative_words | 0.779556 |
| **726** | data_channel_is_bus | LDA_00 | 0.774651 |
| **821** | data_channel_is_tech | LDA_04 | 0.749737 |
| **1786** | avg_negative_polarity | min_negative_polarity | 0.748896 |
| **1680** | global_sentiment_polarity | rate_positive_words | 0.727827 |
| **1812** | title_subjectivity | abs_title_sentiment_polarity | 0.714528 |
| **1754** | avg_positive_polarity | max_positive_polarity | 0.703558 |
| **1506** | weekday_is_sunday | is_weekend | 0.701648 |

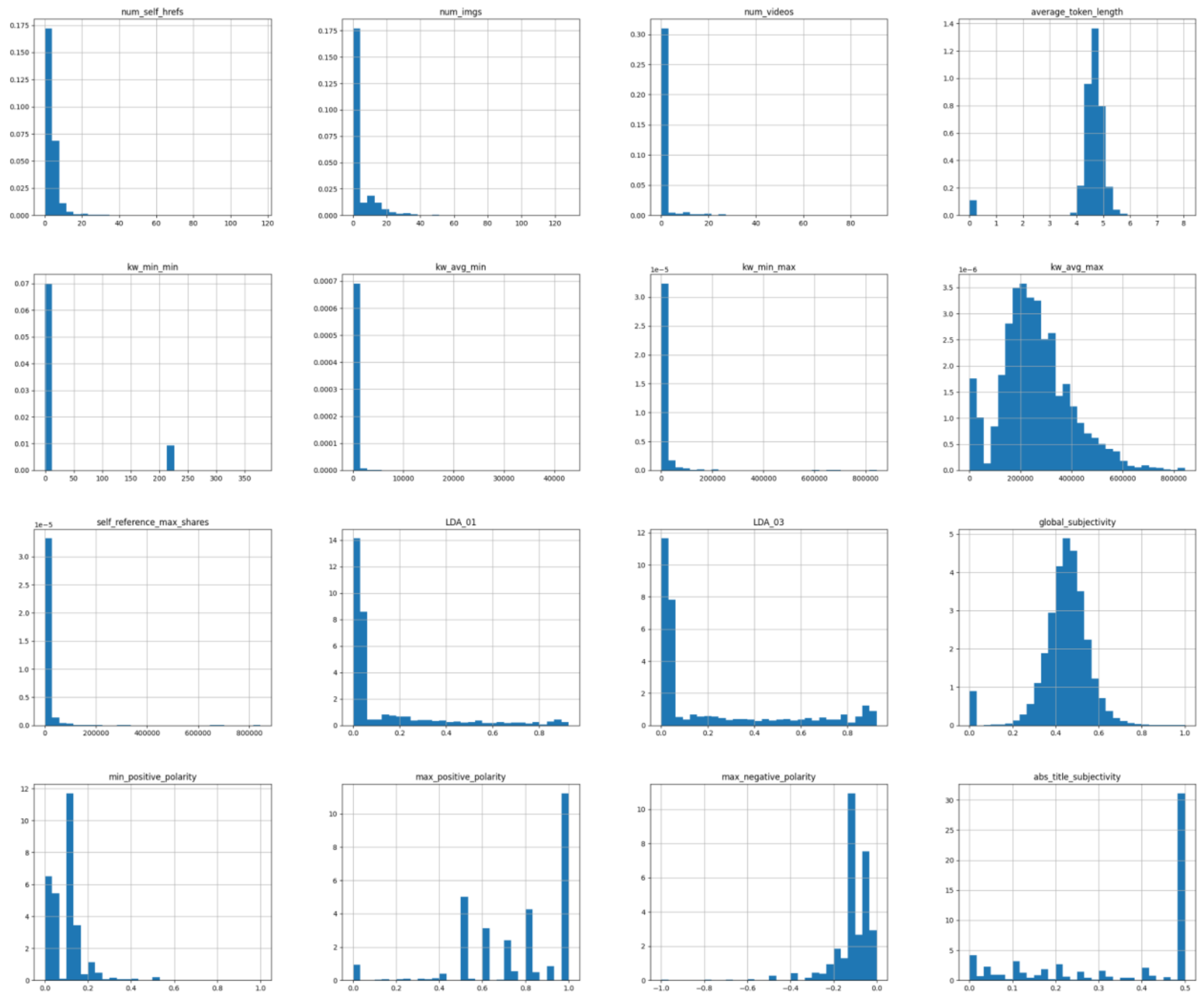
To decide which of the two columns in each pair is to be dropped, we utilized outlier detection and feature interpretability in a real-world scenario.

***Handling Outliers:*** To address outliers, particularly among highly correlated columns, we opted to drop those with a greater number of outliers. However, exceptions were made for certain columns like 'data_channel_is_bus', 'data_channel_is_world', and 'data_channel_is_tech', which possess categorical distributions (0 or 1) and hold interpretive significance in real-world scenarios.

Using correlation values and outliers, the following columns were dropped

```
columns_to_drop = ['self_reference_avg_sharess',
 'kw_max_min',
 'avg_positive_polarity',
 'abs_title_sentiment_polarity',
 'global_rate_negative_words',
 'n_non_stop_words',
 'self_reference_min_shares',
 'is_weekend',
 'kw_max_avg',
 'avg_negative_polarity',
 'rate_positive_words',
 'kw_max_max',
 'n_non_stop_unique_tokens',
 'LDA_00',
 'LDA_02',
 'LDA_04'
 ]
```

***Skewed Distribution Treatment:*** Recognizing right-skewed distributions among continuous variables, we further examined the distribution of highly skewed features. We noted that the 'number of shares' distribution exhibited heavy right skewness. The image below shows the skewness of numeric features in the dataset.



We applied the cubed root transformation to a dataset, particularly to variables with right-skewed distributions to make data more symmetrical and approximately normally distributed. Log transformation was not applied due to the presence of 0 values in the dataset. Data transformations will handle the outliers to some extent instead of dropping them. This avoids the loss of data.

