Project Report for AAI-501

Group 8: Assignment 7.1 - Final Project Report

Physics vs Chemistry vs Biology

NLP text classification dataset

Puneet Chopra, Rohan Sharma and Nitin Mishra

University of San Diego

AAI-501: Artificial Intelligence

Dr. Ebrahim Tarshizi /Dr. Rakesh Das

Aug 10, 2024

Table of Contents

1. Introduction: Describing the Selected Problem:

Given a dataset with comments, the project aims to classify whether a given comment belongs to a particular category: Physics, Biology or Chemistry. This is a classification problem in that sense. It will involve dealing with the following major aspects outlined in 1.1

1.1 Summary of Work Carried out in this project

1. Exploratory Data Analysis to understand the type of data.
2. Preprocessing: To be able to classify long texts containing URL, punctuations, mixed casing and spaces, we will need to process the data to remove all these.
3. Vectorization: Since the models proposed to be evaluated require the data to be in the form of numbers instead to text, we will need to vectorize our data to convert it to vectors amenable to these methods. Different vectorisation method may have varying impact on a model's ability to accurately carry out the prediction.
4. Model Training and Prediction on vectorized data.
5. Results Comparison.

2. Data Overview

The dataset consists of comments related to educational topics, each labeled as either Physics, Chemistry, or Math. The dataset is divided into a training set for model development and a test set for evaluating the models' predictive performance.

- **Train Dataset**: Contains comments along with their corresponding topic labels.
- **Test Dataset**: Contains comments without labels, which will be predicted by the models.

2.1 Dataset Source:

https://www.kaggle.com/datasets/vivmankar/physics-vs-chemistry-vs-biology/data

Dataset Context: This dataset is created to practice various text classification techniques. The text data is collected using praw API from reddit.com.

2.2 Dataset details:

- Number of classes: 3 (Physics, Chemistry, and Biology)
- Total number of data points: 10281 comments

3. Exploratory Data Analysis on Raw Data:

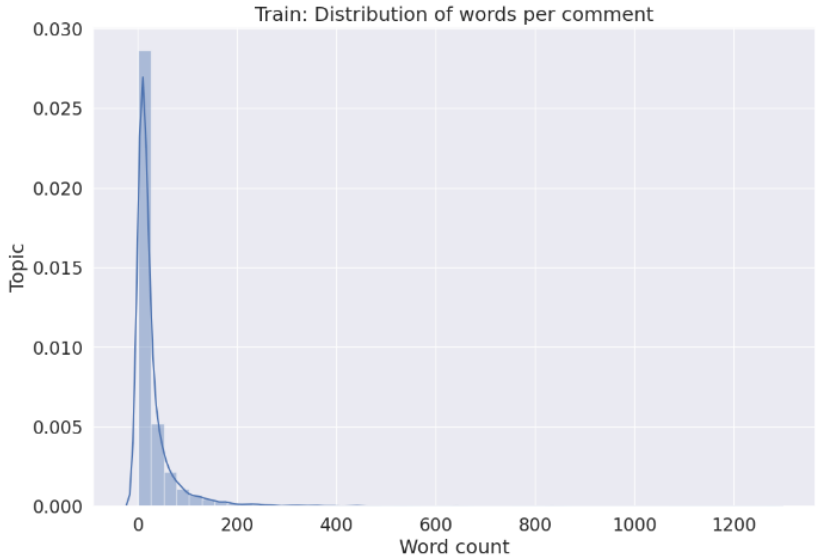
3.1 **Display the Data:** The raw data file is read and displayed.

3.2 Shape Check:

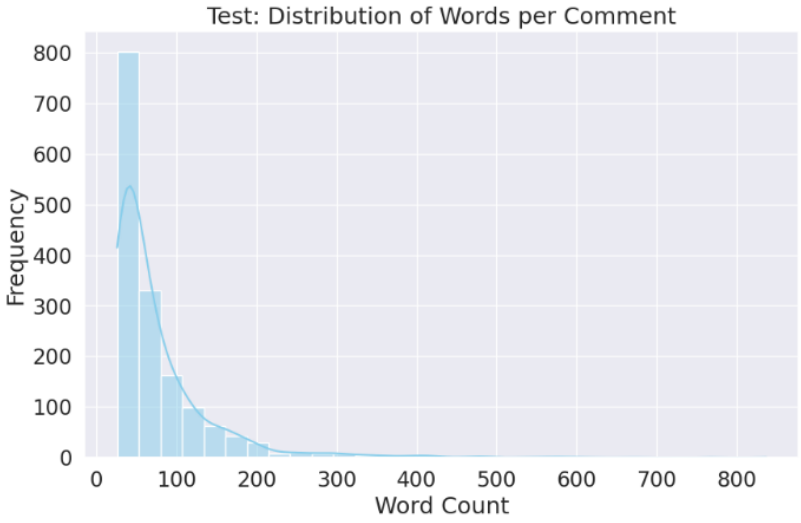
Shape of Training Data: (8695, 3) Shape of Test Data: (1586, 3)

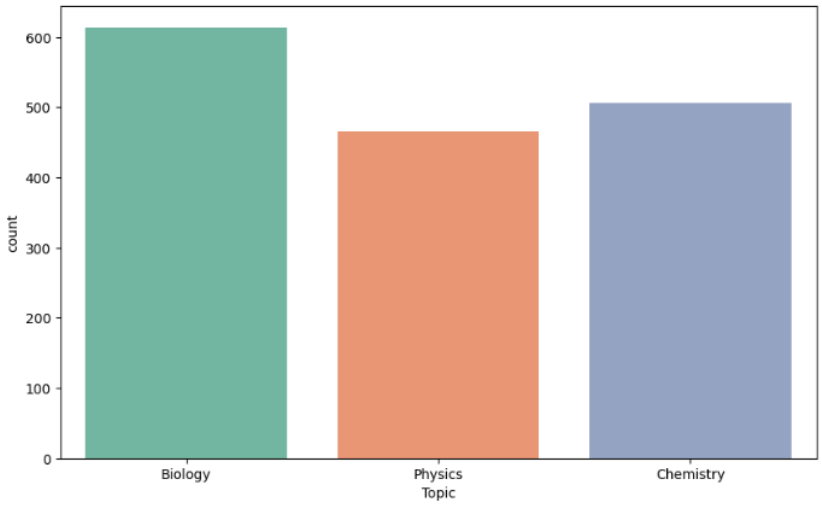
3.3 Missing values Check: No missing Values were found

3.4 Train Data: Distribution of Words per comment and Count Plot of Topics.

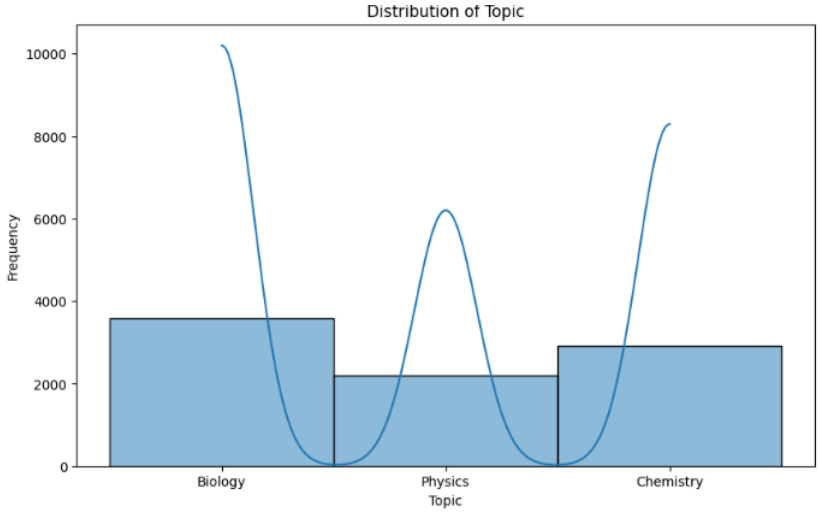


3.5 Test Data: Distribution of Words per comment and Count Plot of Topics.

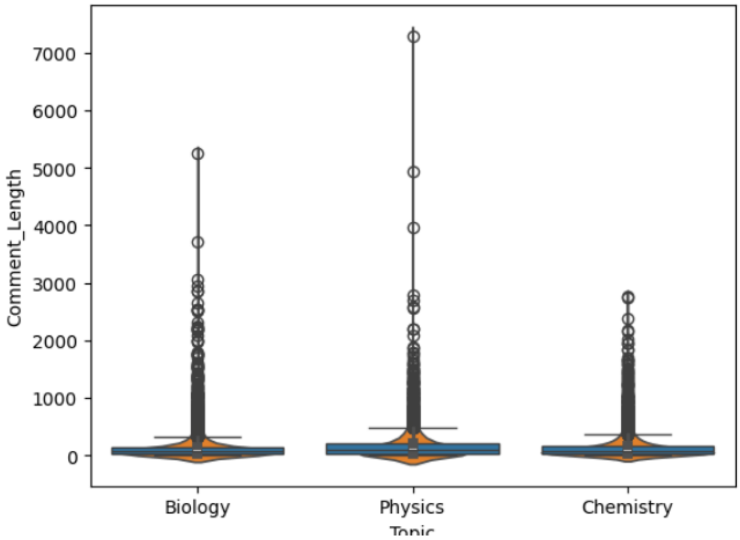




3.6 KDE Plot for Distribution of Topics.

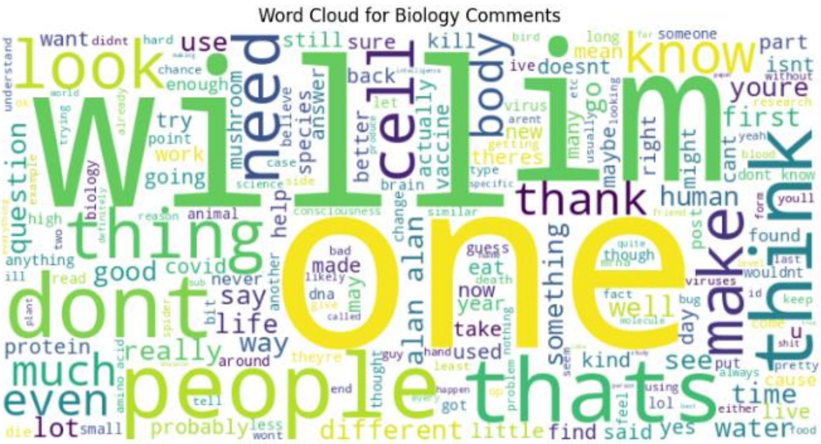


3.7 Comment Length Analysis with Violin Plot

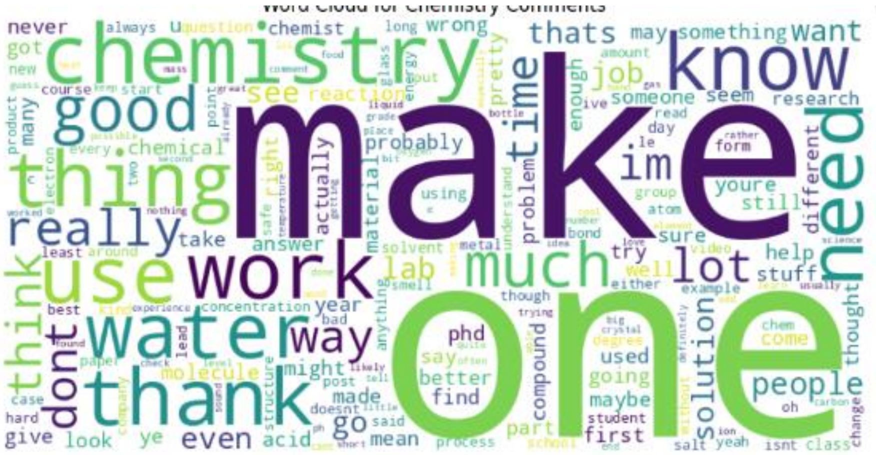


3.8 Word Cloud and Word Frequency Analysis.

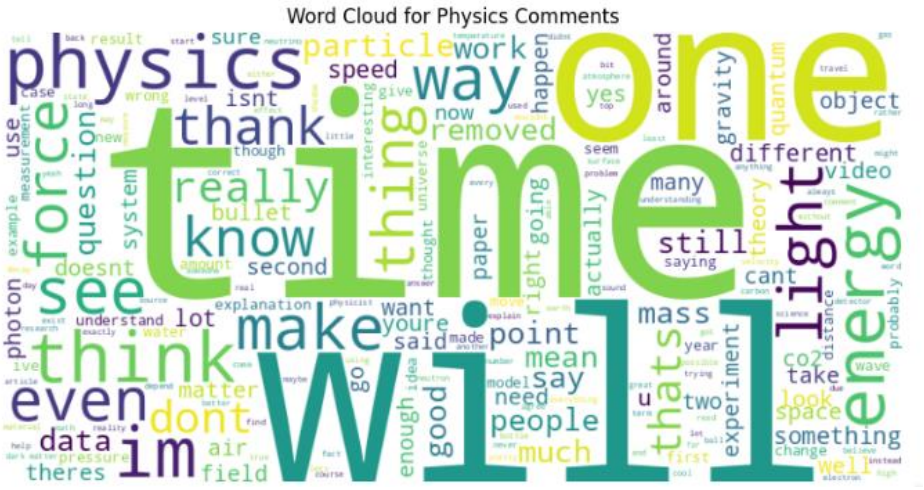
3.8.1 Biology Comments



3.8.2 Chemistry Comments



3.8.3 Physics Comments



4 Pre-Processing of Data:

Pre-processing textual data is a crucial step in preparing it for classification problems in Natural Language Processing (NLP). Here's how each pre-processing step contributes to improving the quality and effectiveness of the data:

4.1 URL Removal:

4.1.1 **Purpose:** URLs often do not contribute meaningful information for text classification tasks and can introduce noise.

4.1.2 **Benefit:** Removing URLs helps in reducing the dimensionality of the data and focuses the model on more relevant features.

4.2 Conversion to Lower Case:

4.2.1 **Purpose**: Ensures uniformity by treating words like "Text" and "text" as the same.

4.2.2 **Benefit**: Reduces the complexity of the data by minimizing the number of unique tokens, which helps in improving model performance and reducing overfitting.

4.3 Punctuation Removal:

4.3.1 **Purpose**: Punctuation marks are generally not useful for text classification and can be considered noise.

4.3.2 **Benefit**: Simplifies the text and reduces the number of tokens, making the data cleaner and more manageable for the model.

4.4 Tokenization:

4.4.1 Purpose: Splits the text into individual words or tokens.

4.4.2 Benefit: Converts the text into a format that can be easily analysed and processed by machine learning algorithms. It also helps in identifying the frequency and context of words.

4.5 **Stop Word Removal:**

4.5.1 Purpose: Removes common words (like “and”, “the”, “is”) that do not carry significant meaning.

4.5.2 Benefit: Focuses the model on more meaningful words, improving the relevance of the features and enhancing the model’s ability to learn important patterns.

4.6 **Lemmatization**

4.6.1 Purpose: Reduces words to their base or root form (e.g., “running” to “run”).

4.6.2 Benefit: Normalizes the text, reducing the number of unique tokens and capturing the essence of words. This helps in improving the generalization of the model.

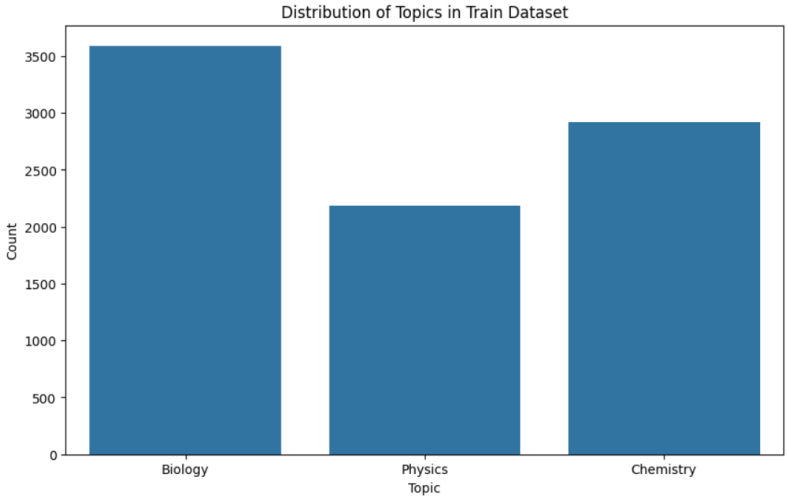
4.7 Overall Benefits for Classification Problems

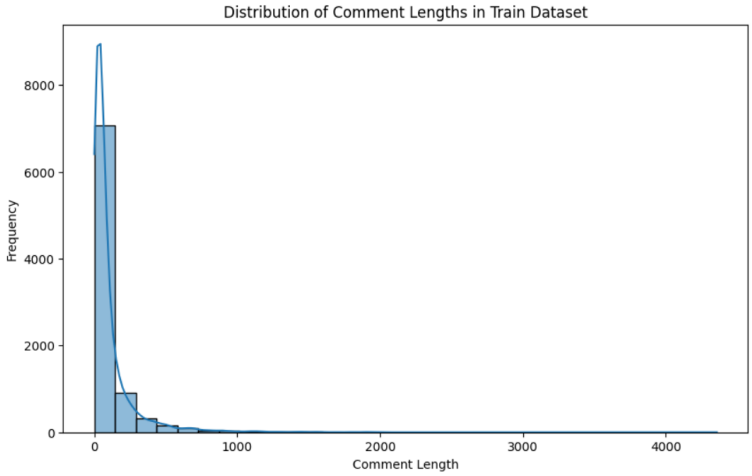
1. **Noise Reduction**: By removing irrelevant elements like URLs and punctuation, the data becomes cleaner and more focused.
2. **Uniformity**: Converting text to lower case and lemmatizing ensures that similar words are treated the same, reducing redundancy.
3. **Feature Relevance**: Removing stop words and focusing on meaningful tokens helps the model to learn from the most relevant features.
4. **Dimensionality Reduction**: Simplifying the text reduces the number of unique tokens, making the data more manageable and improving computational efficiency.
5. **Improved Model Performance**: Cleaner and more relevant data leads to better feature extraction, which in turn enhances the accuracy and robustness of the classification model.