## EXPLORATORY DATA ANALYSIS

### 1. Creating insightful columns:

On having a closer look at the data, we combined some of the features in the data to come up with new features and reduce the dimension. PCA was not applied as we will lose the interpretability of the model. Also, applying PCA did not improve the model performance. So the following original features were combined using simple mathematical operations.

['kw_min_max', 'kw_min_min', 'kw_avg_max', 'kw_avg_min', 'n_tokens_title', 'n_tokens_content','num_hrefs','LDA_01','LDA_03', 'num_self_hrefs','num_imgs', 'num_videos','average_token_length','num_keywords','min_positive_polarity','max_positive_ polarity','min_negative_polarity','max_negative_polarity']

***kw_min_max_spread:*** This feature represents the spread between the minimum and maximum occurrences of keywords in the content. It indicates the range of keyword occurrences, which may reflect the diversity or focus of the content.

***kw_avg_spread:*** Similar to kw_min_max_spread, this feature represents the spread between the average minimum and maximum keyword occurrences. It provides another perspective on the distribution of keyword occurrences in the content.

***num_links_to_content_length:*** This feature represents the ratio of the number of links to other content (e.g., articles, websites) in the content to the content length. It reflects the richness of external references in the content.

***num_self_links_to_content_length:*** This feature represents the ratio of the number of links to other content within the same website or domain to the content length. It indicates the internal linking structure within the content.

***average_token_length_times_num_keywords:*** This feature represents the product of the average token length in the content and the number of keywords. It provides a measure of the overall complexity or informativeness of the content.

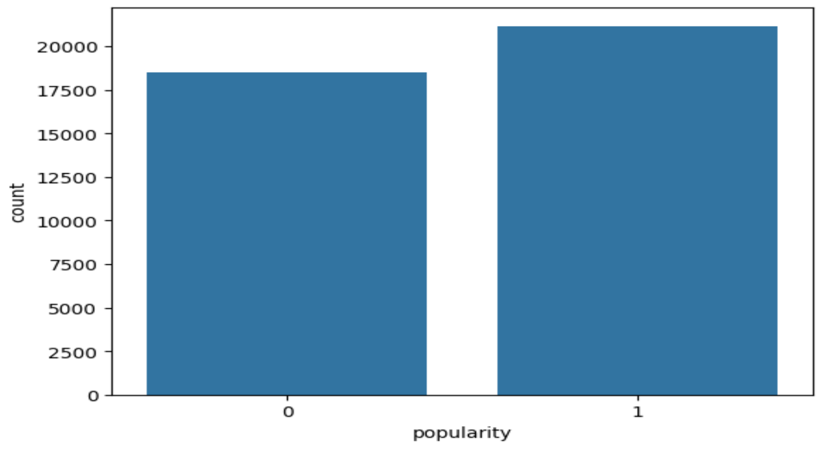
***average_lda_topic_score:*** This feature represents the average score of two LDA topics (LDA_01 and LDA_03). LDA (Latent Dirichlet Allocation) is a topic modeling technique used to discover the topics present in a corpus of text. This feature reflects the overall topic distribution in the content.

***average_positive_polarity:*** This feature represents the average positive polarity score, which is a measure of the sentiment polarity (positivity) of the content. It indicates the overall positivity of the content.

***average_negative_polarity:*** This feature represents the average negative polarity score, which is a measure of the sentiment polarity (negativity) of the content. It indicates the overall negativity of the content.

***popularity:***

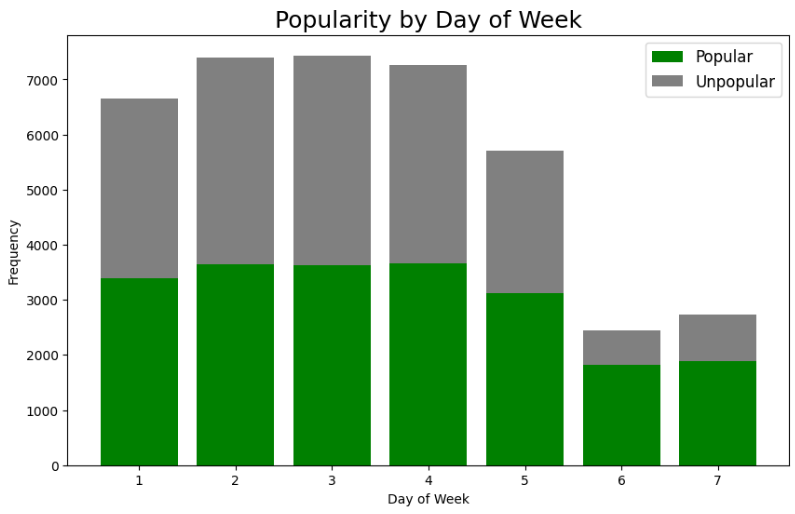
Initially, we planned on doing linear regression on the shares column. However upon literature review and building a linear regression model ourselves, we realized that there is no linear relationship between the target and features. Hence, we converted our target variable ‘shares’ into a binary target column named 'popularity' based on the number of shares of each article. Articles with fewer than 1400 shares are labeled as 0 (indicating low popularity), while those with 1400 shares or more are labeled as 1 (indicating high popularity). The cut-off of 1400 was chosen as it’s the median value. The following chart shows the distribution of our target column ‘popularity’.