By applying these pre-processing steps, we can significantly improve the quality of your textual data, making it more suitable for classification tasks in NLP. This ultimately leads to more accurate and reliable models.

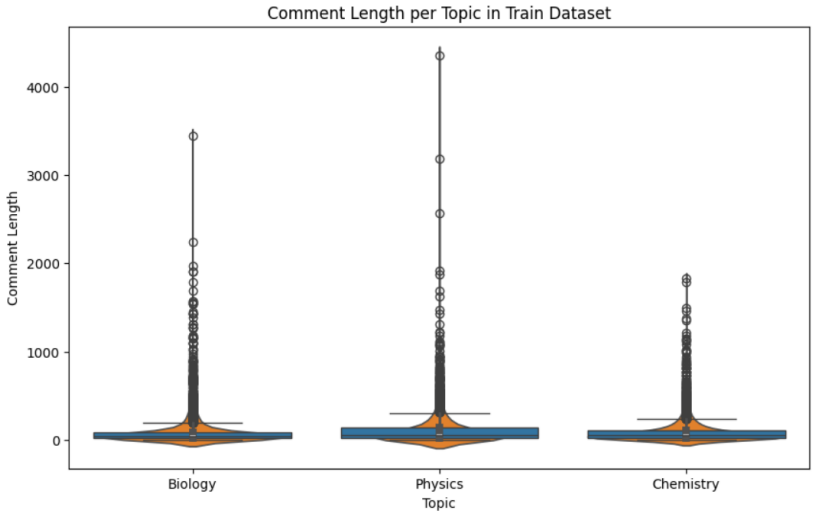
5 EDA on Processed Data:

5.1 Train Data: Distribution of Words per comment and Count Plot of Topics.

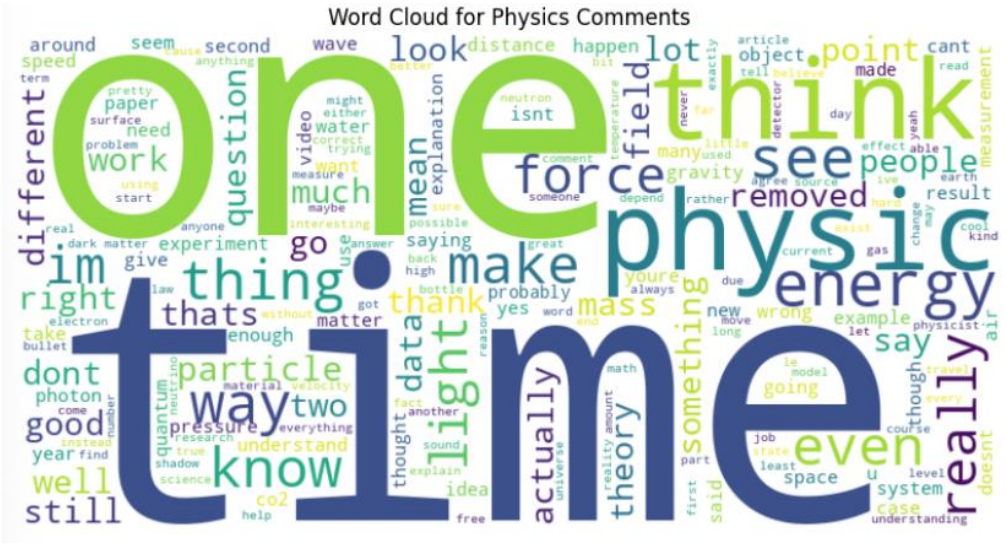


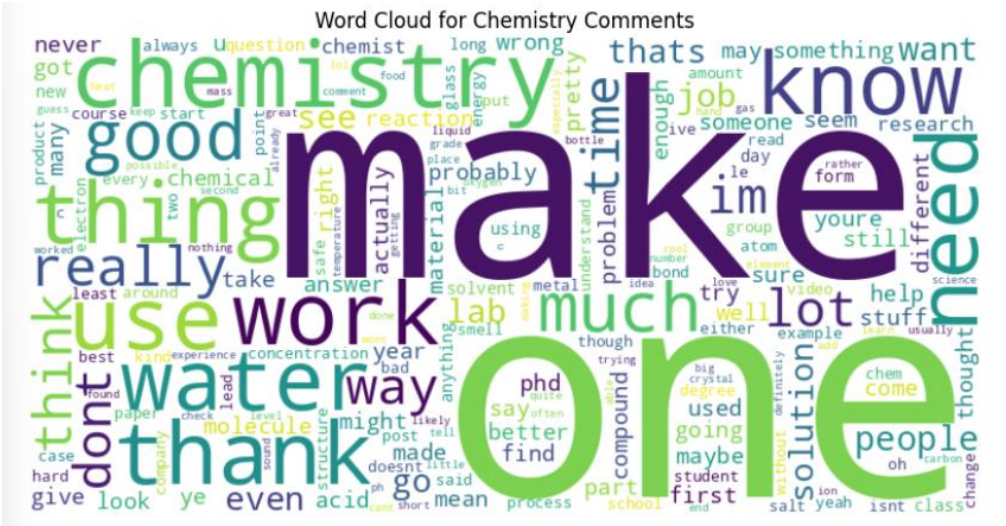


5.2 Comment Length Analysis



5.3 Word Cloud Visualization





6 Vectorization of Processed Text Data.

6.1 TF-IDF Vectorizer: TF-IDF, which stands for Term Frequency-Inverse Document Frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used in the fields of text mining and information retrieval as a weighting factor during text analysis, especially in the preprocessing steps for machine learning and deep learning models for tasks like classification, search, and topic modelling.

6.1.1 Components of TF-IDF

6.1.2 **Term Frequency (TF)**: This measures how frequently a term occurs in a document. There are several ways to calculate term frequency, with the simplest being the raw count of a term in a document. Alternatively, term frequency can be adjusted to account for document length (e.g., the raw count divided by the total number of words in the document).

6.1.3 **Inverse Document Frequency (IDF)**: This measures how important a term is within the entire corpus. The idea is that certain terms like "is" or "and" may appear a lot of times across documents but have little importance. Thus, the IDF of a term is calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term. This diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.

6.1.4 Formula for TF-IDF

The TF-IDF value is calculated by multiplying TF and IDF:

$$\text{TF-IDF}(t,d)=\text{TF}(t,d)\times\text{IDF}(t)$$

where:

- t represents the term,
- d represents the document,
- TF(t,d) is the term frequency of t in d,
- IDF(t) is the inverse document frequency of t.

6.1.5 **Usage of TF-IDF** TF-IDF is widely used for:

6.1.5.1 **Feature generation**: Transforming text into a meaningful vector (or set of numbers) that can be used in machine learning modelling.

6.1.5.2 **Search engines**: Ranking the relevance of a document based on the query terms.

6.1.5.3 **Document similarity and clustering**: Identifying how similar two documents are to each other which can be

useful in tasks like plagiarism detection or grouping similar documents together.

6.1.5.4 **Keyword extraction**: Identifying terms that are particularly characteristic of a document within a collection.

TF-IDF is a simple yet powerful and effective approach for converting textual data into a structured, numerical format that machine learning algorithms can manipulate, thereby bridging the gap between the unstructured nature of text and the structured demands of algorithmic processing.

6.2 N Grams: N-gram vectorization is a technique used in text processing where the frequency of N-grams in a text is converted into a vector format. This vector representation is suitable for use in various machine learning algorithms, especially for tasks like classification, clustering, and retrieval. The method essentially transforms text data into numerical data while preserving some contextual information that is lost when using simpler methods like unigram vectorization.

6.2.1 How N-gram Vectorization Works:

6.2.1.1 **Generating N-grams**: First, the N-grams are generated from the text. This involves sliding a window of size 'n' over the text data to capture continuous sequences of 'n' items (usually words or characters). For example, with bigrams (2-grams) of the sentence "The cat sat on the mat", the bigrams would be: ["The cat", "cat sat", "sat on", "on the", "the mat"].

6.2.1.2 **Constructing the Vocabulary**: Once N-grams are generated across all documents in a dataset, a vocabulary of all unique N-grams is constructed. Each unique N-gram becomes a feature in the feature space.

6.2.1.3 **Vector Representation**: Each document is then represented as a vector in this N-gram feature space. The vector elements can be binary (indicating the presence or absence of an N-gram), counts (indicating the frequency of each N-gram in the document), or more commonly, TF-IDF scores (which adjust the counts based on how common the N-gram is across all documents, giving more weight to rare N-grams).

6.2.2 Steps in N-gram Vectorization:

6.2.2.1 **Tokenization**: Break down text into words, characters, or tokens, which will form the basis of N-grams.

6.2.2.2 **Generate N-grams**: Slide the window over tokens to generate N-grams.

6.2.2.3 **Create a Vector Model:** Each dimension of the vector corresponds to a particular N-gram from the vocabulary. The value in each dimension could be a simple count, binary value, or a TF-IDF score.

6.2.3 Applications of N-gram Vectorization:

6.2.3.1 **Text Classification**: N-gram vectors are used as features in models to classify documents into categories, like spam detection in emails.

6.2.3.2 **Sentiment Analysis**: They help capture phrases that convey sentiments which might be missed with individual word analysis.

6.2.3.3 **Information Retrieval**: N-gram vectorization can improve search accuracy by matching on phrases rather than single words.

6.2.3.4 **Language Modeling and Translation**: N-grams help in understanding the context or predicting the next item in sequence, useful in language translation services.

6.2.4 Advantages of N-gram Vectorization:

6.2.4.1 **Context Preservation:** It retains more contextual information than unigrams, as it captures the local ordering of words or characters.

6.2.4.2 **Flexibility:** It can be adapted for different levels of precision and different languages or types of text.

6.2.4.3 **Improved Accuracy:** For many NLP tasks, using N-grams results in higher accuracy because they capture more about the structure and usage of language in text.

6.2.5 Challenges:

6.2.5.1 **Dimensionality**: The number of possible N-grams grows exponentially with 'n', leading to very high-dimensional feature spaces, which can be computationally challenging and may lead to sparse data problems.

6.2.5.2 **Computational Cost**: More memory and processing power are required to handle large vocabularies of N-grams.

6.2.5.3 **Overfitting**: Higher order N-grams can lead to overfitting on training data, as they might capture noise rather than the underlying pattern intended for learning.

N-gram vectorization is a powerful method in NLP, providing a balance between capturing meaningful linguistic patterns and maintaining a manageable computational complexity, especially when combined with techniques like TF-IDF and dimensionality reduction methods.

7 Model Training and Prediction on Vectorised Data

7.1 Multinomial Naïve Bayes with Both Vectorizers

```
Accuracy without pre-processing: 0.7148114075436982
Classification Report without pre-processing:
              precision    recall  f1-score   support

     Biology       0.65      0.89      0.75       904
   Chemistry       0.75      0.64      0.69       740
     Physics       0.90      0.51      0.65       530

    accuracy                           0.71      2174
   macro avg       0.77      0.68      0.70      2174
weighted avg       0.75      0.71      0.71      2174
```

```
Accuracy with pre-processing: 0.8278688524590164
Classification Report with pre-processing:
              precision    recall  f1-score   support

     Biology       0.87      0.82      0.85       614
   Chemistry       0.77      0.84      0.81       506
     Physics       0.84      0.83      0.83       466

    accuracy                           0.83      1586
   macro avg       0.83      0.83      0.83      1586
weighted avg       0.83      0.83      0.83      1586
```

```
Accuracy with bigrams: 0.8423707440100883
Classification Report with bigrams:
              precision    recall  f1-score   support

     Biology       0.85      0.88      0.86       614
   Chemistry       0.80      0.83      0.82       506
     Physics       0.89      0.80      0.84       466

    accuracy                           0.84      1586
   macro avg       0.85      0.84      0.84      1586
weighted avg       0.84      0.84      0.84      1586
```

7.2 Ridge Regression Model

```
Accuracy: 0.6979823455233292
Classification Report:
              precision    recall  f1-score   support

     Biology       0.94      0.62      0.74       614
   Chemistry       0.52      0.92      0.66       506
     Physics       0.92      0.56      0.70       466

    accuracy                           0.70      1586
   macro avg       0.79      0.70      0.70      1586
weighted avg       0.80      0.70      0.71      1586
```

7.3 Logistic Regression Model

```
Logistic Regression Accuracy: 0.7009775733179988
              precision    recall  f1-score   support

     Biology       0.69      0.79      0.74       725
   Chemistry       0.67      0.68      0.67       589
     Physics       0.77      0.59      0.67       425

    accuracy                           0.70      1739
   macro avg       0.71      0.68      0.69      1739
weighted avg       0.71      0.70      0.70      1739
```

7.4 Support Vector Machine Model

```
SVM Accuracy: 0.7124784358826912
              precision    recall  f1-score   support

     Biology       0.71      0.81      0.75       725
   Chemistry       0.67      0.71      0.69       589
     Physics       0.83      0.55      0.66       425

    accuracy                           0.71      1739
   macro avg       0.74      0.69      0.70      1739
weighted avg       0.72      0.71      0.71      1739
```

7.5 Random forest Model

Random Forest Accuracy: 0.6658999424956872

	precision	recall	f1-score	support
Biology	0.70	0.73	0.71	725
Chemistry	0.61	0.68	0.64	589
Physics	0.71	0.55	0.62	425
accuracy			0.67	1739
macro avg	0.67	0.65	0.66	1739
weighted avg	0.67	0.67	0.66	1739

7.6 CNN – Convoluted Neural Network Model.

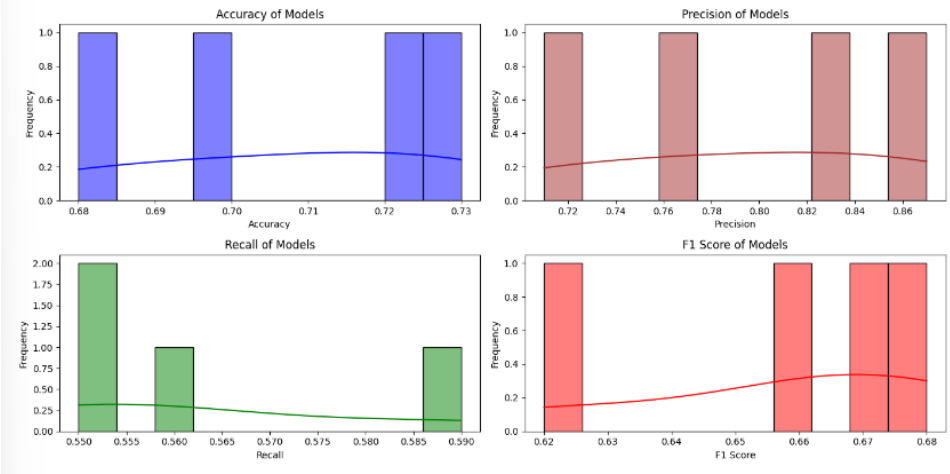
CNN Accuracy: 0.6940770557791834

	precision	recall	f1-score	support
Biology	0.76	0.72	0.74	725
Chemistry	0.65	0.68	0.67	589
Physics	0.66	0.67	0.66	425
accuracy			0.69	1739
macro avg	0.69	0.69	0.69	1739
weighted avg	0.70	0.69	0.69	1739

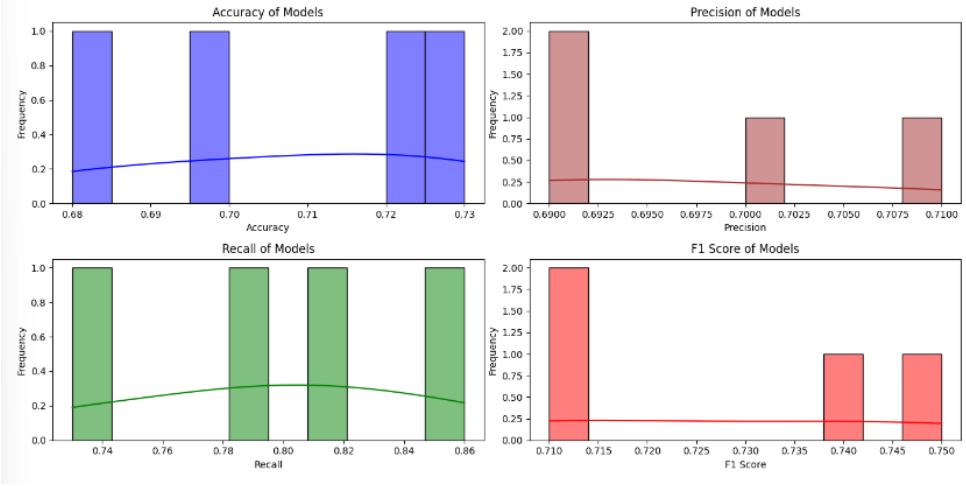
8 Results Comparison:

8.1 Topic Wise Histograms for Accuracy, Precision, Recall and F1 Score of Models for Physics, Chemistry and Biology.

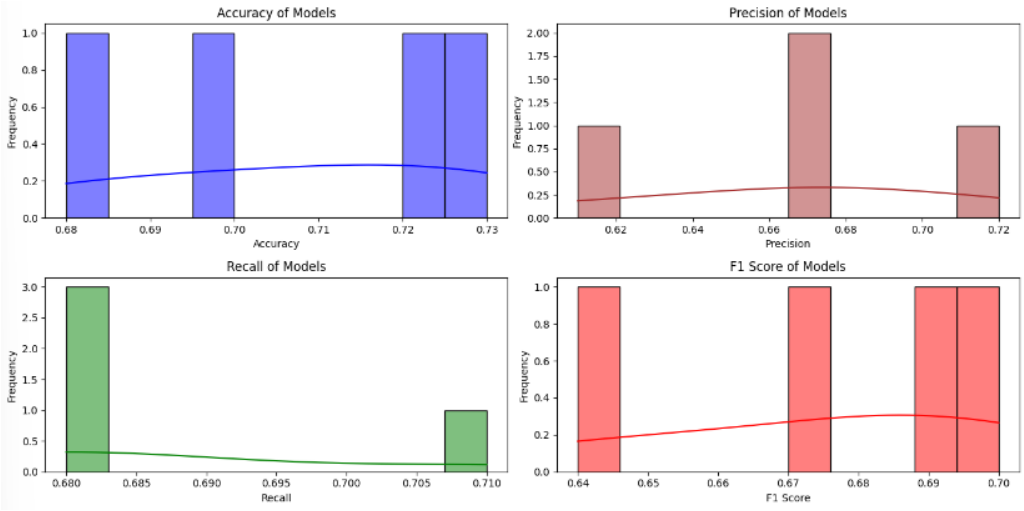
Physics



Biology

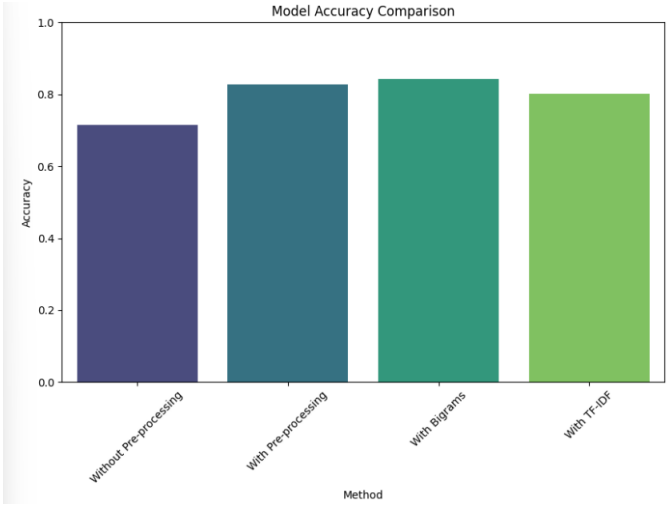


Chemistry

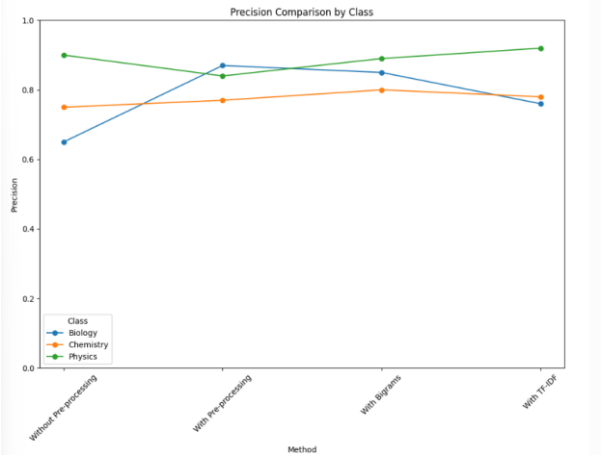


8.2 Model Accuracy Comparison: Histograms comparing the Accuracy, Precision, Recall and F1 Score by class

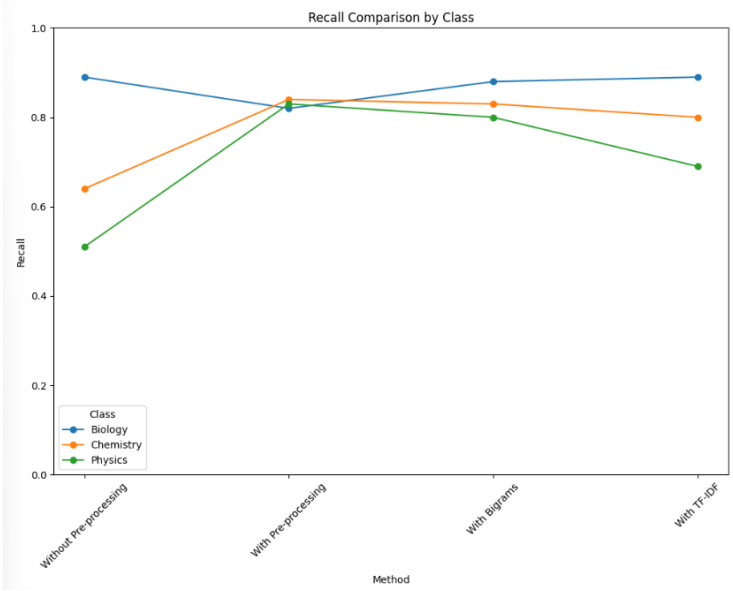
Model Accuracy comparison



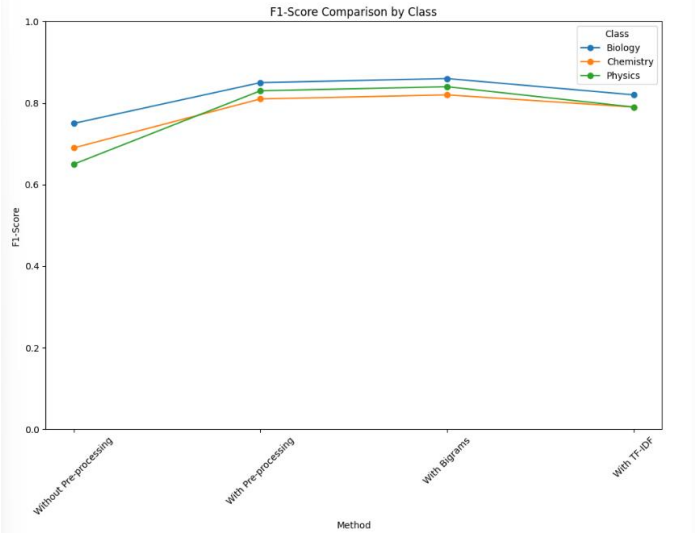
Precision comparison



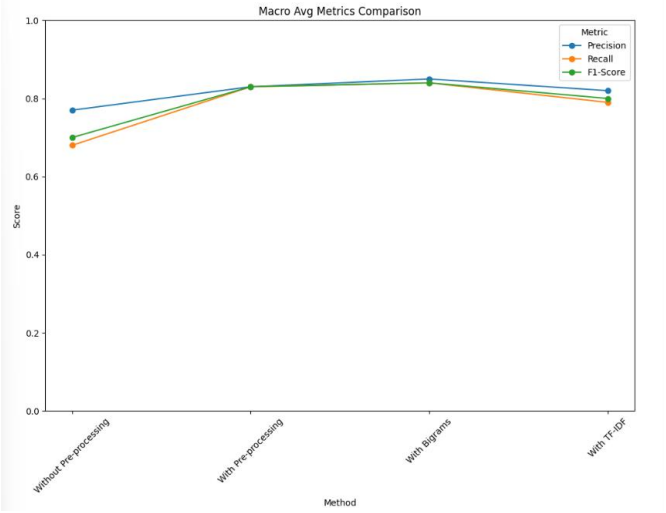
Recall comparison



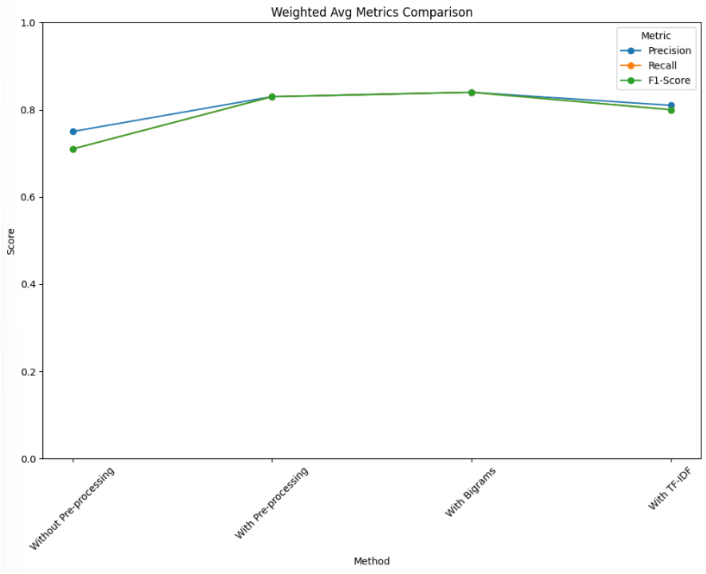
F1 Score comparison



Macro Average Metrics Comparison



Weighted Average Metric Comparison



9 Conclusion:

It can be seen that the Multinomial Native bayes model with N grams vectorization has the best accuracy score of all the model studies. We have used Count vectorization, TF-IDF and N grams as the text vectorization methods. N grams is applied only with Multinomial NB method. Improved accuracy scores could have been obtained if N-grams with processing was applied in conjunction with other models such as CNN.

10 References:

1. Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4th ed.). Pearson.
2. Kaggle Data Set URL: Physics vs Chemistry vs Biology (kaggle.com)

11 Git hub link:

pchoprasandiego/NLP-Dataset-for-Physics-Chemistry-Biology (github.com)

12 Code Annexure:

In [ ]:
```
from numpy.random import seed

import tensorflow as tf

seed(1234)

tf.random.set_seed(seed = 1234)
```

In [1]:
```
from google.colab import drive

drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

EDA ON RAW DATA

In [ ]:
```
## imports

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

from sklearn import metrics

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
from IPython.core.display import display, HTML
display(HTML("<style>div.output_scroll { height: 70em; }</style>"))
```

In [ ]:
```
sns.set(style= 'darkgrid',
        color_codes=True,
        font = 'Arial',
        font_scale= 1.5,
        rc={'figure.figsize':(12,8)})
```

In [2]:
```
import pandas as pd

file_Path_Train = '/content/drive/MyDrive/train.csv'
file_Path_Test = '/content/drive/MyDrive/test.csv'
train_data = pd.read_csv(file_Path_Train)
test_data = pd.read_csv(file_Path_Test)
```

In [5]:
```
# Display the first few rows of the training data

train_data.head()
```

Out[5]:

	Id	Comment	Topic
0	0x840	A few things. You might have negative- frequen...	Biology
1	0xbf0	Is it so hard to believe that there exist part...	Physics
2	0x1dfc	There are bees	Biology
3	0xc7e	I'm a medication technician. And that's alot o...	Biology
4	0xbba	Cesium is such a pretty metal.	Chemistry

In [6]:
```
# Display the first few rows of the test data

test_data.head()
```

Out[6]:

	Id	Comment	Topic
0	0x1aa9	Personally I have no idea what my IQ is. I've ...	Biology
1	0x25e	I'm skeptical. A heavier lid would be needed t...	Physics
2	0x1248	I think I have 100 cm of books on the subject....	Biology
3	0x2b9	Is chemistry hard in uni. Ive read somewhere t...	Chemistry
4	0x24af	In addition to the other comment, you can crit...	Physics

In [7]:
```
# Check the shape of the datasets
print("\nShape of Training Data:", train_data.shape)
print("Shape of Test Data:", test_data.shape)
```

```
Shape of Training Data: (8695, 3)
Shape of Test Data: (1586, 3)
```

In [ ]:
```
# Check for missing values
print("\nMissing Values in Training Data:")
print(train_data.isnull().sum())

print("\nMissing Values in Test Data:")
print(test_data.isnull().sum())
```

```
Missing Values in Training Data:
Id         0
Comment    0
Topic      0
dtype: int64

Missing Values in Test Data:
Id         0
Comment    0
Topic      0
dtype: int64
```

In [ ]:
```
# Get basic statistics of the training data
print("\nBasic Statistics of Training Data:")
```

```
print(train_data.describe())
```

```
Basic Statistics of Training Data:
           Id    Comment    Topic
count    8695       8695     8695
unique   8695       7949        3
top     0x840  [removed]  Biology
freq        1        114     3591
```

In [ ]:
```
print("\nTrain Data Types:")
print(train_data.dtypes)

print("\nTest Data Types:")
print(test_data.dtypes)
```

```
Train Data Types:
Id         object
Comment    object
Topic      object
dtype: object

Test Data Types:
Id         object
Comment    object
Topic      object
dtype: object
```

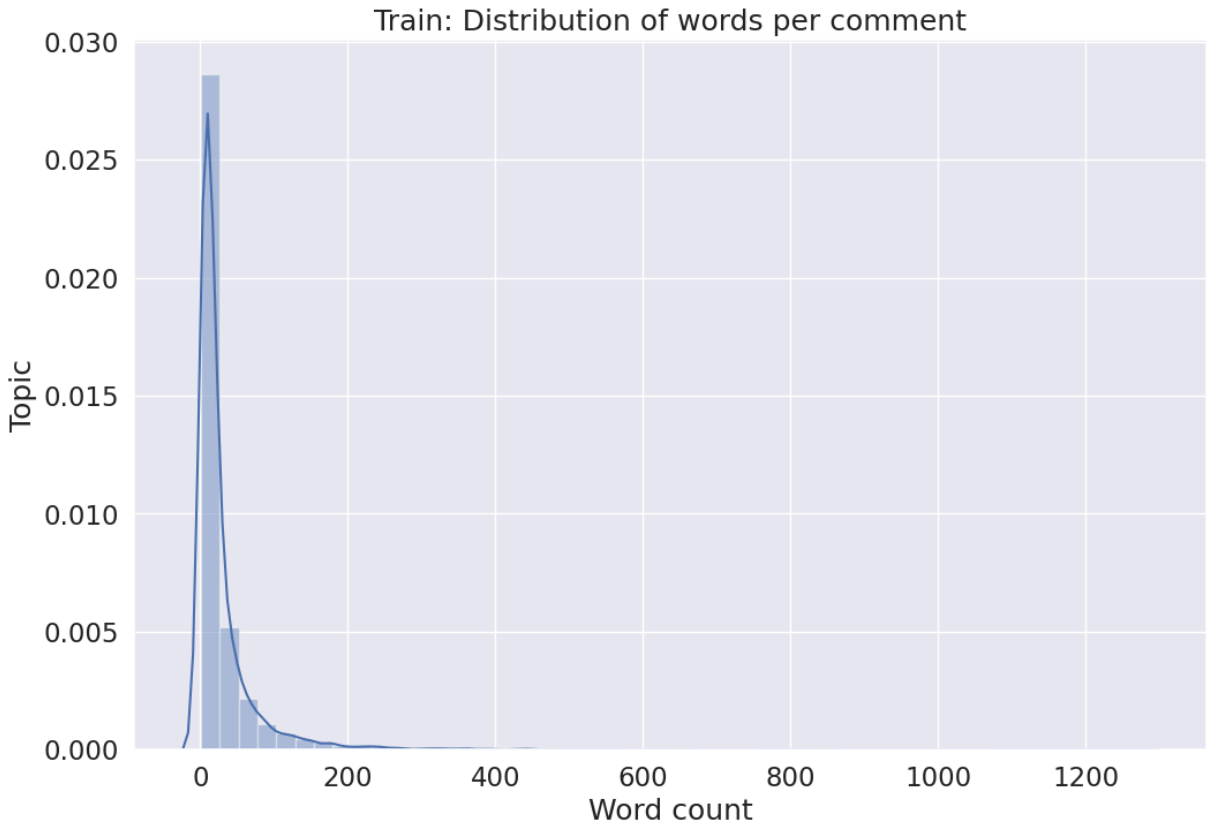
In [ ]:
```
words_per_comments_train = train_data.Comment.map(lambda x : len(x.split()))
min(words_per_comments_train),max(words_per_comments_train)
```

Out[ ]: (1, 1274)

In [ ]:
```
print(train_data.columns.tolist())
print(test_data.columns.tolist())
```

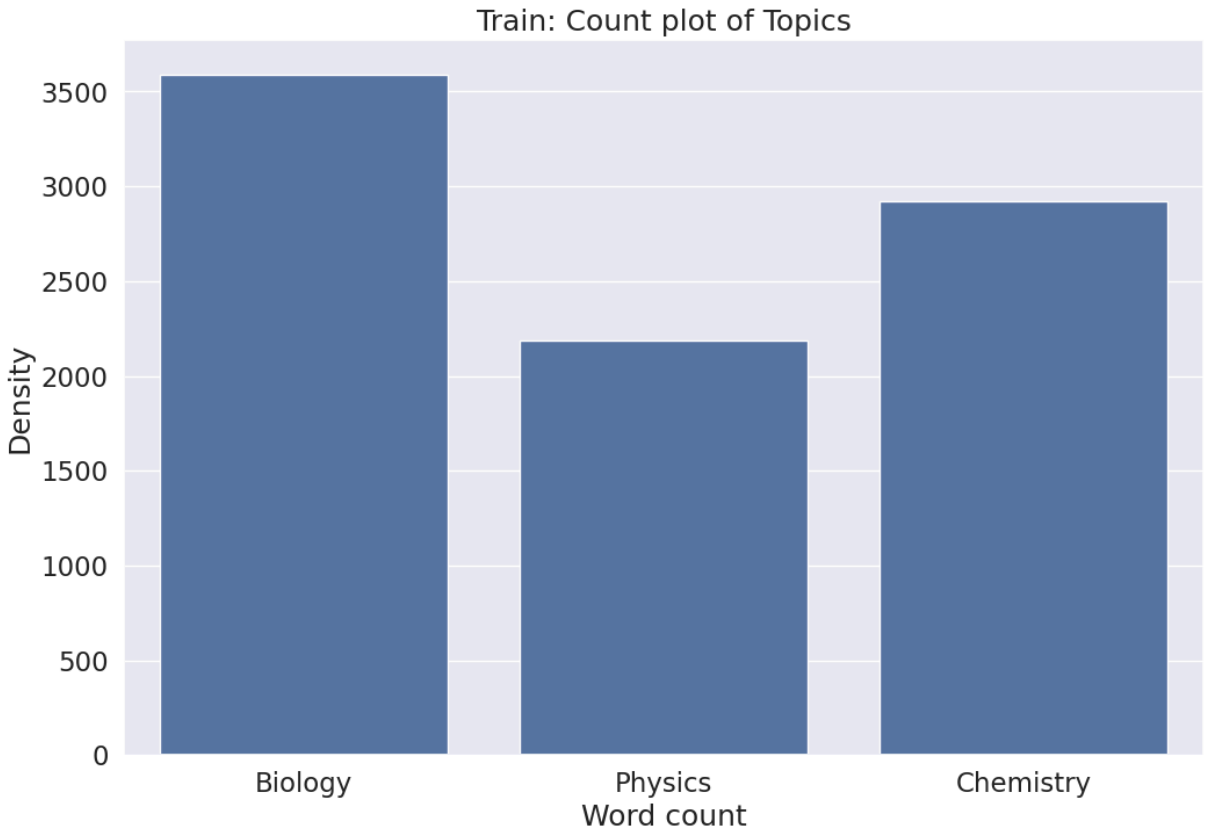
```
['Id', 'Comment', 'Topic']
['Id', 'Comment', 'Topic']
```

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.font_manager as font_manager
import seaborn as sns
sns.set(style= 'darkgrid',
       color_codes=True,
       font_scale= 1.5,
       rc={'figure.figsize':(12,8)})
sns.distplot(words_per_comments_train)
plt.title("Train: Distribution of words per comment")
plt.xlabel("Word count")
plt.ylabel("Topic")
plt.show()
```