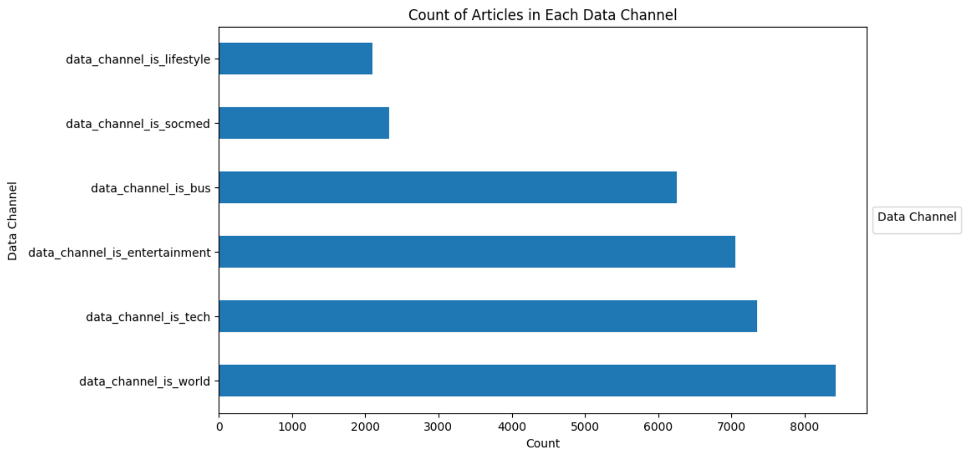
## 2. Some data insights:

### ***Popularity of articles vs day_of_week:***

We created a stacked bar plot to visualize the distribution of article popularity across different days of the week and we observed Although the number of articles published during weekends is less, a higher percentage of articles become popular during weekends than on weekdays.

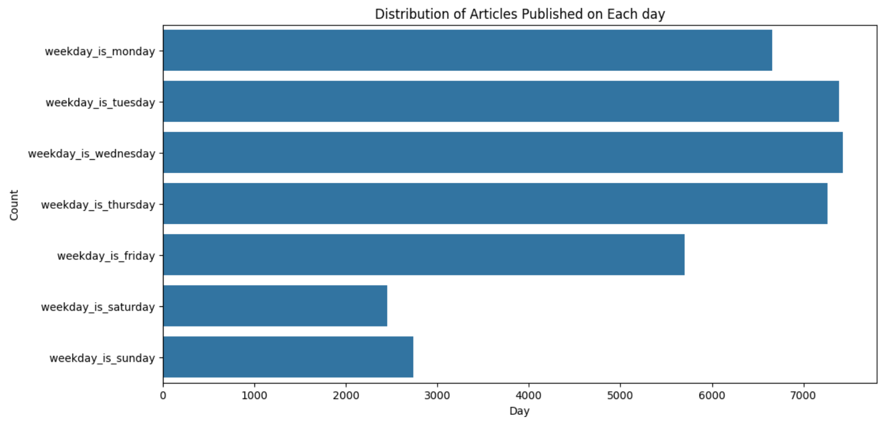


### ***Count of articles in each data column :***



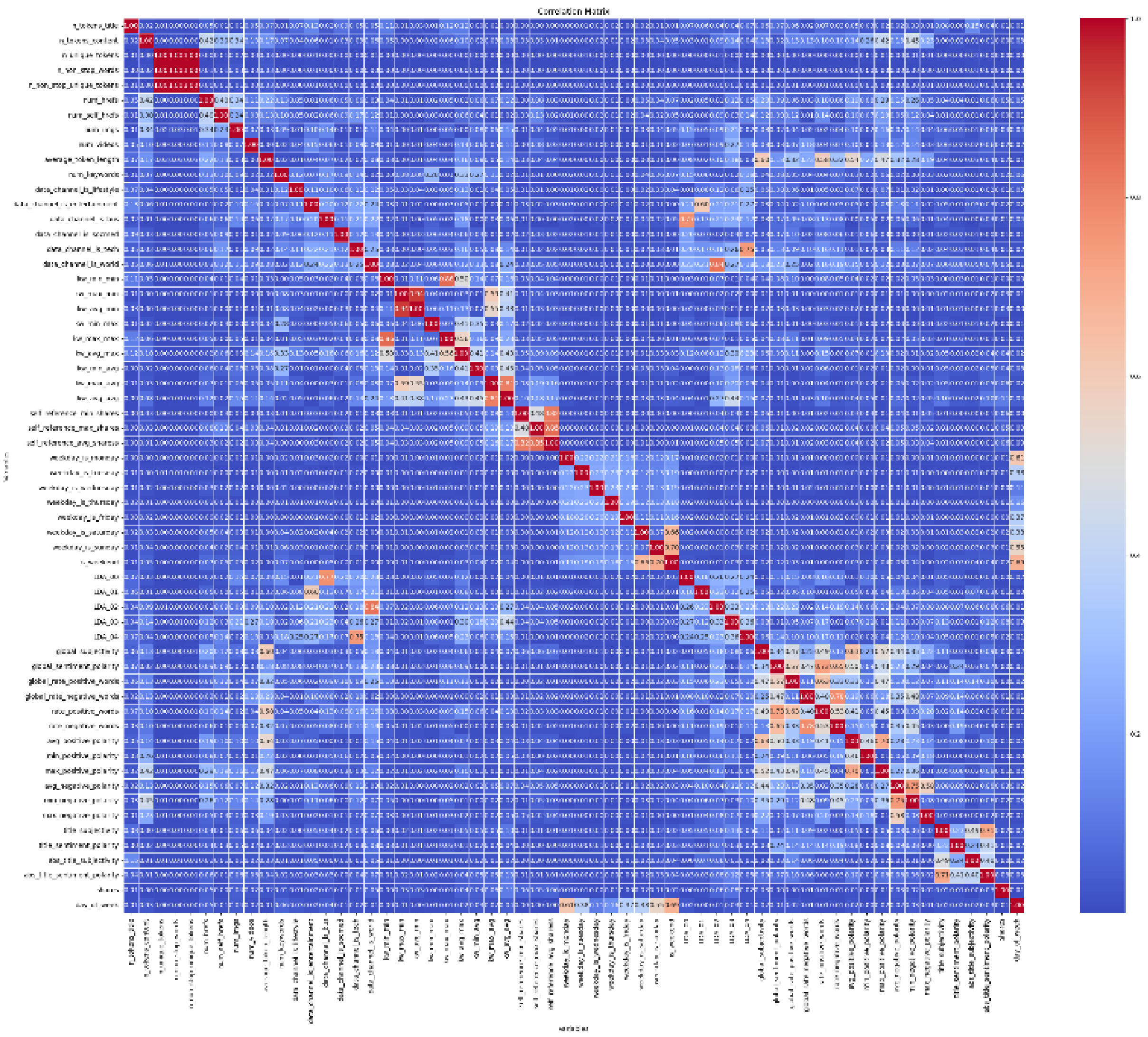
The bar plot provides a clear visualization of the distribution, making it easy to see which weekdays have the highest and lowest number of articles published. This information can be valuable for understanding trends in publishing behavior and optimizing content publication strategies.