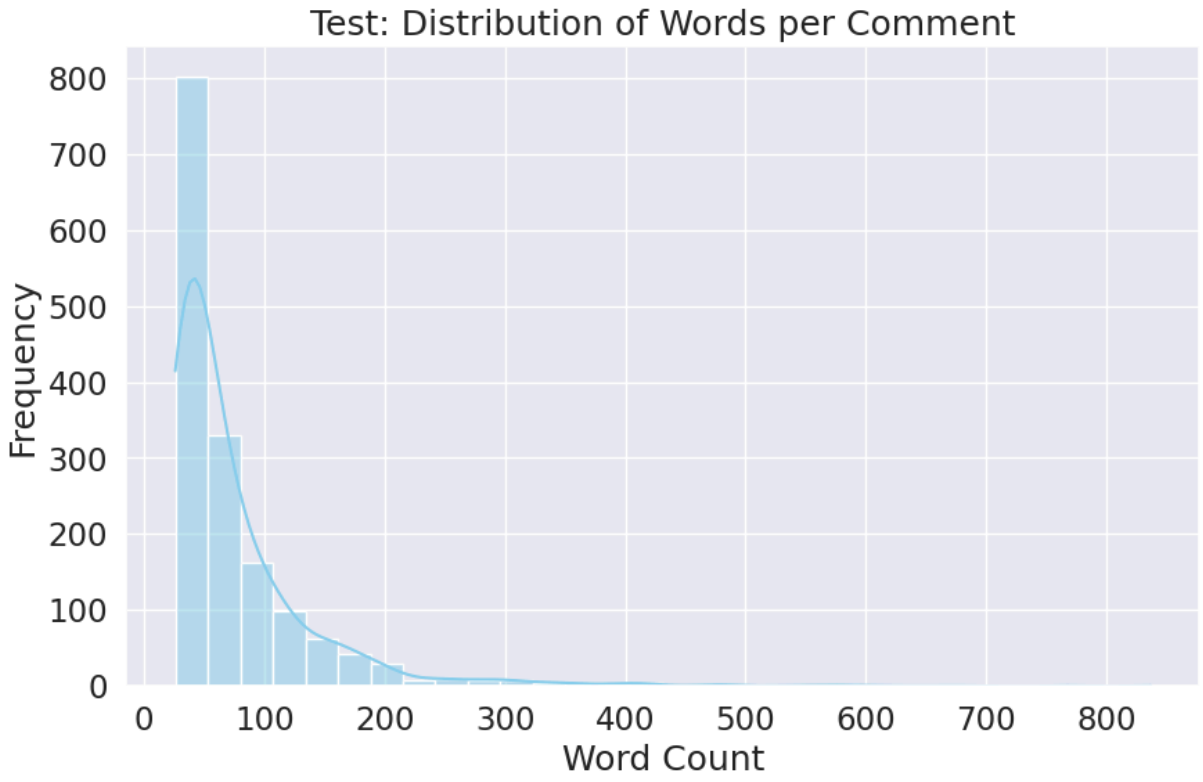
In [ ]:
```
import matplotlib.pyplot as plt
words_per_comments_test = test_data.Comment.map(lambda x : len(x.split()))
sns.distplot(words_per_comments_test)
plt.title("Test: Distribution of words per comment")
plt.xlabel("Word count")
words_per_comments_test = train_data.Comment.map(lambda x : len(x.split()))
sns.countplot(x='Topic', data=train_data)
plt.title("Train: Count plot of Topics")
```

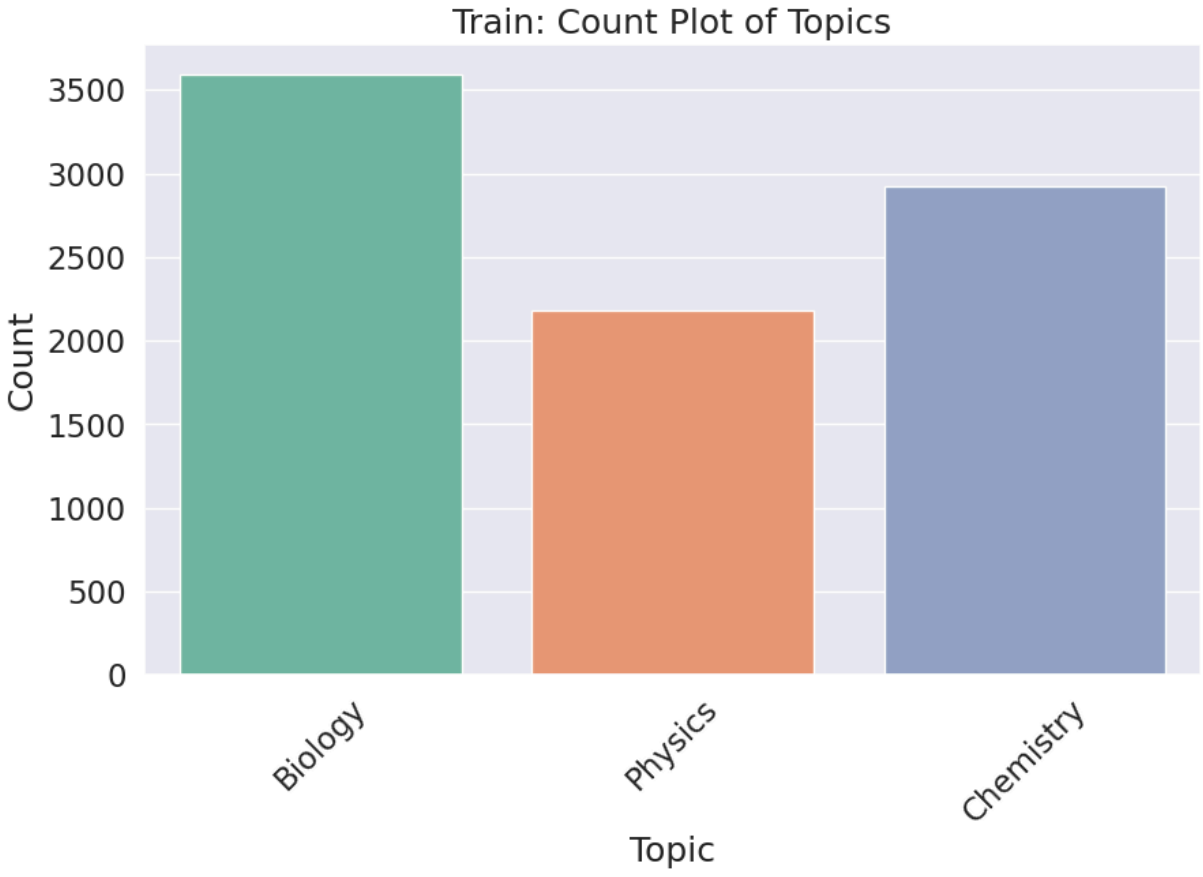
Out[ ]:
```
Text(0.5, 1.0, 'Train: Count plot of Topics')
```



In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate words per comment for the test data
words_per_comments_test = test_data.Comment.map(lambda x: len(x.split()))

# Create a distribution plot with customized colors
plt.figure(figsize=(10, 6))
sns.histplot(words_per_comments_test, kde=True, color='skyblue', bins=30)  # Use hi
plt.title("Test: Distribution of Words per Comment")
plt.xlabel("Word Count")
plt.ylabel("Frequency")
plt.show()

# Calculate words per comment for the train data (if needed)
words_per_comments_train = train_data.Comment.map(lambda x: len(x.split()))

# Create a count plot for the 'Topic' column in the train data
plt.figure(figsize=(10, 6))
sns.countplot(x='Topic', data=train_data, palette='Set2')  # Use a color palette fo
plt.title("Train: Count Plot of Topics")
plt.xlabel("Topic")
plt.ylabel("Count")
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

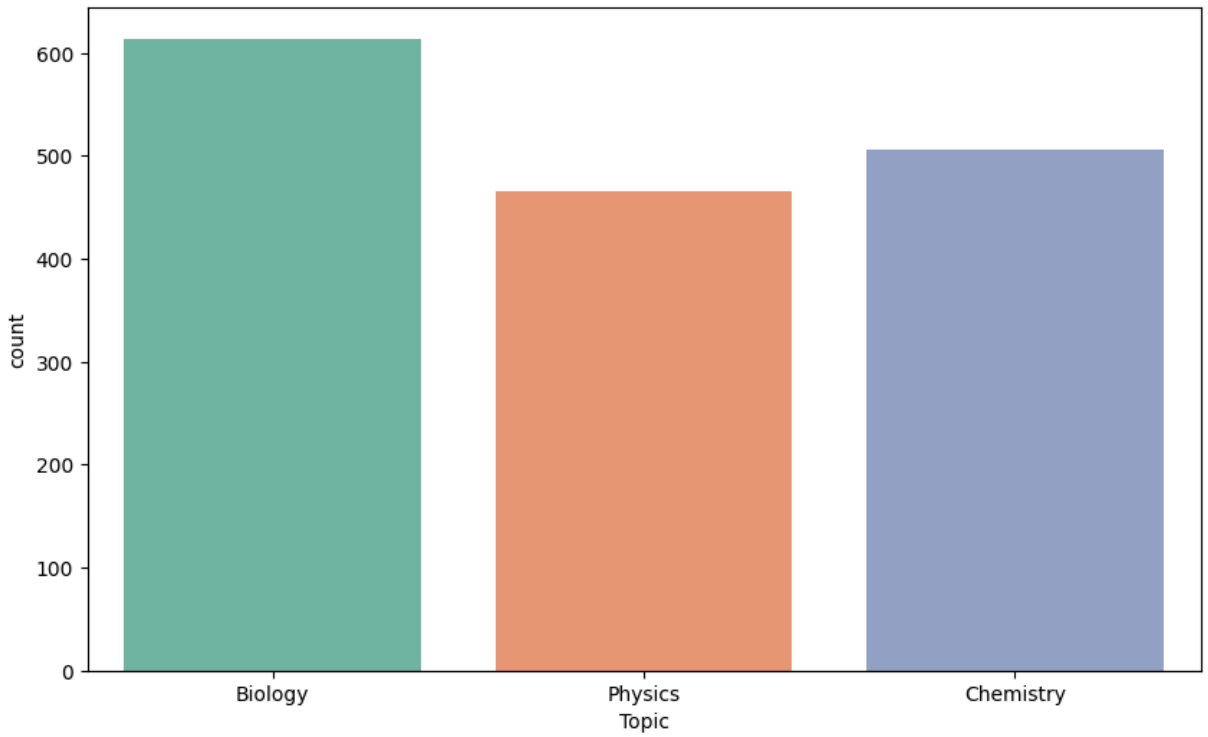




In [6]:
```
import seaborn as sns
import matplotlib.pyplot as plt
# Create a count plot for the 'Topic' column in the test data
plt.figure(figsize=(10, 6))
sns.countplot(x='Topic', data=test_data, palette='Set2')
```

```
<ipython-input-6-ac351707195f>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='Topic', data=test_data, palette='Set2')
```

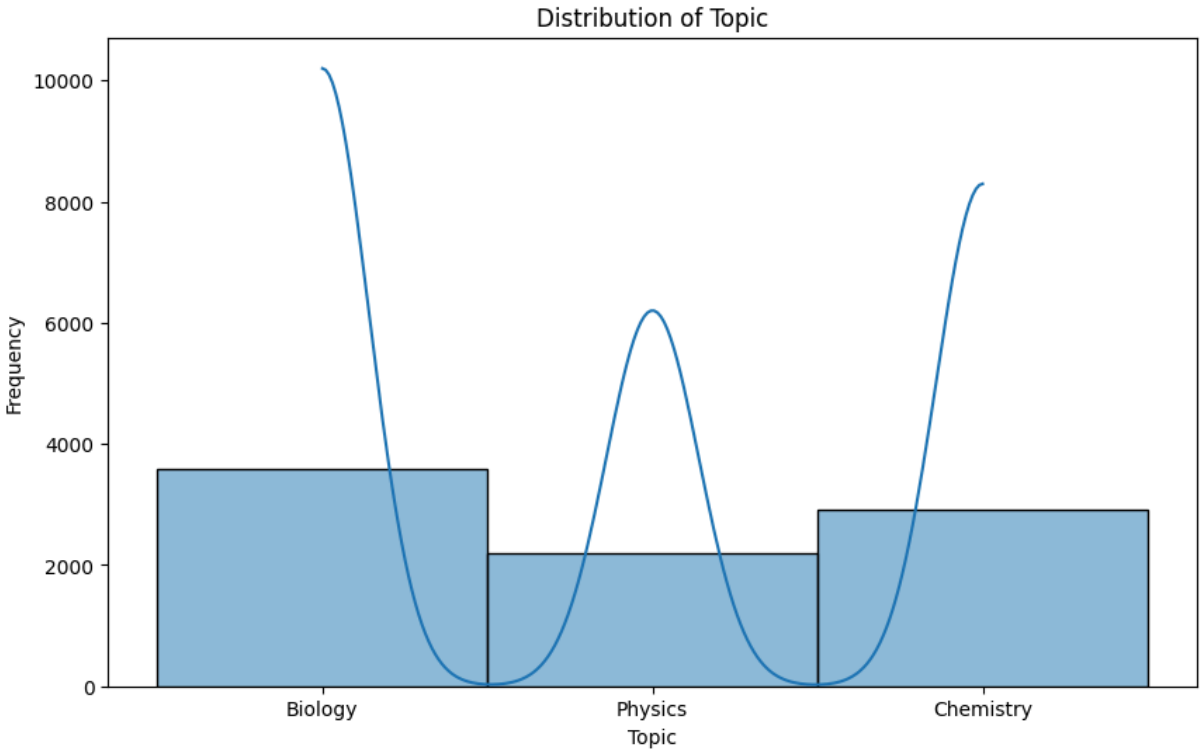
Out[6]: <Axes: xlabel='Topic', ylabel='count'>



In [ ]:
```
print(train_data.dtypes)
```

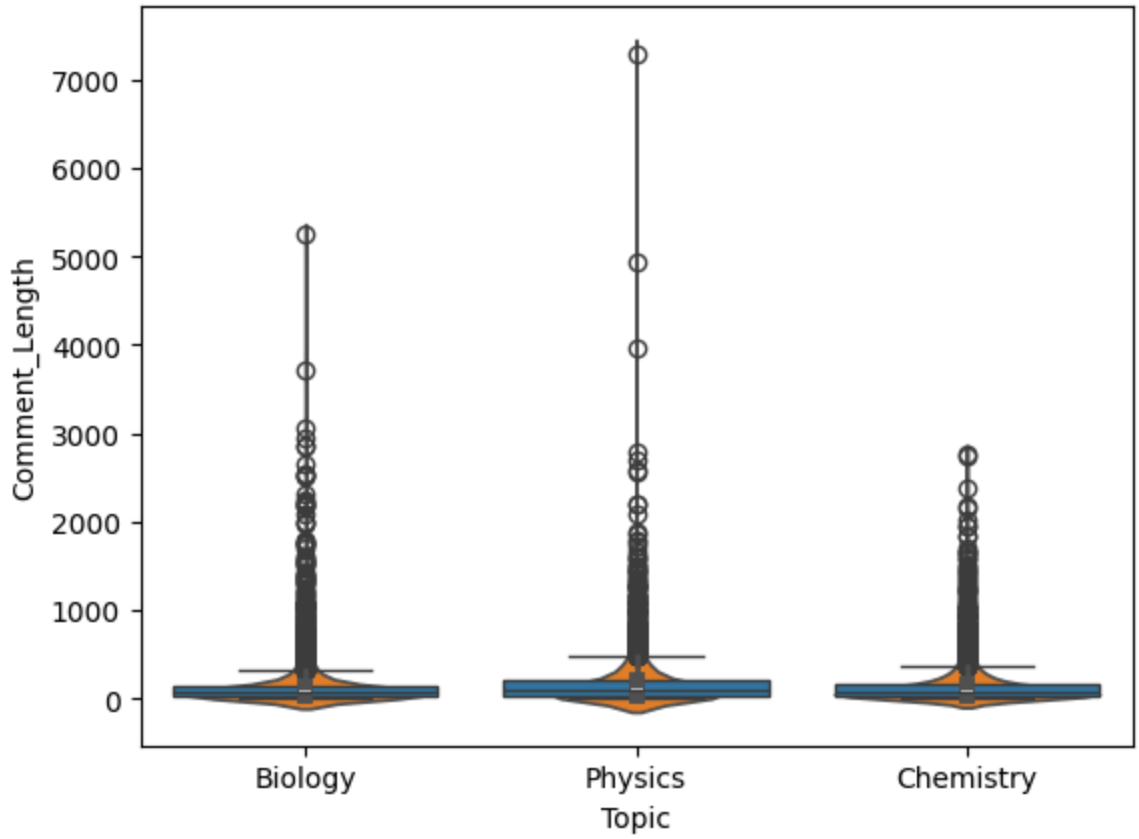
```
Id         object
Comment    object
Topic      object
dtype: object
```

In [12]:
```
import seaborn as sns
import matplotlib.pyplot as plt
train_data['Comment_Length'] = train_data['Comment'].apply(len)
test_data['Comment_Length'] = test_data['Comment'].apply(len)
# Plot a histogram for each column
#for col in train_data.columns:
plt.figure(figsize=(10, 6))
sns.histplot(train_data['Topic'], kde=True) # Select a specific column since iterat

plt.title(f'Distribution of Topic') # Update title to reflect column selected.
plt.xlabel('Topic') # Update xlabel to reflect column selected.
plt.ylabel('Frequency')
plt.show()
```



In [13]:
```
sns.boxplot(data=train_data, x='Topic', y='Comment_Length') # Update xlabel to refl
sns.violinplot(x="Topic", y="Comment_Length", data=train_data)
```

Out[13]:
```
<Axes: xlabel='Topic', ylabel='Comment_Length'>
```



In [17]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
## Create the vectorizer
tfidf = TfidfVectorizer(stop_words='english')