***Distribution of articles published on each day***



The plot visualizes the distribution of articles across different data channels, including lifestyle, entertainment, business, social media, technology, and world news. Each bar represents the count of articles in a specific data channel, with the height of the bar indicating the relative number of articles. The stacked bars allow for easy comparison between the data channels, highlighting the popularity of each topic. This visualization can help identify trends in content consumption and guide content strategy decisions.

**3. Correlation Matrix:** We also performed correlation analysis to identify highly correlated columns in the dataset. Drop highly correlated, less useful columns. We will drop these columns that are highly correlated with another column in the data frame. The exception is made for data_channel_is_bus,data_channel_is_world, data_channel_is_tech' because these features are more interpretable in real-world scenarios, and also their distribution is clearly categorical(0 or 1) as opposed to LDA_00, LDA_02, and LDA_04. We have also considered the correlation heatmap(manual check). Let's say if a feature is highly correlated with two other features(that are not correlated), then that particular feature should be dropped from the other two features.



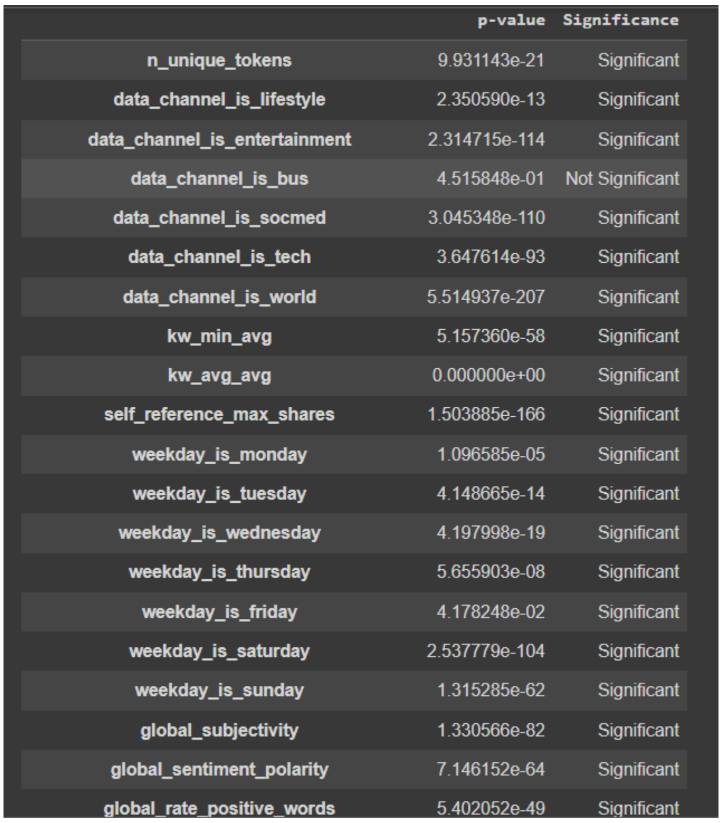
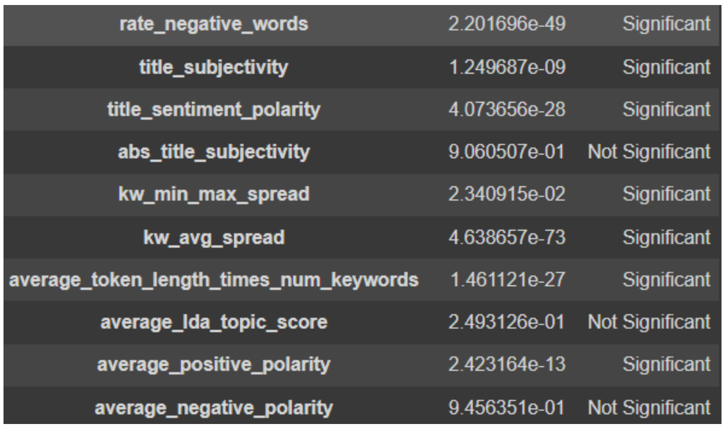
## 5. Statistical Tests

Statistical tests are essential for machine learning models, especially in the context of feature selection, for several reasons:

1. **Identifying Significant Features**: These tests help determine which features have a significant impact on the target variable. By identifying these features, we can focus on them during model training, leading to more efficient and accurate models.
2. **Reducing Overfitting**: Including irrelevant features in a model can lead to overfitting, where the model performs well on the training data but poorly on new, unseen data. Statistical tests help us avoid overfitting by selecting only the most relevant features, improving the model's generalization ability.

3. **Improving Model Interpretability**: Understanding the relationship between features and the target variable is crucial for model interpretability. Statistical tests provide insights into these relationships, helping us understand how each feature contributes to the model's predictions.
4. **Saving Computational Resources**: Including irrelevant features in a model increases its complexity and the computational resources required for training and inference. By selecting only relevant features, we can reduce the model's complexity and improve its efficiency.
5. **Handling Multicollinearity**: Statistical tests can also help identify multicollinearity, where features are highly correlated with each other. Including such features in a model can lead to unstable coefficients and inaccurate predictions. By identifying and removing multicollinear features, we can improve the model's stability and performance.