## fit the vectorizer on train data
tfidf.fit(train_data.Comment)
```

Out[17]:
```
TfidfVectorizer(stop_words='english')
```

In [22]:
```
features = tfidf.transform(train_data.Comment).toarray()
features.shape
```

Out[22]:
```
(8695, 17887)
```

In [23]:
```
#Make predictions on test data
test_data_actual = test_data.Topic
test_data_features = tfidf.transform(test_data.Comment).toarray()
test_data_features.shape
```

Out[23]:
```
(1586, 17887)
```

In [20]:
```
from sklearn.naive_bayes import MultinomialNB
label = train_data.Topic
clf = MultinomialNB()
clf.fit(features, label)
```

Out[20]:
```
MultinomialNB()
```

In [21]:
```
test_data_prediction = clf.predict(test_data_features)
from sklearn import metrics
#Check performence on test data
test_data_score = 100*(metrics.accuracy_score(test_data_actual , test_data_predicti

print(test_data_score)
```

```
81.84110970996217
```

This was the baseline score, various other machine learning/deep learning models can be developed to achieve better accuracy

In [28]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score

# Prepare data
X = train_data['Comment']
y = train_data['Topic']
```

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st

# Bag of Words
tfidf = TfidfVectorizer(stop_words='english')
X_train_bow = tfidf.fit_transform(X_train)
X_test_bow = tfidf.transform(X_test)

# Train Naive Bayes model
model = MultinomialNB()
model.fit(X_train_bow, y_train)

# Make predictions
y_pred = model.predict(X_test_bow)

# Check lengths
print(f"Length of y_test: {len(y_test)}")
print(f"Length of y_pred: {len(y_pred)}")

# Evaluate
print("Accuracy without pre-processing:", accuracy_score(y_test, y_pred))
print("Classification Report without pre-processing:")
print(classification_report(y_test, y_pred))
```

```
Length of y_test: 2174
Length of y_pred: 2174
Accuracy without pre-processing: 0.7148114075436982
Classification Report without pre-processing:
              precision    recall  f1-score   support

     Biology       0.65      0.89      0.75       904
   Chemistry       0.75      0.64      0.69       740
     Physics       0.90      0.51      0.65       530

    accuracy                           0.71      2174
   macro avg       0.77      0.68      0.70      2174
weighted avg       0.75      0.71      0.71      2174
```

Overall, this code implements a basic text classification pipeline using the Bag of Words model and a Naive Bayes classifier. It prepares the data, splits it into training and testing sets, transforms the text data into a numerical format, trains the model, makes predictions, and evaluates the model's performance. This is a common approach in natural language processing tasks.

In [23]:

In [ ]:
```
# select Physics
train_data_Physics = train_data[train_data['Topic'] == 'Physics']
print(train_data_Physics.shape)
# select Chemistry
train_data_Chemistry = train_data[train_data['Topic'] == 'Chemistry']
print(train_data_Chemistry.shape)
```

```
# select Biology
train_data_Biology = train_data[train_data['Topic'] == 'Biology']
print(train_data_Biology.shape)
```

```
(2184, 3)
(2920, 3)
(3591, 3)
```

In [31]:
```
from sklearn.model_selection import train_test_split

# Assuming df_set7 is the filtered DataFrame
# Define the features (X) and the target variable (y)
# Ensure the column you're trying to drop actually exists in the DataFrame
X = train_data.drop(labels=['Topic'], axis=1)  # Features (all columns except 'Topi
y = train_data['Topic']                 # Target variable (adjust if needed)

# Create a train/test split
X_train_data, X_test_data, y_train_data, y_test_data = train_test_split(X, y, test_

# Check the shapes of the resulting datasets
print(f"Training set size: {X_train_data.shape[0]} samples")
print(f"Test set size: {X_test_data.shape[0]} samples")
```

```
Training set size: 6956 samples
Test set size: 1739 samples
```

In [30]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
import pandas as pd

# Sample data (replace this with your actual data loading)
Xtrain = pd.DataFrame({'Comment': ["This is a sample Comment.", "Another comment fo
Xtest = pd.DataFrame({'Comment': ["A test comment.", "One more for testing."]})

# Sample target scores (replace with your actual scores)
ytrain = pd.Series([1, 3])  # Example scores for training comments
ytest = pd.Series([2, 4])    # Example scores for test comments

# Initialize the TfidVectorizer
tfidf = TfidfVectorizer(stop_words='english')

# Fit the vectorizer on the training comments
tfidf.fit(Xtrain['Comment'])

# Transform the training and test comments into vectors
xtrain = tfidf.transform(Xtrain['Comment'])
xtest = tfidf.transform(Xtest['Comment'])

# Initialize the Ridge regression model
ridge_model = Ridge()

# Fit the model on the training data
ridge_model.fit(xtrain, ytrain)

# Evaluate the model on the test data
r2_score = ridge_model.score(xtest, ytest)
```

```
# Report the coefficient of determination (R² score)
print("Coefficient of Determination (R² score):", r2_score)
```

```
Coefficient of Determination (R² score): -1.0
```

In [2]:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Ridge
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score

# Load the train and test data from CSV files
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Extract comments
Xtrain = train_data['Comment']
Xtest = test_data['Comment']

# Encode the target topics
le = LabelEncoder()
ytrain = le.fit_transform(train_data['Topic'])
ytest = le.transform(test_data['Topic'])

# Initialize the TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')

# Fit the vectorizer on the training comments
tfidf.fit(Xtrain)

# Transform the training and test comments into vectors
xtrain = tfidf.transform(Xtrain)
xtest = tfidf.transform(Xtest)

# Initialize the Ridge regression model
ridge_model = Ridge()

# Fit the model on the training data
ridge_model.fit(xtrain, ytrain)

# Predict the labels for the test data
y_pred = ridge_model.predict(xtest)

# Convert continuous predictions to categorical by rounding
y_pred_rounded = y_pred.round().astype(int)

# Ensure that the predicted labels are within the valid range of labels
y_pred_rounded = np.clip(y_pred_rounded, 0, len(le.classes_) - 1)

# Evaluate the model
accuracy = accuracy_score(ytest, y_pred_rounded)
report = classification_report(ytest, y_pred_rounded, target_names=le.classes_)
```

```
print("Accuracy:", accuracy)
print("Classification Report:\n", report)
```

```
Accuracy: 0.6979823455233292
Classification Report:
               precision    recall  f1-score   support

      Biology       0.94      0.62      0.74       614
    Chemistry       0.52      0.92      0.66       506
      Physics       0.92      0.56      0.70       466

     accuracy                           0.70      1586
    macro avg       0.79      0.70      0.70      1586
weighted avg       0.80      0.70      0.71      1586
```

In [4]:

```
# Word frequency analysis
from collections import Counter  # Import the Counter class
from wordcloud import WordCloud  # Import the WordCloud class
import matplotlib.pyplot as plt

# Preprocess the text data
def preprocess_text(text):
    text = text.lower()
    text = ''.join([char for char in text if char.isalnum() or char.isspace()])
    return text

train_data['Clean_Comment'] = train_data['Comment'].apply(preprocess_text)
test_data['Clean_Comment'] = test_data['Comment'].apply(preprocess_text)

all_words = ' '.join(train_data['Clean_Comment'])
word_freq = Counter(all_words.split())

# Top 20 most common words
most_common_words = word_freq.most_common(20)
print("\nTop 20 most common words in train dataset:")
print(most_common_words)

# Word cloud visualization
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all

plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Train Dataset Comments')
plt.show()

# Most common words per topic
for topic in train_data['Topic'].unique():
    topic_words = ' '.join(train_data[train_data['Topic'] == topic]['Clean_Comment'
    topic_word_freq = Counter(topic_words.split())
    topic_most_common_words = topic_word_freq.most_common(20)
    print(f"\nTop 20 most common words in {topic} comments:")
    print(topic_most_common_words)

    wordcloud = WordCloud(width=800, height=400, background_color='white').generate
```

```
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f'Word Cloud for {topic} Comments')
    plt.show()

print("\nEDA complete.")
```

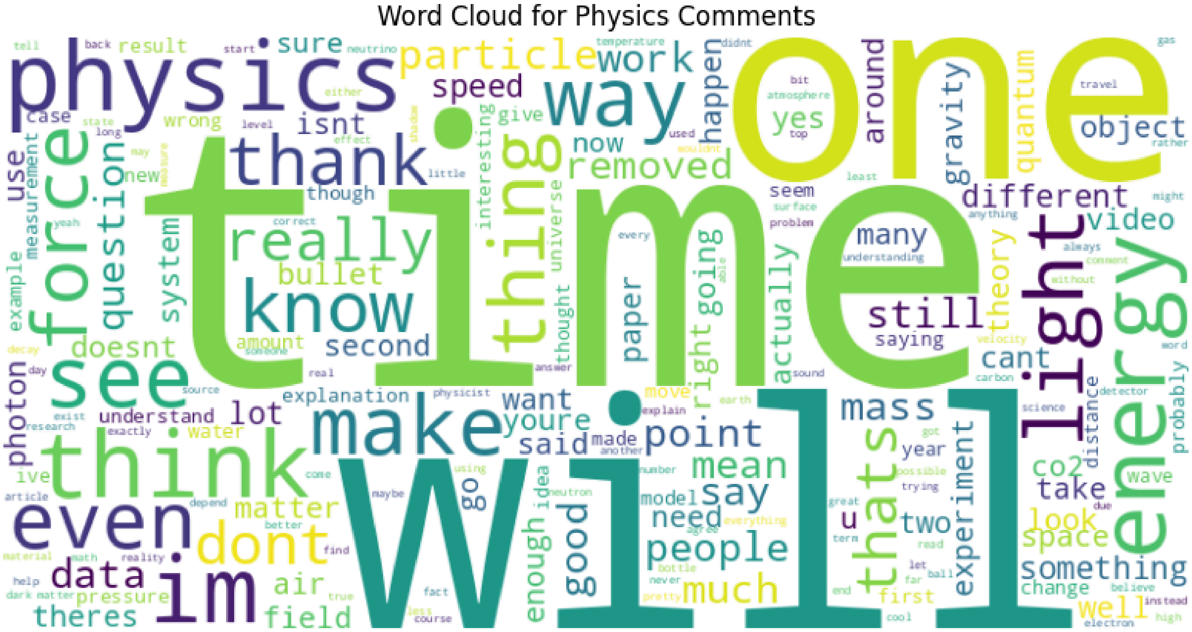
```
Top 20 most common words in train dataset:
[('the', 10886), ('to', 6334), ('a', 6323), ('of', 5060), ('and', 4902), ('is', 436
3), ('you', 3980), ('it', 3968), ('i', 3738), ('in', 3663), ('that', 3372), ('for',
2225), ('are', 2082), ('be', 1934), ('but', 1811), ('this', 1796), ('not', 1736),
('with', 1631), ('its', 1596), ('if', 1514)]
```


```
Top 20 most common words in Biology comments:
[('the', 3865), ('a', 2483), ('to', 2481), ('and', 2088), ('of', 1931), ('it', 155
4), ('you', 1544), ('i', 1516), ('is', 1495), ('in', 1399), ('that', 1310), ('are',
942), ('for', 891), ('be', 759), ('its', 755), ('but', 750), ('this', 724), ('not',
697), ('have', 630), ('if', 616)]
```

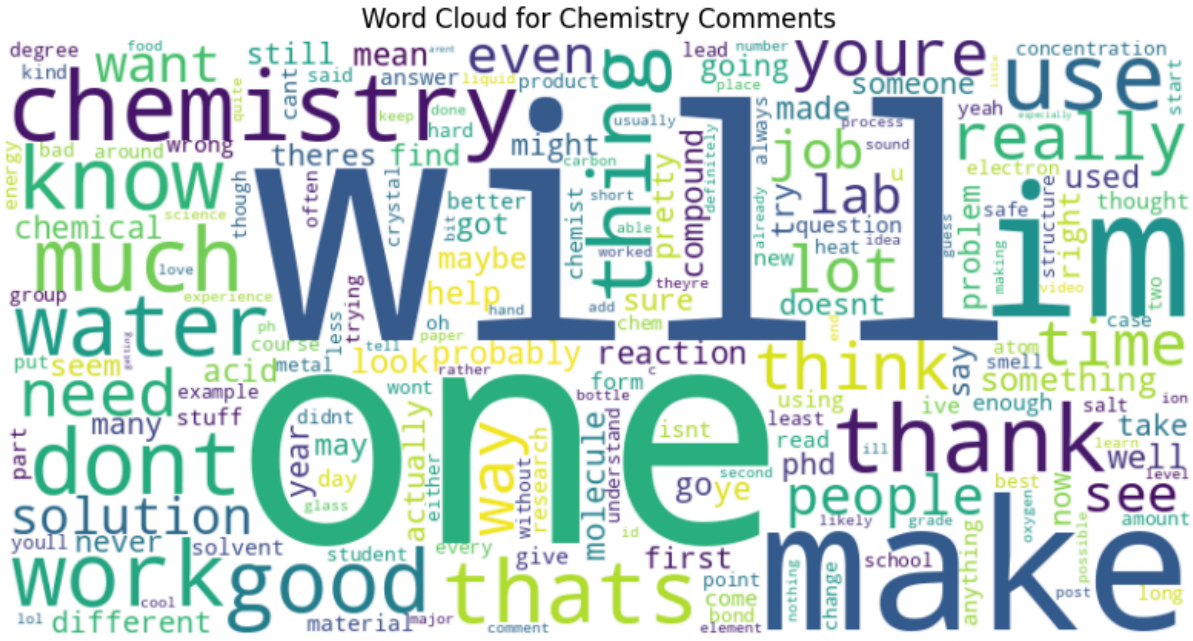
Word Cloud for Biology Comments

Top 20 most common words in Physics comments:
[('the', 3802), ('to', 1833), ('a', 1788), ('of', 1655), ('is', 1523), ('and', 125
3), ('that', 1104), ('it', 1072), ('in', 1014), ('i', 972), ('you', 958), ('this', 6
15), ('be', 569), ('for', 541), ('but', 516), ('are', 504), ('not', 486), ('on', 44
9), ('with', 445), ('as', 432)]

Word Cloud for Physics Comments



Top 20 most common words in Chemistry comments:
[('the', 3219), ('a', 2052), ('to', 2020), ('and', 1561), ('you', 1478), ('of', 147
4), ('is', 1345), ('it', 1342), ('i', 1250), ('in', 1250), ('that', 958), ('for', 79
3), ('are', 636), ('be', 606), ('with', 605), ('not', 553), ('but', 545), ('if', 50
4), ('its', 474), ('have', 470)]



EDA complete.

Data Preprocessing and EDA of Preprocessed Data

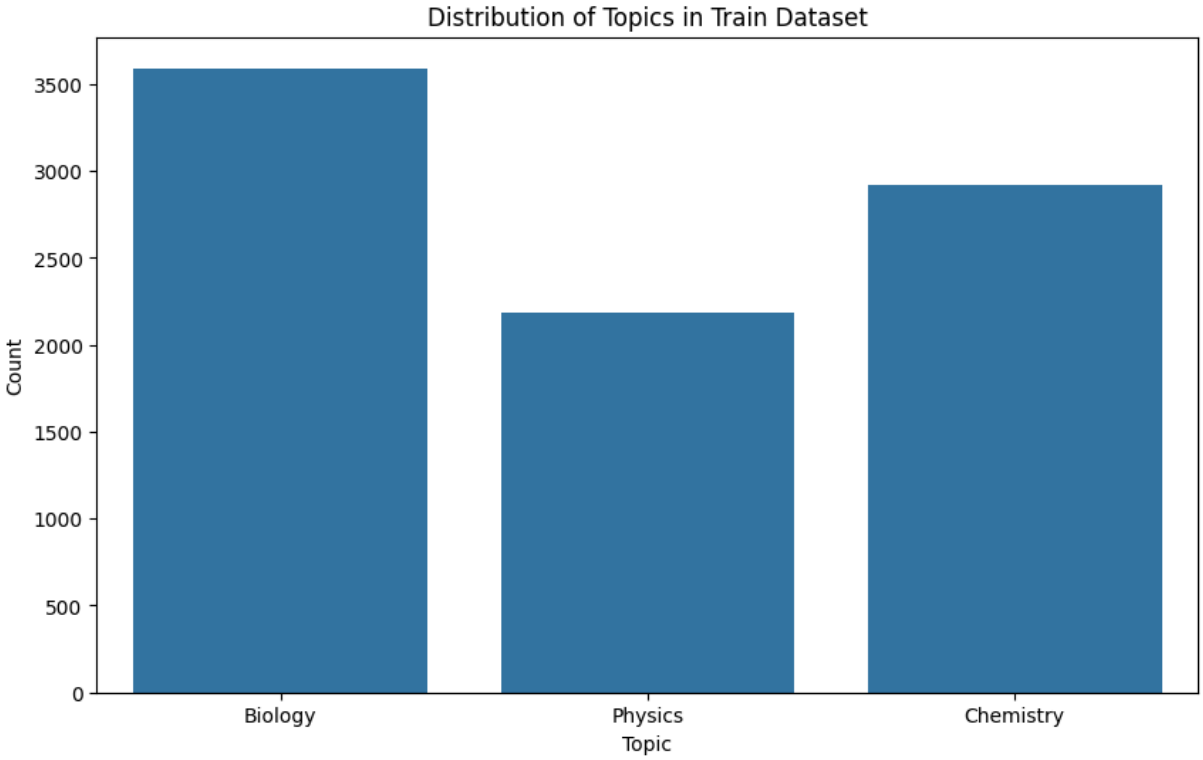
In [15]:
```
# EDA on Processed Data
# We can compare the Effects of data Processing

import pandas as pd
import re
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from collections import Counter

# Uncomment the following lines if running for the first time
#nltk.download('punkt')
#nltk.download('stopwords')
#nltk.download('wordnet')