The Kruskal-Wallis test is preferred over ANOVA in situations where the assumptions of ANOVA are not met. ANOVA assumes that the data is normally distributed and that the variances across groups are equal (homogeneity of variances). If these assumptions are violated, using ANOVA may lead to incorrect conclusions. Kruskal-Wallis, on the other hand, is a non-parametric test that does not rely on the normality assumption. It is used when the data is ordinal or when the assumptions of ANOVA are not met. By using Kruskal-Wallis, we can ensure that the statistical test is appropriate for the type of data we have, providing more accurate results.

| | p-value | Significance |
|---|---|---|
| n_unique_tokens | 9.931143e-21 | Significant |
| data_channel_is_lifestyle | 2.350590e-13 | Significant |
| data_channel_is_entertainment | 2.314715e-114 | Significant |
| data_channel_is_bus | 4.515848e-01 | Not Significant |
| data_channel_is_socmed | 3.045348e-110 | Significant |
| data_channel_is_tech | 3.647614e-93 | Significant |
| data_channel_is_world | 5.514937e-207 | Significant |
| kw_min_avg | 5.157360e-58 | Significant |
| kw_avg_avg | 0.000000e+00 | Significant |
| self_reference_max_shares | 1.503885e-166 | Significant |
| weekday_is_monday | 1.096585e-05 | Significant |
| weekday_is_tuesday | 4.148665e-14 | Significant |
| weekday_is_wednesday | 4.197998e-19 | Significant |
| weekday_is_thursday | 5.655903e-08 | Significant |
| weekday_is_friday | 4.178248e-02 | Significant |
| weekday_is_saturday | 2.537779e-104 | Significant |
| weekday_is_sunday | 1.315285e-62 | Significant |
| global_subjectivity | 1.330566e-82 | Significant |
| global_sentiment_polarity | 7.146152e-64 | Significant |
| global_rate_positive_words | 5.402052e-49 | Significant |
| rate_negative_words | 2.201696e-49 | Significant |
| title_subjectivity | 1.249687e-09 | Significant |
| title_sentiment_polarity | 4.073656e-28 | Significant |
| abs_title_subjectivity | 9.060507e-01 | Not Significant |
| kw_min_max_spread | 2.340915e-02 | Significant |
| kw_avg_spread | 4.638657e-73 | Significant |
| average_token_length_times_num_keywords | 1.461121e-27 | Significant |
| average_lda_topic_score | 2.493126e-01 | Not Significant |
| average_positive_polarity | 2.423164e-13 | Significant |
| average_negative_polarity | 9.456351e-01 | Not Significant |

## MACHINE LEARNING METHODS

Before building the model, we first did a train test split of the final cleaned dataset(df2) in the ratio of 70:30. This dataset will be used for all the models henceforth.

### 1. Logistic Regression

We implemented a logistic regression model for predicting the popularity of online news articles using the cube root transformed features.

Following are the steps in detail:

1. **Class Initialization:** The LogisticRegression class is initialized with training and testing data, as well as parameters like learning rate, epsilon, and maximum iterations for gradient descent.
2. **Feature Scaling:** Methods train_scaling and test_scaling are implemented to scale only continuous features, not one-hot encoded ones, using mean normalization and standardization
3. **Gradient Descent Training:** The gradient_descent method performs gradient descent to train the logistic regression model, updating the weights iteratively until convergence or reaching the maximum number of iterations. The following are the hyperparameters used:

   Learning Rate: 0.00001 controls the step size during gradient descent.
   Convergence Criteria (epsilon): 0.0 defines the threshold for convergence.
   Maximum Iterations: 1000 limits the number of optimization iterations.

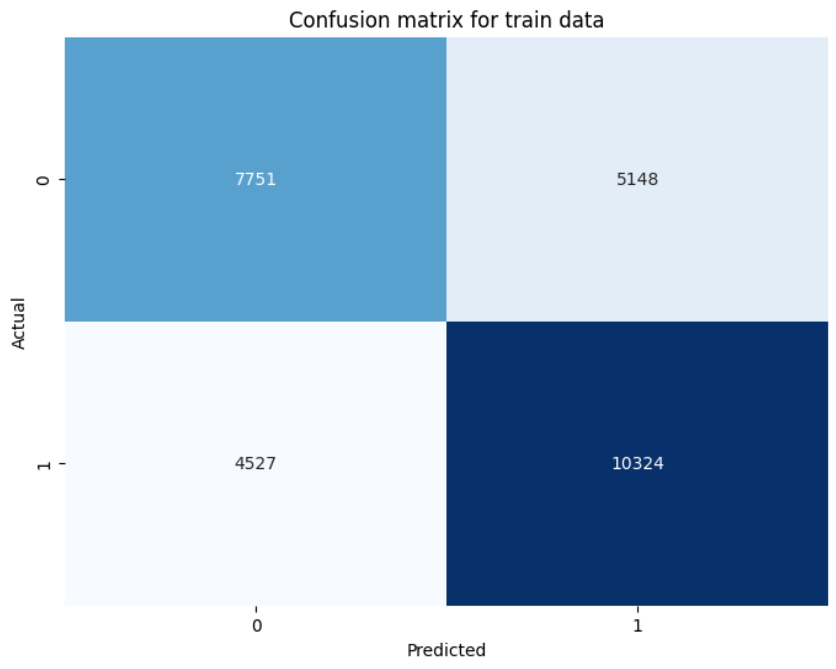
4. **Model Evaluation:** The classification_report method calculates classification metrics such as accuracy, precision, recall, and F1-score for both training and testing data. It also generates confusion matrices to visualize the model's predictions on both training and testing data.
5. **Cost Trend Visualization:** The plot_cost method visualizes the cost trend during training iterations, showing how the cost changes over time.
6. **Instance Creation and Training:** An instance of the LogisticRegression class is created (lr1) with specified parameters and trained using the fit method.
7. **Evaluation and Reporting:** After training, the model's performance is evaluated using the classification_report method, providing insights into its accuracy and other metrics on both training and testing data.