# Load datasets
train_data = pd.read_csv(file_Path_Train)
test_data = pd.read_csv(file_Path_Test)

# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Define a function for text preprocessing
def preprocess_text(text):
```

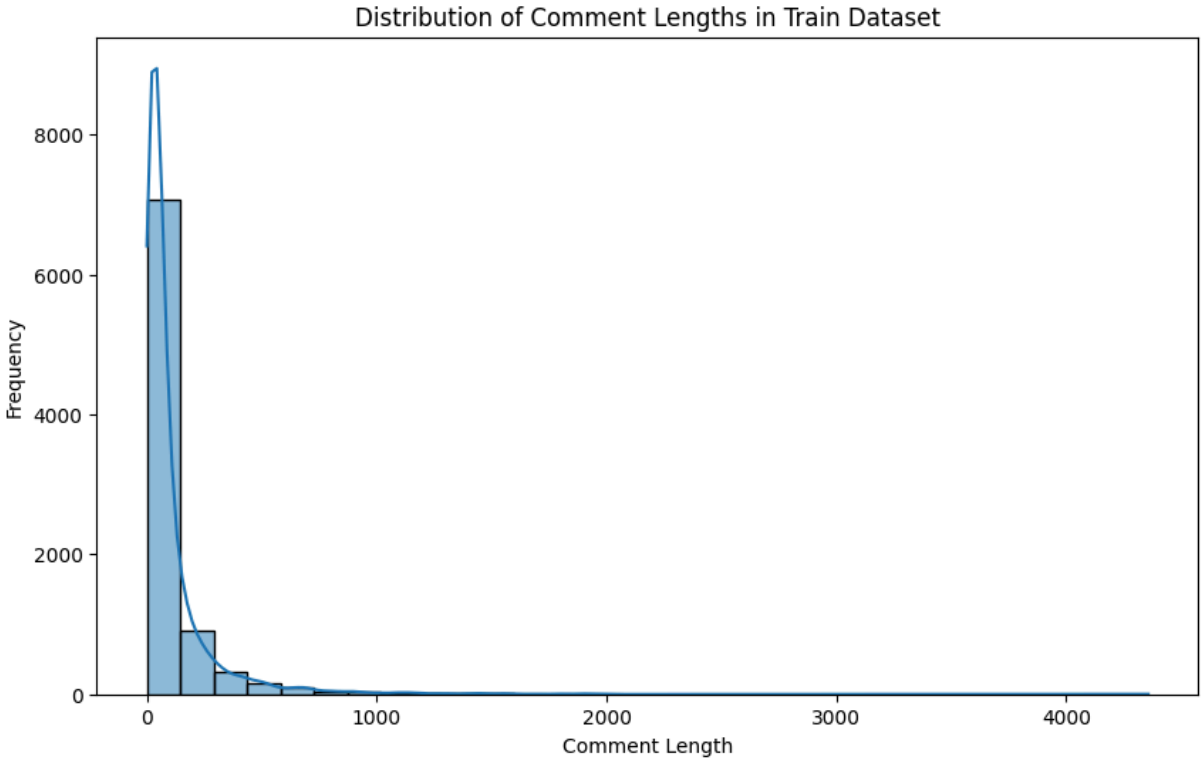
```
    if pd.isnull(text):
        return ""
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stop words
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    # Lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)

# Apply the preprocessing function to the 'Comment' column
train_data['Processed_Comment'] = train_data['Comment'].apply(preprocess_text)
test_data['Processed_Comment'] = test_data['Comment'].apply(preprocess_text)

# Display the processed DataFrame
print(train_data.head())
print(test_data.head())

# EDA
# Data overview
print("\nTrain Dataset Info:")
print(train_data.info())
print("\nTest Dataset Info:")
print(test_data.info())

print("\nTrain Dataset Description:")
print(train_data.describe())

# Distribution of topics in the train dataset
plt.figure(figsize=(10, 6))
sns.countplot(data=train_data, x='Topic')
plt.title('Distribution of Topics in Train Dataset')
plt.xlabel('Topic')
plt.ylabel('Count')
plt.show()

# Comment length analysis
train_data['Comment_Length'] = train_data['Processed_Comment'].apply(len)
test_data['Comment_Length'] = test_data['Processed_Comment'].apply(len)

plt.figure(figsize=(10, 6))
sns.histplot(train_data['Comment_Length'], kde=True, bins=30)
plt.title('Distribution of Comment Lengths in Train Dataset')
plt.xlabel('Comment Length')
plt.ylabel('Frequency')
plt.show()

# Word frequency analysis
all_words = ' '.join(train_data['Processed_Comment'])
word_freq = Counter(all_words.split())
```

```
# Top 20 most common words
most_common_words = word_freq.most_common(20)
print("\nTop 20 most common words in train dataset:")
print(most_common_words)

# Word cloud visualization
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(all

plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Train Dataset Comments')
plt.show()

# Comment length per topic
plt.figure(figsize=(10, 6))
sns.boxplot(data=train_data, x='Topic', y='Comment_Length')
sns.violinplot(x="Topic", y="Comment_Length", data=train_data)
plt.title('Comment Length per Topic in Train Dataset')
plt.xlabel('Topic')
plt.ylabel('Comment Length')
plt.show()

# Most common words per topic
for topic in train_data['Topic'].unique():
    topic_words = ' '.join(train_data[train_data['Topic'] == topic]['Processed_Comm
    topic_word_freq = Counter(topic_words.split())
    topic_most_common_words = topic_word_freq.most_common(20)
    print(f"\nTop 20 most common words in {topic} comments:")
    print(topic_most_common_words)

    wordcloud = WordCloud(width=800, height=400, background_color='white').generate
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f'Word Cloud for {topic} Comments')
    plt.show()

print("\nEDA complete.")
```

```
       Id                                            Comment      Topic  \
0   0x840  A few things. You might have negative- frequen...    Biology
1   0xbf0  Is it so hard to believe that there exist part...    Physics
2  0x1dfc                                  There are bees    Biology
3   0xc7e  I'm a medication technician. And that's alot o...    Biology
4   0xbba                Cesium is such a pretty metal.  Chemistry

                                Processed_Comment
0  thing might negative frequency dependent selec...
1  hard believe exist particular cant detect anyt...
2                                              bee
3  im medication technician thats alot drug liver...
4                            cesium pretty metal
       Id                                            Comment      Topic  \
0  0x1aa9  Personally I have no idea what my IQ is. I’ve ...    Biology
1   0x25e  I'm skeptical. A heavier lid would be needed t...    Physics
2  0x1248  I think I have 100 cm of books on the subject....    Biology
3   0x2b9  Is chemistry hard in uni. Ive read somewhere t...  Chemistry
4  0x24af  In addition to the other comment, you can crit...    Physics

                                Processed_Comment
0  personally idea iq ’ never tested however test...
1  im skeptical heavier lid would needed build pr...
2  think 100 cm book subject tldr problem conscio...
3  chemistry hard uni ive read somewhere hardest ...
4  addition comment criticize theory without chec...

Train Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8695 entries, 0 to 8694
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Id                 8695 non-null   object
 1   Comment            8695 non-null   object
 2   Topic              8695 non-null   object
 3   Processed_Comment  8695 non-null   object
dtypes: object(4)
memory usage: 271.8+ KB
None

Test Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1586 entries, 0 to 1585
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Id                 1586 non-null   object
 1   Comment            1586 non-null   object
 2   Topic              1586 non-null   object
 3   Processed_Comment  1586 non-null   object
dtypes: object(4)
memory usage: 49.7+ KB
None

Train Dataset Description:
```

	Id	Comment	Topic	Processed_Comment
count	8695	8695	8695	8695
unique	8695	7949	3	7813
top	0x840	[removed]	Biology	removed
freq	1	114	3591	114

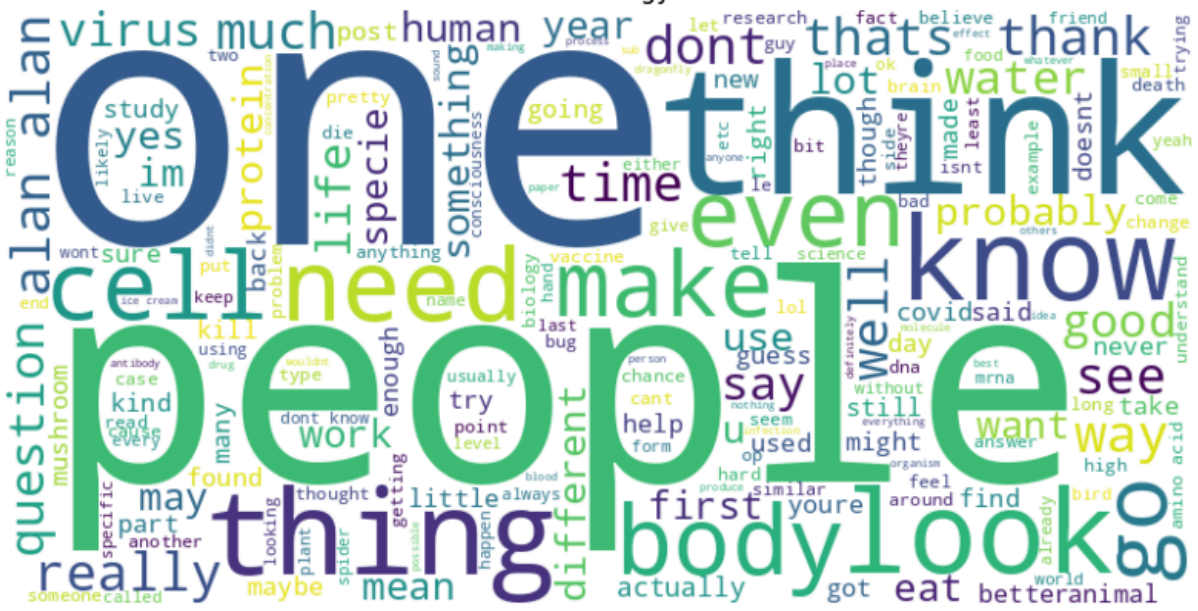




Top 20 most common words in train dataset:
[('’', 2388), ('like', 1054), ('would', 1048), ('one', 837), ('get', 718), ('think', 602), ('make', 574), ('know', 564), ('time', 548), ('also', 530), ('dont', 499), ('thing', 483), ('people', 465), ('could', 460), ('much', 456), ('im', 426), ('even', 417), ('see', 411), ('way', 409), ('need', 409)]

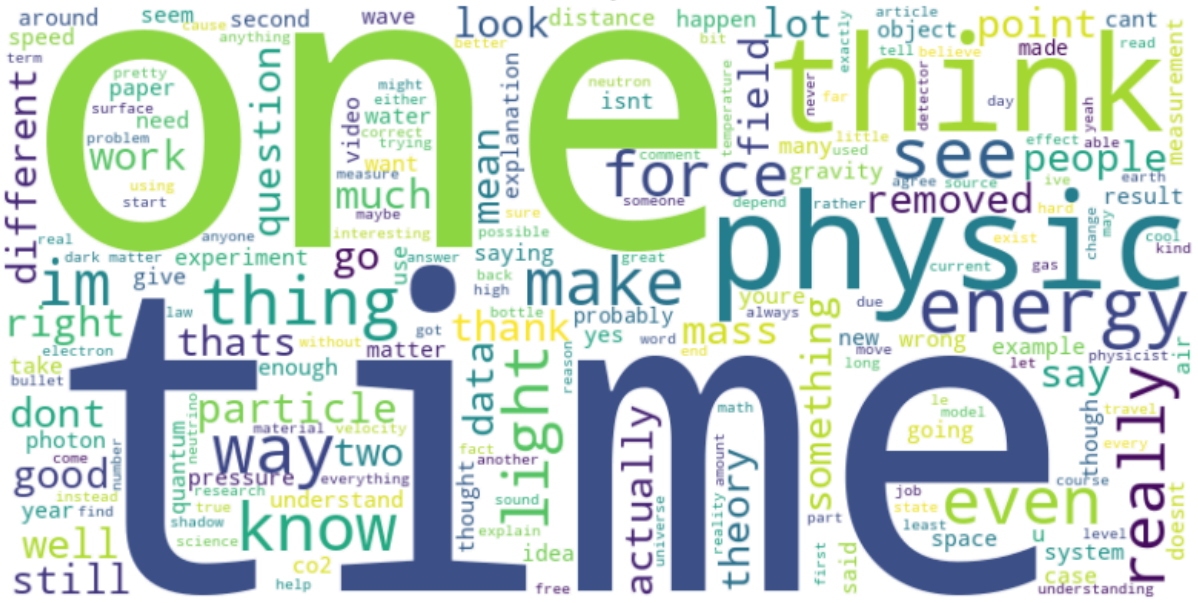
Top 20 most common words in Biology comments:
[('’', 1265), ('like', 468), ('one', 382), ('would', 379), ('get', 285), ('people', 247), ('think', 238), ('know', 237), ('cell', 209), ('thing', 203), ('also', 203), ('dont', 196), ('could', 189), ('look', 184), ('make', 177), ('even', 175), ('body', 174), ('need', 173), ('time', 160), ('much', 159)]

Word Cloud for Biology Comments



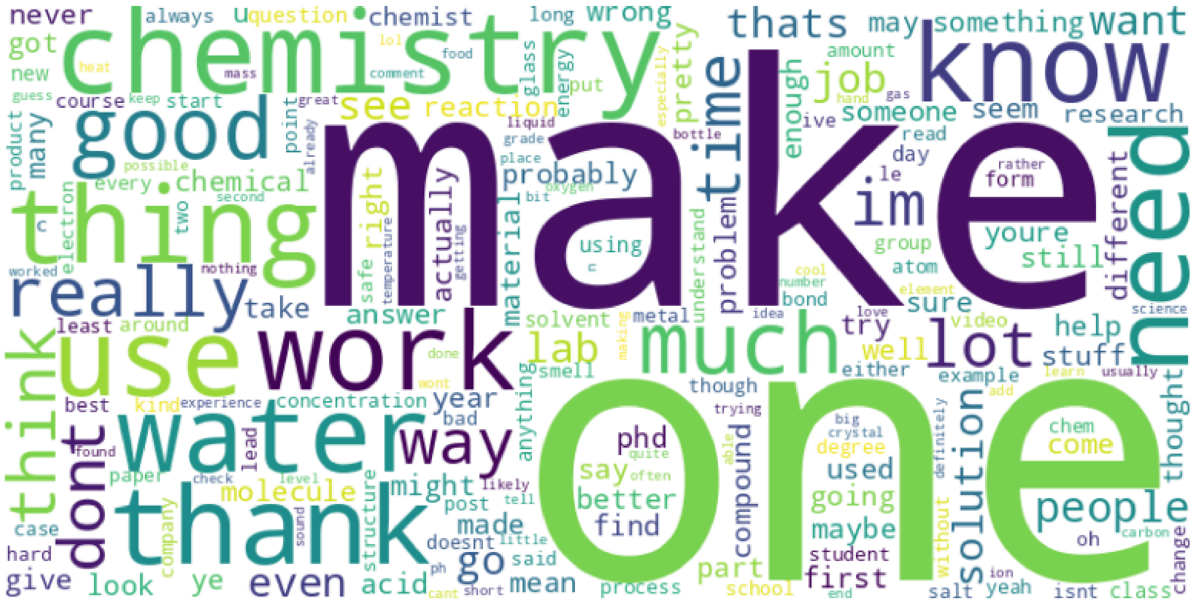
Top 20 most common words in Physics comments:
[('’', 515), ('would', 367), ('time', 249), ('like', 243), ('energy', 233), ('think', 207), ('one', 207), ('physic', 204), ('light', 178), ('get', 173), ('see', 163), ('make', 161), ('also', 154), ('force', 153), ('know', 146), ('could', 145), ('way', 139), ('even', 130), ('thing', 128), ('im', 127)]

Word Cloud for Physics Comments



Top 20 most common words in Chemistry comments:
[('’', 608), ('like', 343), ('would', 302), ('get', 260), ('one', 248), ('make', 236), ('chemistry', 229), ('much', 188), ('know', 181), ('dont', 177), ('also', 173), ('acid', 168), ('work', 164), ('water', 161), ('good', 161), ('use', 158), ('think', 157), ('need', 155), ('thing', 152), ('im', 150)]

Word Cloud for Chemistry Comments



EDA complete.

Model Training and Prediction

In [20]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, GlobalMaxPooling1D, Embedding, D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


# preprocessed data from the cell above
train_data['Processed_Comment']
test_data['Processed_Comment']


# Load processed data
#train_df = pd.read_csv('/mnt/data/train.csv')
#test_df = pd.read_csv('/mnt/data/test.csv')

# Encode the labels
label_encoder = LabelEncoder()
train_data['Encoded_Topic'] = label_encoder.fit_transform(train_data['Topic'])

# Split the training data for validation
X_train, X_val, y_train, y_val = train_test_split(
    train_data['Processed_Comment'], train_data['Encoded_Topic'], test_size=0.2, ra
```

```
# Initialize TF-IDF Vectorizer
tfidf = TfidfVectorizer(max_features=5000)

# Transform the training and validation data
X_train_tfidf = tfidf.fit_transform(X_train)
X_val_tfidf = tfidf.transform(X_val)
X_test_tfidf = tfidf.transform(test_data['Processed_Comment'])

# Logistic Regression
lr_model = LogisticRegression()
lr_model.fit(X_train_tfidf, y_train)
lr_pred = lr_model.predict(X_val_tfidf)
print("Logistic Regression Accuracy:", accuracy_score(y_val, lr_pred))
print(classification_report(y_val, lr_pred, target_names=label_encoder.classes_))

# Support Vector Machine (SVM)
svm_model = SVC()
svm_model.fit(X_train_tfidf, y_train)
svm_pred = svm_model.predict(X_val_tfidf)
print("SVM Accuracy:", accuracy_score(y_val, svm_pred))
print(classification_report(y_val, svm_pred, target_names=label_encoder.classes_))

# Random Forest
rf_model = RandomForestClassifier()
rf_model.fit(X_train_tfidf, y_train)
rf_pred = rf_model.predict(X_val_tfidf)
print("Random Forest Accuracy:", accuracy_score(y_val, rf_pred))
print(classification_report(y_val, rf_pred, target_names=label_encoder.classes_))

# Naive Bayes
nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)
nb_pred = nb_model.predict(X_val_tfidf)
print("Naive Bayes Accuracy:", accuracy_score(y_val, nb_pred))
print(classification_report(y_val, nb_pred, target_names=label_encoder.classes_))

# Convolutional Neural Network (CNN)

# Tokenization and Padding
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_val_seq = tokenizer.texts_to_sequences(X_val)
X_test_seq = tokenizer.texts_to_sequences(test_data['Processed_Comment'])

# Padding sequences to ensure uniform length
max_seq_len = 200  # Adjust this based on your dataset's average sequence length
X_train_pad = pad_sequences(X_train_seq, maxlen=max_seq_len)
X_val_pad = pad_sequences(X_val_seq, maxlen=max_seq_len)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_seq_len)

# CNN Model
cnn_model = Sequential([
    Embedding(input_dim=5000, output_dim=128, input_length=max_seq_len),
    Conv1D(filters=128, kernel_size=5, activation='relu'),
```

```
    GlobalMaxPooling1D(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(len(label_encoder.classes_), activation='softmax')
])

cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics
cnn_model.fit(X_train_pad, y_train, epochs=5, validation_data=(X_val_pad, y_val), b

cnn_pred = cnn_model.predict(X_val_pad)
cnn_pred_classes = cnn_pred.argmax(axis=1)
print("CNN Accuracy:", accuracy_score(y_val, cnn_pred_classes))
print(classification_report(y_val, cnn_pred_classes, target_names=label_encoder.cla

# Model Comparison
models = ['Logistic Regression', 'SVM', 'Random Forest', 'Naive Bayes', 'CNN']
accuracies = [
    accuracy_score(y_val, lr_pred),
    accuracy_score(y_val, svm_pred),
    accuracy_score(y_val, rf_pred),
    accuracy_score(y_val, nb_pred),
    accuracy_score(y_val, cnn_pred_classes)
]

# Display model comparison results
for model, accuracy in zip(models, accuracies):
    print(f'{model}: {accuracy:.4f}')

# Predict on the test data using the best model (Assume CNN performed the best here
best_model = cnn_model
test_pred = best_model.predict(X_test_pad)
test_pred_classes = test_pred.argmax(axis=1)
test_data['Predicted_Topic'] = label_encoder.inverse_transform(test_pred_classes)

# Save the test predictions
test_data[['Id', 'Predicted_Topic']].to_csv('/content/drive/MyDrive/test_with_predi
print("Predictions saved to 'test_with_predictions_NN.csv'.")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:460: Conve
rgenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Logistic Regression Accuracy: 0.7009775733179988
              precision    recall  f1-score   support

     Biology       0.69      0.79      0.74       725
   Chemistry       0.67      0.68      0.67       589
     Physics       0.77      0.59      0.67       425

    accuracy                           0.70      1739
   macro avg       0.71      0.68      0.69      1739
weighted avg       0.71      0.70      0.70      1739

SVM Accuracy: 0.7124784358826912
              precision    recall  f1-score   support

     Biology       0.71      0.81      0.75       725
   Chemistry       0.67      0.71      0.69       589
     Physics       0.83      0.55      0.66       425

    accuracy                           0.71      1739
   macro avg       0.74      0.69      0.70      1739
weighted avg       0.72      0.71      0.71      1739

Random Forest Accuracy: 0.6658999424956872
              precision    recall  f1-score   support

     Biology       0.70      0.73      0.71       725
   Chemistry       0.61      0.68      0.64       589
     Physics       0.71      0.55      0.62       425

    accuracy                           0.67      1739
   macro avg       0.67      0.65      0.66      1739
weighted avg       0.67      0.67      0.66      1739

Naive Bayes Accuracy: 0.7257044278320874
              precision    recall  f1-score   support

     Biology       0.69      0.86      0.77       725
   Chemistry       0.72      0.68      0.70       589
     Physics       0.87      0.56      0.68       425

    accuracy                           0.73      1739
   macro avg       0.76      0.70      0.71      1739
weighted avg       0.74      0.73      0.72      1739
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserW
arning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

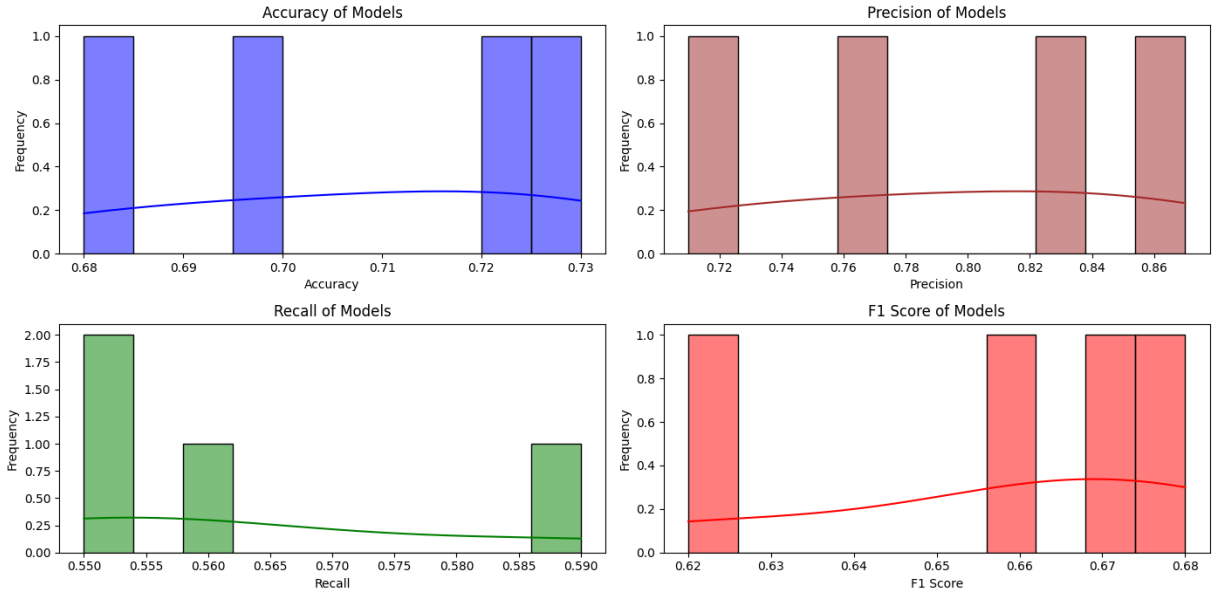
```
Epoch 1/5
218/218 ━━━━━━━━━━━━━━━━━━━━ 23s 95ms/step - accuracy: 0.4418 - loss: 1.0508 - val_a
ccuracy: 0.6671 - val_loss: 0.7649
Epoch 2/5
218/218 ━━━━━━━━━━━━━━━━━━━━ 21s 95ms/step - accuracy: 0.7428 - loss: 0.6074 - val_a
ccuracy: 0.7165 - val_loss: 0.6629
Epoch 3/5
218/218 ━━━━━━━━━━━━━━━━━━━━ 39s 85ms/step - accuracy: 0.8677 - loss: 0.3332 - val_a
ccuracy: 0.7211 - val_loss: 0.7501
Epoch 4/5
218/218 ━━━━━━━━━━━━━━━━━━━━ 20s 90ms/step - accuracy: 0.9280 - loss: 0.1927 - val_a
ccuracy: 0.7096 - val_loss: 0.9168
Epoch 5/5
218/218 ━━━━━━━━━━━━━━━━━━━━ 24s 105ms/step - accuracy: 0.9380 - loss: 0.1446 - val_
accuracy: 0.6941 - val_loss: 1.0732
55/55 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step
CNN Accuracy: 0.6940770557791834
              precision    recall  f1-score   support

     Biology       0.76      0.72      0.74       725
   Chemistry       0.65      0.68      0.67       589
     Physics       0.66      0.67      0.66       425

    accuracy                           0.69      1739
   macro avg       0.69      0.69      0.69      1739
weighted avg       0.70      0.69      0.69      1739

Logistic Regression: 0.7010
SVM: 0.7125
Random Forest: 0.6659
Naive Bayes: 0.7257
CNN: 0.6941
50/50 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step
Predictions saved to 'test_with_predictions_NN.csv'.
```

PHYSICS

In [9]:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns



# Create a DataFrame for easier plotting
import pandas as pd



# Physics data:
models = ['Logistic Regression','SVM','Random Forest', 'Naive Bayes']
precision = [0.77, 0.83, 0.71, 0.87]  # Replaced with actual precision values
recall = [0.59, 0.55, 0.55, 0.56]    # Replaced with actual recall values
f1_score = [0.67, 0.66, 0.62, 0.68]  # Replaced with actual F1 score values

# Add accuracy data - you need to get these values from your model evaluation
accuracy = [0.70, 0.72, 0.68, 0.73]
```

```python
# Create a DataFrame for easier plotting
import pandas as pd

data = {
    'Model': models,
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1_score
}
df = pd.DataFrame(data)

# Set the figure size
plt.figure(figsize=(14, 7))

# Create histplots for each metric
plt.subplot(2, 2, 1)
sns.histplot(df['Accuracy'], bins=10, color='blue', kde=True)
plt.title('Accuracy of Models')
plt.xlabel('Accuracy')
plt.ylabel('Frequency')

plt.subplot(2, 2, 2)
sns.histplot(df['Precision'], bins=10, color='brown', kde=True)
plt.title('Precision of Models')
plt.xlabel('Precision')
plt.ylabel('Frequency')

plt.subplot(2, 2, 3)
sns.histplot(df['Recall'], bins=10, color='green', kde=True)
plt.title('Recall of Models')
plt.xlabel('Recall')
plt.ylabel('Frequency')

plt.subplot(2, 2, 4)
sns.histplot(df['F1 Score'], bins=10, color='red', kde=True)
plt.title('F1 Score of Models')
plt.xlabel('F1 Score')
plt.ylabel('Frequency')

# Adjust layout
plt.tight_layout()
plt.show()
```

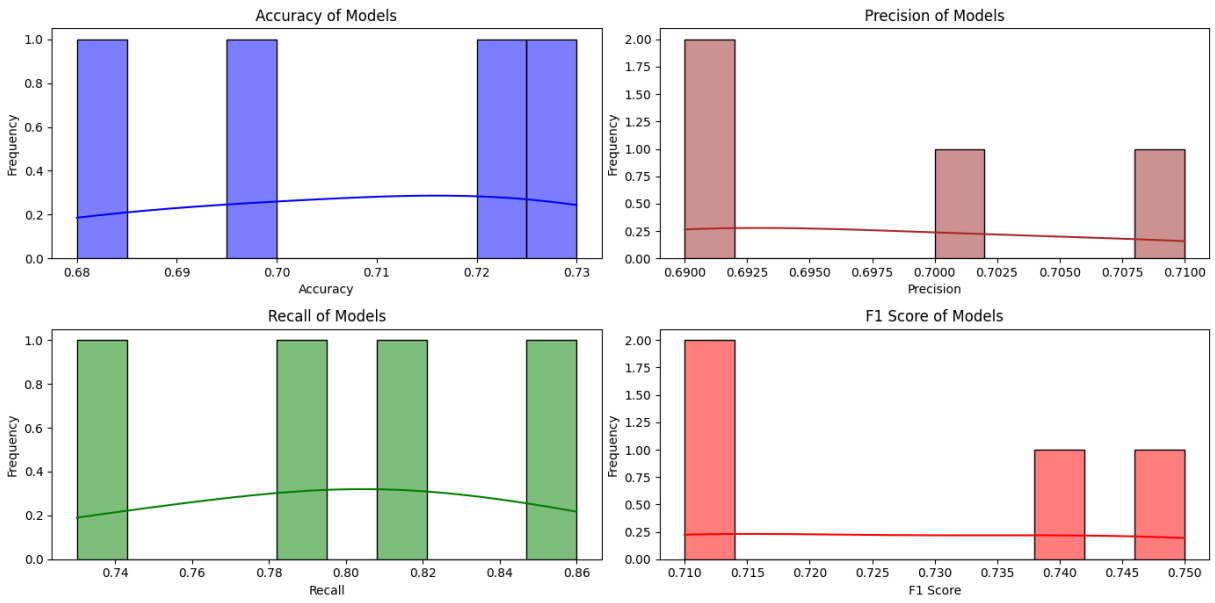


BIOLOGY

In [10]:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns



# Create a DataFrame for easier plotting
import pandas as pd
# Biology data:
models = ['Logistic Regression','SVM','Random Forest', 'Naive Bayes']
precision = [0.69, 0.71, 0.70, 0.69]  # Replaced with actual precision values
recall = [0.79, 0.81, 0.73, 0.86]    # Replaced with actual recall values
f1_score = [0.74, 0.75, 0.71, 0.71]  # Replaced with actual F1 score values

# Add accuracy data - you need to get these values from your model evaluation
accuracy = [0.70, 0.72, 0.68, 0.73]

# Create a DataFrame for easier plotting
import pandas as pd

data = {
    'Model': models,
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1_score
}
df = pd.DataFrame(data)

# Set the figure size
plt.figure(figsize=(14, 7))

# Create histplots for each metric
```

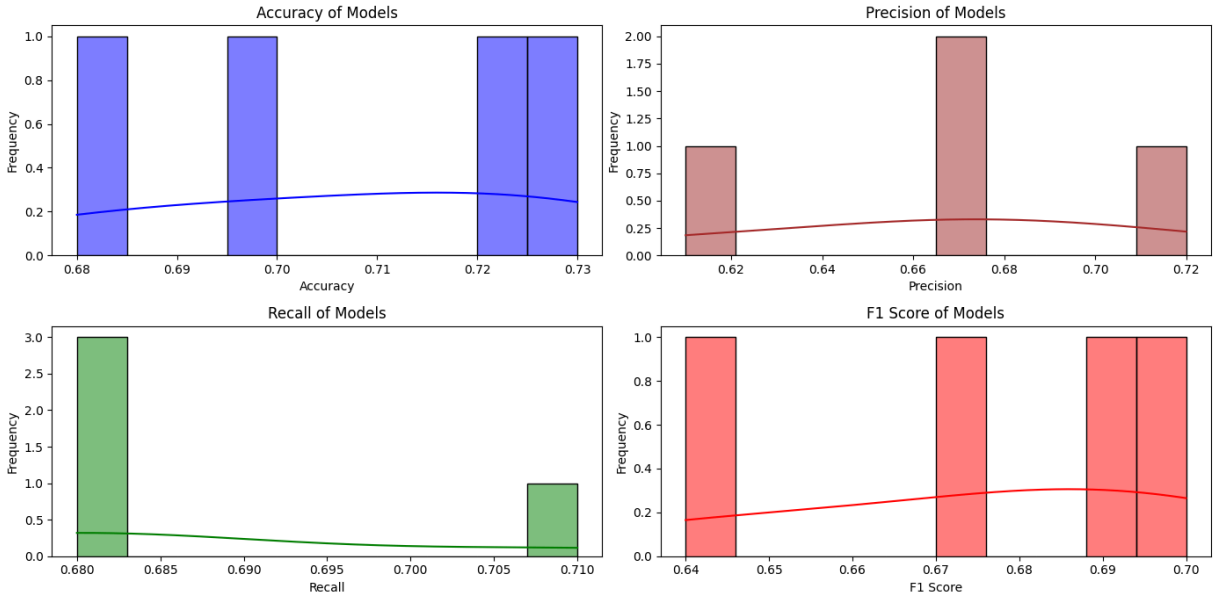
```
plt.subplot(2, 2, 1)
sns.histplot(df['Accuracy'], bins=10, color='blue', kde=True)
plt.title('Accuracy of Models')
plt.xlabel('Accuracy')
plt.ylabel('Frequency')

plt.subplot(2, 2, 2)
sns.histplot(df['Precision'], bins=10, color='brown', kde=True)
plt.title('Precision of Models')
plt.xlabel('Precision')
plt.ylabel('Frequency')

plt.subplot(2, 2, 3)
sns.histplot(df['Recall'], bins=10, color='green', kde=True)
plt.title('Recall of Models')
plt.xlabel('Recall')
plt.ylabel('Frequency')

plt.subplot(2, 2, 4)
sns.histplot(df['F1 Score'], bins=10, color='red', kde=True)
plt.title('F1 Score of Models')
plt.xlabel('F1 Score')
plt.ylabel('Frequency')

# Adjust layout
plt.tight_layout()
plt.show()
```



CHEMISTRY

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns



# Create a DataFrame for easier plotting
```

```
import pandas as pd
# Chemistry data:
models = ['Logistic Regression','SVM','Random Forest', 'Naive Bayes']
precision = [0.67, 0.67, 0.61, 0.72]  # Replaced with actual precision values
recall = [0.68, 0.71, 0.68, 0.68]    # Replaced with actual recall values
f1_score = [0.67, 0.69, 0.64, 0.70]  # Replaced with actual F1 score values

# Add accuracy data - you need to get these values from your model evaluation
accuracy = [0.70, 0.72, 0.68, 0.73]

# Create a DataFrame for easier plotting
import pandas as pd

data = {
    'Model': models,
    'Accuracy': accuracy,
    'Precision': precision,
    'Recall': recall,
    'F1 Score': f1_score
}
df = pd.DataFrame(data)

# Set the figure size
plt.figure(figsize=(14, 7))

# Create histplots for each metric
plt.subplot(2, 2, 1)
sns.histplot(df['Accuracy'], bins=10, color='blue', kde=True)
plt.title('Accuracy of Models')
plt.xlabel('Accuracy')
plt.ylabel('Frequency')

plt.subplot(2, 2, 2)
sns.histplot(df['Precision'], bins=10, color='brown', kde=True)
plt.title('Precision of Models')
plt.xlabel('Precision')
plt.ylabel('Frequency')

plt.subplot(2, 2, 3)
sns.histplot(df['Recall'], bins=10, color='green', kde=True)
plt.title('Recall of Models')
plt.xlabel('Recall')
plt.ylabel('Frequency')

plt.subplot(2, 2, 4)
sns.histplot(df['F1 Score'], bins=10, color='red', kde=True)
plt.title('F1 Score of Models')
plt.xlabel('F1 Score')
plt.ylabel('Frequency')

# Adjust layout
plt.tight_layout()
plt.show()
```