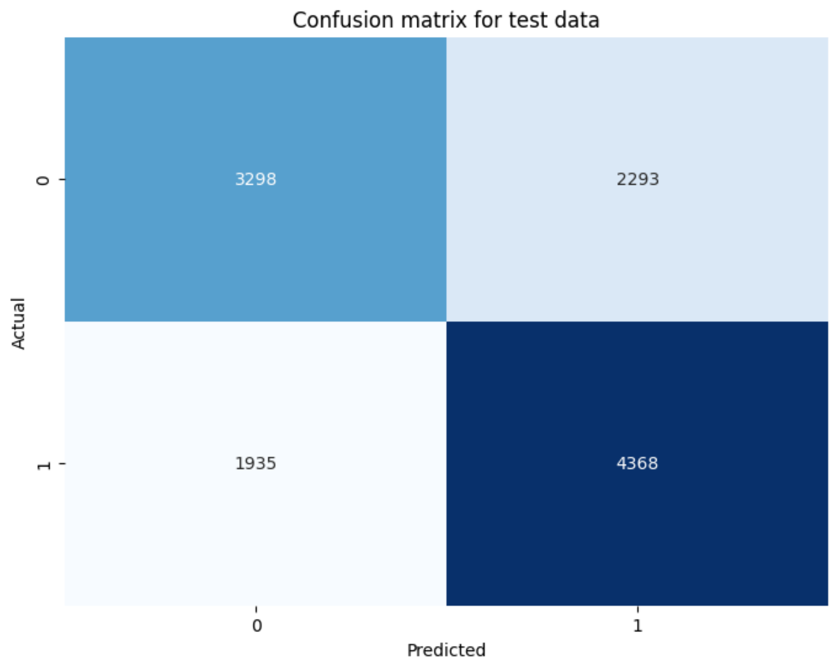
In the testing phase, the logistic regression model yielded the following results:

- Accuracy: 64.45%
- Recall: 69.30%
- Precision: 65.58%
- F1-score: 67.39%

The metrics values during the train and test phase were very similar indicating there is no overfitting of data.

These metrics indicate how well the model generalizes to unseen data. The accuracy reflects the proportion of correctly predicted instances, while recall measures the model's ability to correctly identify positive cases. Precision represents the ratio of correctly predicted positive instances among all predicted positive instances. The F1-score, being the harmonic mean of precision and recall, provides a balanced measure of the model's performance. Overall, these results suggest that the model performs reasonably well on the testing data.





## 2. Support Vector Machine

**Sampling:** As SVM's Dual optimization requires a lot of computational power we have used samples of the data. Converting class labels 0 to -1, and making sure there is no imbalance in data by sampling the +1 and -1 classes equally. Due to computational limitations, we have used only 6000 records from the original data with an 80:20 train-test split ratio.

**Class Initialization:** The KernelSVM class is initialized with the training and testing data, as well as parameters like the kernel function, regularization parameter (C), and the maximum number of iterations.

The training and testing data are stored as instance variables, with the target variable 'popularity' separated from the feature variables.

**Kernel Function:** The kernel_function method implements the kernel function used in the Kernel SVM. It supports linear, polynomial, and RBF (Radial Basis Function) kernels.

The appropriate kernel function is selected based on the kernel parameter provided during class initialization.

**Kernel SVM Optimization:** The fit method performs the Kernel SVM optimization using the Sequential Minimal Optimization (SMO) algorithm.It initializes the Lagrange multipliers (alpha) with random values between -C and C, and the bias term (b) to 0. The optimization process iterates until the maximum number of iterations is reached or the number of changed Lagrange multipliers is less than the number of training samples.

During each iteration, the method updates the Lagrange multipliers and the bias term based on the SMO algorithm.

The cost history is stored during the training process for visualization purposes.

**Cost Function Computation:** The compute_cost method calculates the cost function for the Kernel SVM, which is the sum of the hinge loss and the regularization term.

**Prediction:** The predict method uses the trained Kernel SVM model to predict the class labels for new input data.

It computes the decision function based on the Lagrange multipliers, bias term, and the kernel function, and returns the sign of the decision function as the predicted class labels.

**Metric Calculation:** The metric_calculation method computes the classification metrics (accuracy, precision, recall, F1-score) for the Kernel SVM model.

**Classification Report:** The classification_report method is responsible for the overall evaluation of the Kernel SVM model. It calls the predict method to obtain the predictions on the training and testing data, and then calculates the classification metrics using the metric_calculation method.

It also calls the plotting methods to visualize the cost function, decision boundary, and confusion matrix for both the training and testing data.

**Plotting Methods:**The plot_cost method visualizes the cost function during the training process.

The plot_decision_boundary method plots the decision boundary of the Kernel SVM model for both the training and testing data.The plot_confusion_matrix method plots the confusion matrix for both the training and testing data.

**Usage:**An instance of the KernelSVM class is created with the specified parameters (e.g., kernel function, regularization parameter, maximum iterations).

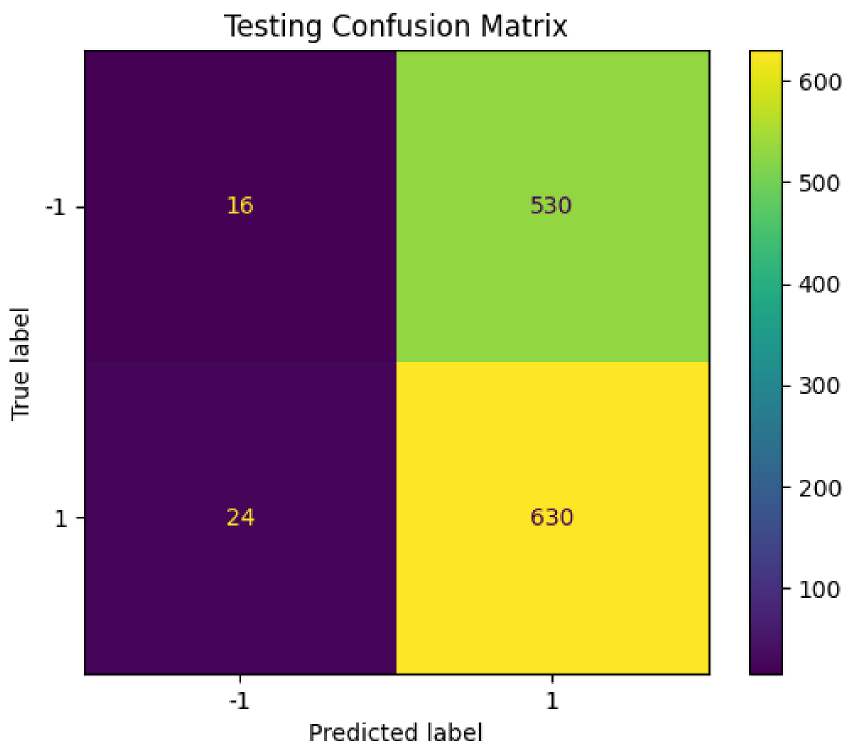
The fit method is called to train the Kernel SVM model on the provided training data.

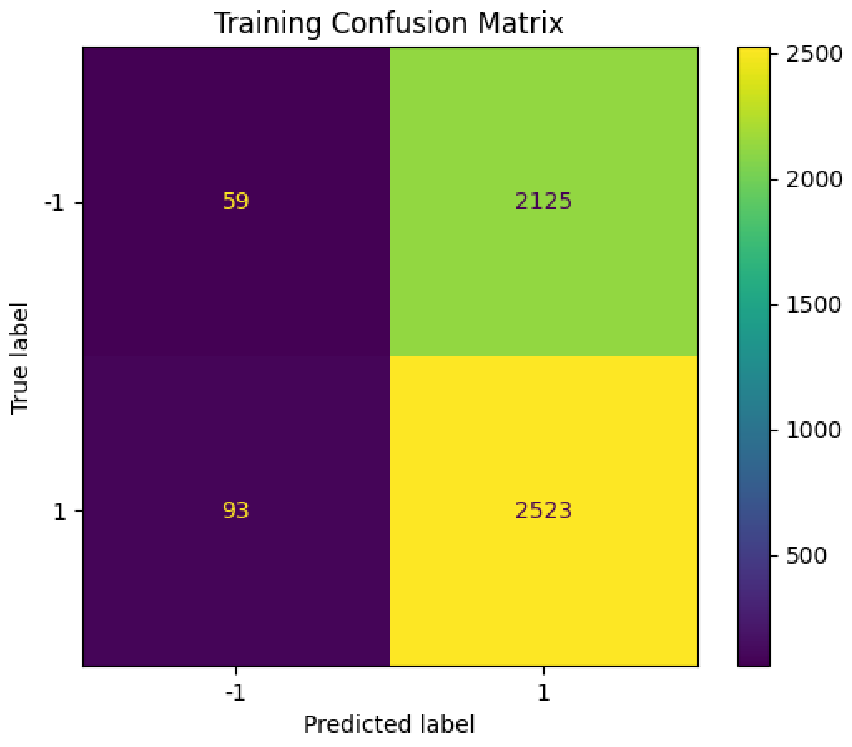
The classification_report method is then called to evaluate the model's performance on both the training and testing data and to generate the visualization plots.

In the testing phase, the logistic regression model yielded the following results:

- Accuracy: 53.8%
- Recall: 96.3%
- Precision: 54.31%
- F1-score: 69.45%

During the train and test phase, the results were very similar. It can be seen that SVM has not performed as good as logistic regression because we have done the training only on the sample dataset and not the entire data.


Testing Confusion Matrix
True label
Predicted label
-1
1
-1
1
16
530
24
630
600
500
400
300
200
100

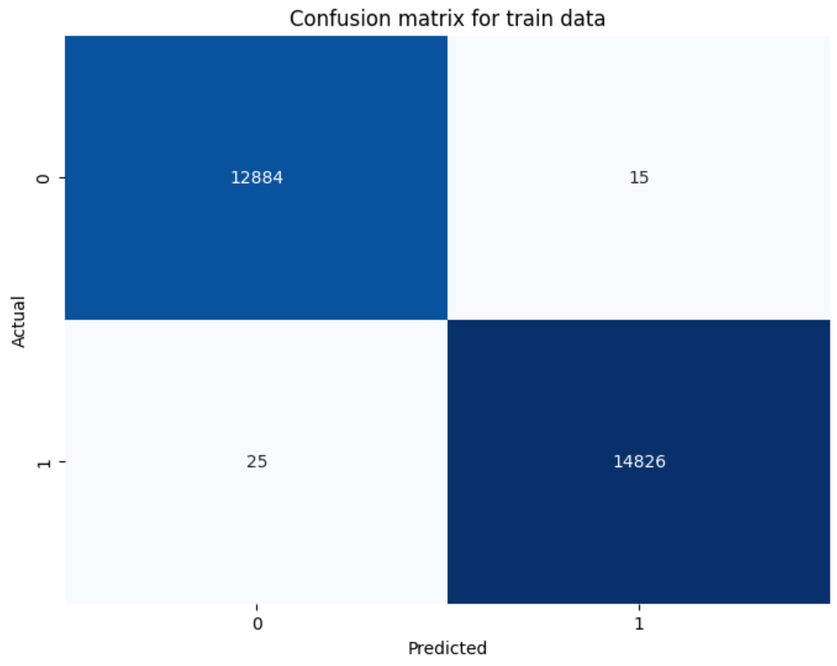

Training Confusion Matrix
True label
Predicted label
-1
1
-1
1
59
2125
93
2523
2500
2000
1500
1000
500

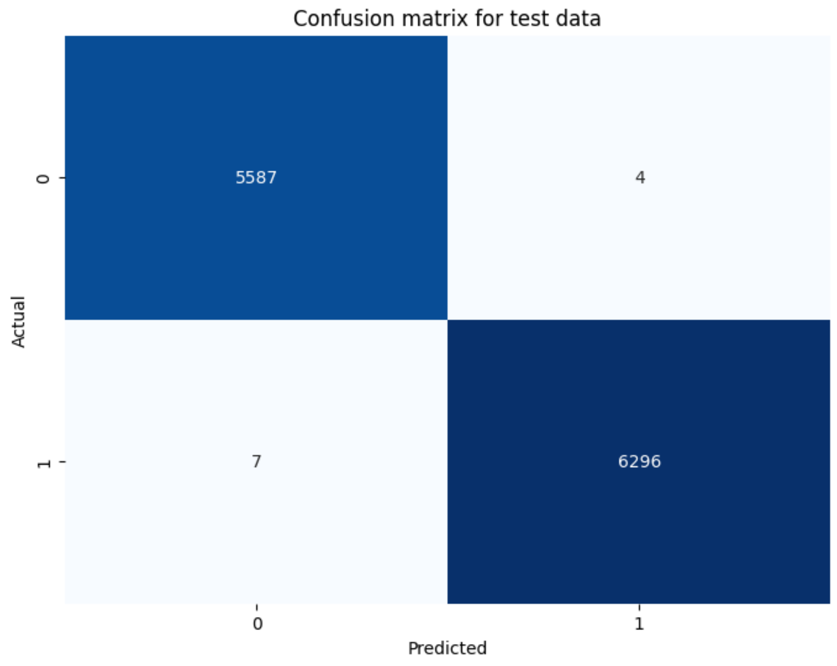
## 3. Naive Bayes

While building the Naive Bayes model, we have used discrete naive bayes for one-hot encoded columns and gaussian naive bayes for continuous features. Here is a brief description of various methods in the Naive Bayes class.

1. **Initialization**: The NaiveBayes class is initialized with the training and testing data (train_data and test_data). It separates the features and target variables for both training and testing data using the extract_features method.
2. **Data Preparation**: The extract_features method segregates the categorical and numerical features from the input data. It creates separate arrays for continuous (X_train_cont, X_test_cont) and categorical (X_train_cat, X_test_cat) features for both training and testing data.
3. **Model Training (fit):** The fit method trains the Naive Bayes classifier on the training data. It performs the following steps: Finds the unique classes and their counts in the target variable y_train. Calculates the prior probabilities for each class. For continuous features: Computes the mean and standard deviation for each feature of each class using the gaussian_fit method. Stores the mean and standard deviation in the cont_distribution dictionary. For categorical features: Calculates the value counts for each feature of each class using the single_feature_counter method. Stores the value counts in the cat_counters dictionary.
4. **Prediction (predict):** The predict method is used to make predictions on new data. It takes the feature matrix X as input and returns the predicted class labels. For each row in X, the method performs the following steps: Calculates the likelihood for numerical features using the Gaussian probability density function (gaussian_probability method). Calculates the likelihood for categorical features using the class_probabilities dictionary. Combines the likelihoods with the prior probabilities to compute the posterior probabilities for each class. Selects the class with the maximum posterior probability as the predicted class for that row.
5. **Model Evaluation:** The metric_calculation method computes classification metrics such as accuracy, precision, recall, and F1-score for both training and testing data. The confusion_matrix method generates confusion matrices for visualizing the performance of the model. Finally, the classification_report method prints a summary of the model's performance metrics for both training and testing data.

All four metrics - accuracy, precision, recall and F1 score were around 99% for both the training and test data, indicating no over-fitting. This high level of performance indicates that the combination of discrete and continuous  Naive Bayes classifiers is effectively distinguishing between the different classes in the dataset with minimal errors, demonstrating its robustness and reliability in making accurate predictions.





## 4. Feed-Forward Neural Network

Finally, we built a feed-forward neural network using Pytorch.

**1. Feedforward Neural Network Class (Feedforward):** This class defines a simple feedforward neural network architecture using PyTorch's nn.Module. It includes methods for forward pass (forward), prediction (predict), accuracy calculation (acc), AUROC calculation (auroc), and F1 score calculation (f1).

The hyperparameters for the feed-forward neural network are:

- input_dim_custom: Number of features, determined as the shape of the training data tensors along the first dimension.
- hidden_dim: Number of neurons in the hidden layer, set to 128.
- output_dim: Number of neurons in the output layer, set to 1.
- learning_rate: Learning rate for gradient descent optimization, set to 0.001.
- epochs: Number of training epochs, set to 10.

**2. Model Training:**

Define hyperparameters such as input dimension, hidden dimension, output dimension, learning rate, and epochs.

- Instantiate the Feedforward model object with the specified dimensions.
- Define the loss function (BCELoss) and optimizer (Adam) for training the model.
- Move the model to the available device (GPU if available, otherwise CPU).
- Train the model for the specified number of epochs:
    - Iterate over the training data batches.
    - Perform a forward pass, compute the loss, backpropagate gradients, and update the model parameters.
    - Track the training loss, accuracy, and predictions for each epoch.

**3. Model Evaluation on Test Data:**

- Define a function (test_evaluate) to evaluate the trained model on the test data.
- Set the model to evaluation mode (eval) and initialize evaluation metrics.

- Iterate over the test data batches, make predictions using the trained model, and update evaluation metrics.
- Print the final evaluation metrics: accuracy, AUROC, and F1 score.

In the testing phase, the Feed Forward Neural Network  yielded the following results:

- Accuracy: 63.48%
- Recall: 69.09%
- Precision: 64.51%
- F1-score: 66.72%

**RESULTS**

Classification metrics value on the test data set.

| Model | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|
| Logistic Regression | 64.45% | 69.3% | 65.57% | 67.38% |
| Support Vector Machine | 58.83% | 54.31% | 96.33% | 69.45% |
| Naive Bayes | **99.90%** | **99.93%** | **99.88%** | **99.91%** |
| Feed-Forward Neural Network | 63.48% | 64.51% | 69.09% | 66.72% |

**SUMMARY**

In our news popularity prediction problem, Naive Bayes emerges as the top performer, showcasing remarkable accuracy, precision, recall, and F1 Score, reaching nearly 99% on both training and test datasets. This suggests that Naive Bayes effectively captures the underlying patterns in the data, making it highly suitable for predicting news popularity. Logistic Regression, while showing moderate performance with an accuracy of around 64%, lags behind Naive Bayes in terms of predictive power. Support Vector Machine (SVM) exhibits a relatively high recall, indicating its ability to correctly identify popular news articles. However, its overall accuracy and precision fall short compared to Naive Bayes and as SVM's Dual optimization requires a lot of computational power, we used a sample of the dataset. Despite leveraging a Feed-Forward Neural Network, the model achieves only a modest accuracy, trailing behind Naive Bayes. With more computing power, hyper-parameter tuning on neural networks may result in an improved performance. Therefore, Naive Bayes stands out as the most effective model for predicting news popularity in this context.