Preprocessing Data

In [36]:

```
from sklearn.feature_extraction.text import CountVectorizer
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
import string
import re
from sklearn.metrics import accuracy_score, classification_report # Import accuracy
from sklearn.naive_bayes import MultinomialNB

# Initialize NLTK resources
# Uncomment the following lines if running for the first time
#nltk.download('punkt')
#nltk.download('stopwords')
#nltk.download('wordnet')
# Define a function for text preprocessing
def preprocess(text):
    if pd.isnull(text):
        return ""
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Tokenize
    tokens = word_tokenize(text)
    # Remove stop words
    tokens = [word for word in tokens if word not in stopwords.words('english')]
    # Lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)
# Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
```

```
# Apply preprocessing
train_data['processed_Comment'] = train_data['Comment'].apply(preprocess)
test_data['processed_Comment'] = test_data['Comment'].apply(preprocess)



# Check processed comments
print(train_data[['Comment', 'processed_Comment']].head())
```

```
                                             Comment  \
0  A few things. You might have negative- frequen...
1  Is it so hard to believe that there exist part...
2                                  There are bees
3  I'm a medication technician. And that's alot o...
4                    Cesium is such a pretty metal.

                                  processed_Comment
0  thing might negative frequency dependent selec...
1  hard believe exist particular cant detect anyt...
2                                              bee
3  im medication technician thats alot drug liver...
4                              cesium pretty metal
```

Text Representation with Preprocessing

In [37]:

```
# Prepare pre-processed data
X_train_processed = train_data['processed_Comment']
X_test_processed = test_data['processed_Comment']
y_train = train_data['Topic']
y_test = test_data['Topic']

# Bag of Words with pre-processed data
vectorizer = CountVectorizer()
X_train_bow = vectorizer.fit_transform(X_train_processed)
X_test_bow = vectorizer.transform(X_test_processed)

# Train Naive Bayes model
model.fit(X_train_bow, y_train)

# Make predictions
y_pred_processed = model.predict(X_test_bow)

# Check lengths
print(f"Length of y_test: {len(y_test)}")
print(f"Length of y_pred_processed: {len(y_pred_processed)}")

# Evaluate
print("Accuracy with pre-processing:", accuracy_score(y_test, y_pred_processed))
print("Classification Report with pre-processing:")
print(classification_report(y_test, y_pred_processed))
```

```
Length of y_test: 1586
Length of y_pred_processed: 1586
Accuracy with pre-processing: 0.8278688524590164
Classification Report with pre-processing:
              precision    recall  f1-score   support

     Biology       0.87      0.82      0.85       614
   Chemistry       0.77      0.84      0.81       506
     Physics       0.84      0.83      0.83       466

    accuracy                           0.83      1586
   macro avg       0.83      0.83      0.83      1586
weighted avg       0.83      0.83      0.83      1586
```

Using N-Grams

In [39]:
```python
# Using bigrams
vectorizer_ngram = CountVectorizer(ngram_range=(1, 2))
X_train_ngram = vectorizer_ngram.fit_transform(X_train_processed)
X_test_ngram = vectorizer_ngram.transform(X_test_processed)

# Initialize the Naive Bayes model
model = MultinomialNB()
# Train Naive Bayes model
model.fit(X_train_ngram, y_train)

# Make predictions
y_pred_ngram = model.predict(X_test_ngram)

# Evaluate
print("Accuracy with bigrams:", accuracy_score(y_test, y_pred_ngram))
print("Classification Report with bigrams:")
print(classification_report(y_test, y_pred_ngram))
```

```
Accuracy with bigrams: 0.8423707440100883
Classification Report with bigrams:
              precision    recall  f1-score   support

     Biology       0.85      0.88      0.86       614
   Chemistry       0.80      0.83      0.82       506
     Physics       0.89      0.80      0.84       466

    accuracy                           0.84      1586
   macro avg       0.85      0.84      0.84      1586
weighted avg       0.84      0.84      0.84      1586
```

TF-IDF

In [40]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

# TF-IDF Vectorization
vectorizer_tfidf = TfidfVectorizer()
X_train_tfidf = vectorizer_tfidf.fit_transform(X_train_processed)
X_test_tfidf = vectorizer_tfidf.transform(X_test_processed)
```

```
# Train Naive Bayes model
model.fit(X_train_tfidf, y_train)

# Make predictions
y_pred_tfidf = model.predict(X_test_tfidf)

# Evaluate
print("Accuracy with TF-IDF:", accuracy_score(y_test, y_pred_tfidf))
print("Classification Report with TF-IDF:")
print(classification_report(y_test, y_pred_tfidf))
```

```
Accuracy with TF-IDF: 0.8013871374527112
Classification Report with TF-IDF:
              precision    recall  f1-score   support

     Biology       0.76      0.89      0.82       614
   Chemistry       0.78      0.80      0.79       506
     Physics       0.92      0.69      0.79       466

    accuracy                           0.80      1586
   macro avg       0.82      0.79      0.80      1586
weighted avg       0.81      0.80      0.80      1586
```

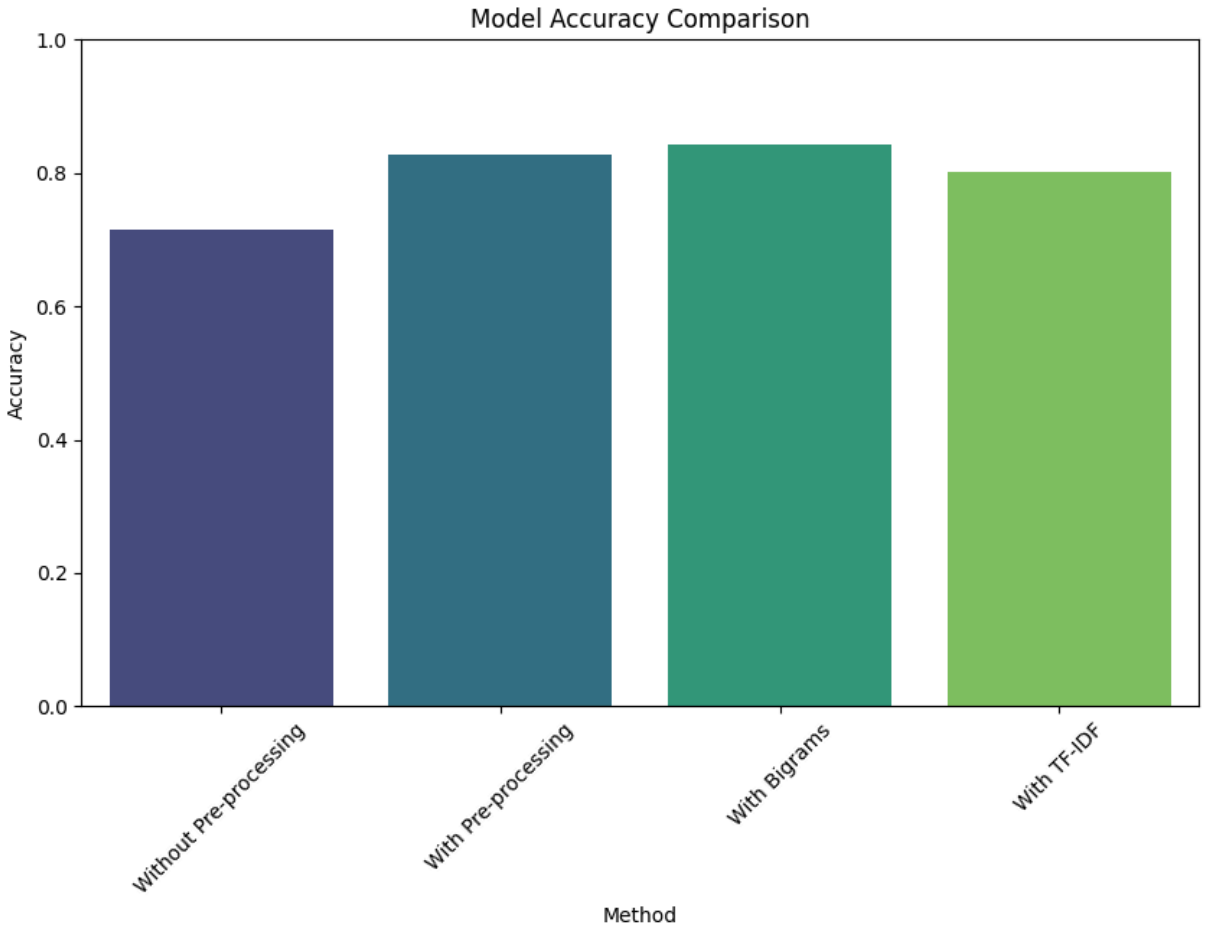
Results Comparison

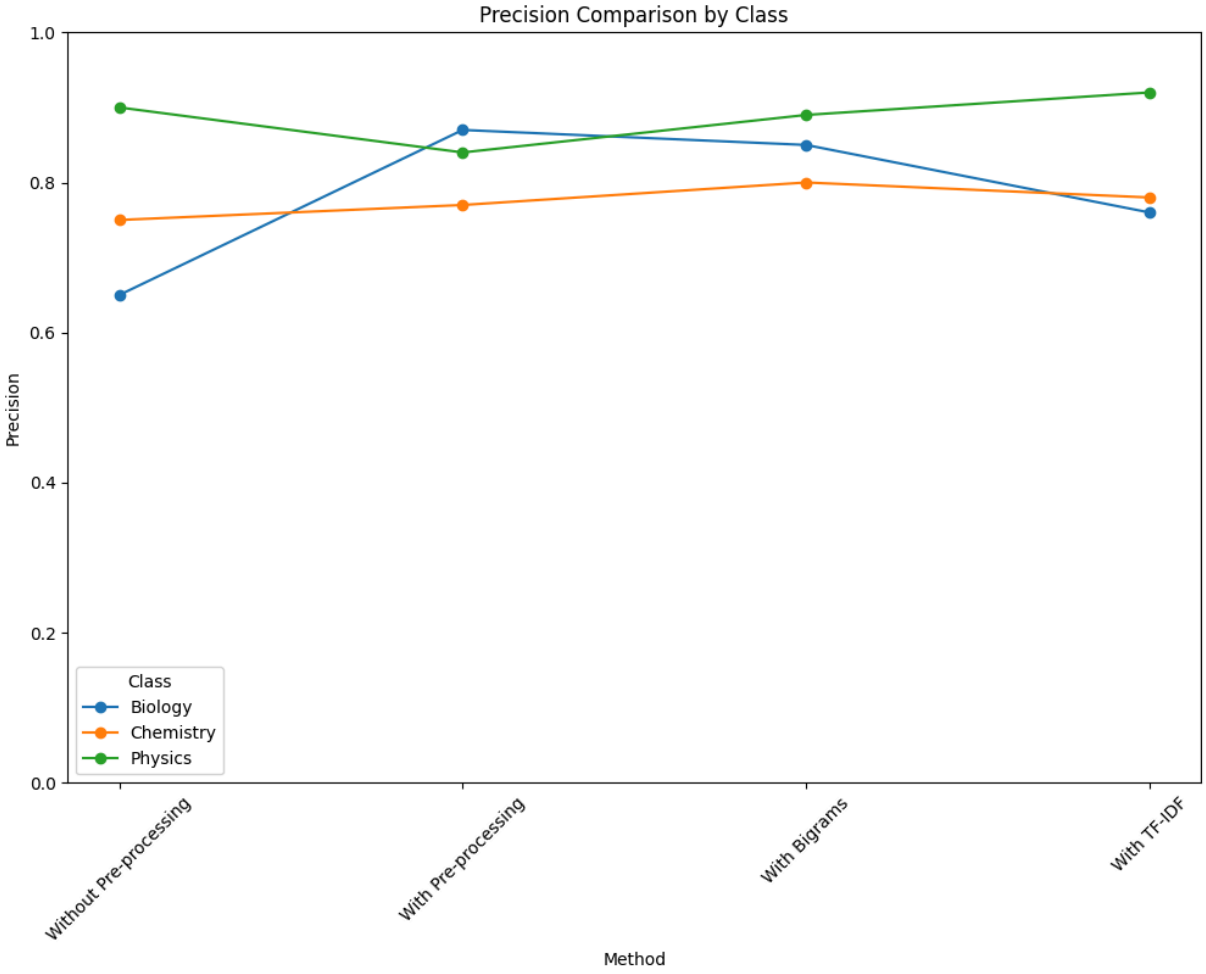
In [41]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Create DataFrames for each method's results
data = {
    'Method': [
        'Without Pre-processing', 'With Pre-processing', 'With Bigrams', 'With TF-I
    ],
    'Accuracy': [
        0.7148114075436982,
        0.8278688524590164,
        0.8423707440100883,
        0.8013871374527112
    ],
    'Biology Precision': [0.65, 0.87, 0.85, 0.76],
    'Biology Recall': [0.89, 0.82, 0.88, 0.89],
    'Biology F1-Score': [0.75, 0.85, 0.86, 0.82],
    'Chemistry Precision': [0.75, 0.77, 0.80, 0.78],
    'Chemistry Recall': [0.64, 0.84, 0.83, 0.80],
    'Chemistry F1-Score': [0.69, 0.81, 0.82, 0.79],
    'Physics Precision': [0.90, 0.84, 0.89, 0.92],
    'Physics Recall': [0.51, 0.83, 0.80, 0.69],
    'Physics F1-Score': [0.65, 0.83, 0.84, 0.79]
}

df_results = pd.DataFrame(data)

# Print the DataFrame for inspection
```

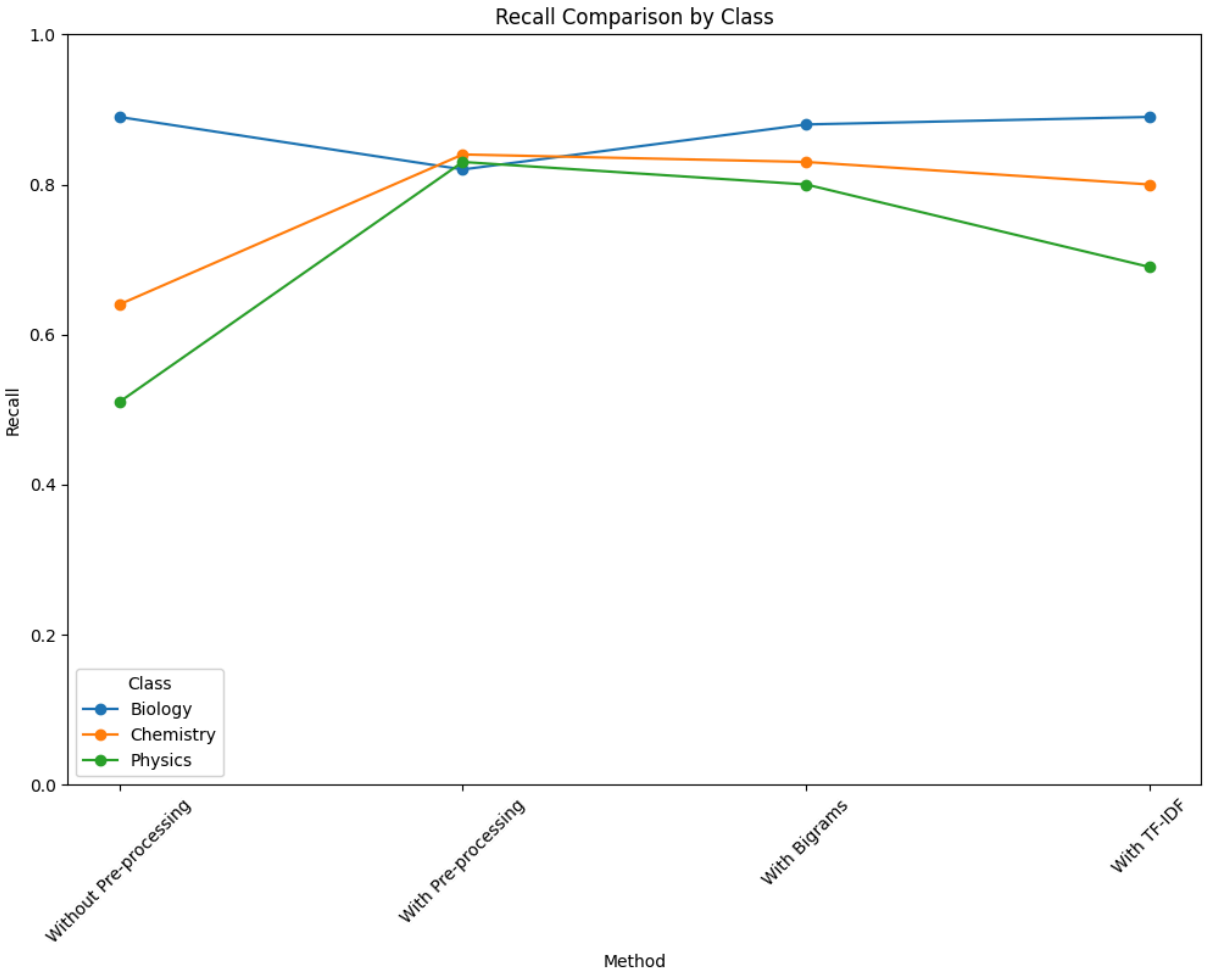
```
print("Comparison DataFrame:")
print(df_results)

# Plot Accuracy Comparison
plt.figure(figsize=(10, 6))
sns.barplot(x='Method', y='Accuracy', data=df_results, palette='viridis')
plt.title('Model Accuracy Comparison')
plt.xlabel('Method')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
plt.xticks(rotation=45)
plt.show()

# Plot Precision, Recall, and F1-Score for each class
metrics = ['Precision', 'Recall', 'F1-Score']
classes = ['Biology', 'Chemistry', 'Physics']

for metric in metrics:
    plt.figure(figsize=(12, 8))
    for cls in classes:
        plt.plot(df_results['Method'], df_results[f'{cls} {metric}'], marker='o', l
    plt.title(f'{metric} Comparison by Class')
    plt.xlabel('Method')
    plt.ylabel(f'{metric}')
    plt.ylim(0, 1)
    plt.legend(title='Class')
    plt.xticks(rotation=45)
    plt.show()

# Plot Macro Avg and Weighted Avg Metrics
avg_data = {
    'Method': ['Without Pre-processing', 'With Pre-processing', 'With Bigrams', 'Wi
    'Macro Avg Precision': [0.77, 0.83, 0.85, 0.82],
    'Macro Avg Recall': [0.68, 0.83, 0.84, 0.79],
    'Macro Avg F1-Score': [0.70, 0.83, 0.84, 0.80],
    'Weighted Avg Precision': [0.75, 0.83, 0.84, 0.81],
    'Weighted Avg Recall': [0.71, 0.83, 0.84, 0.80],
    'Weighted Avg F1-Score': [0.71, 0.83, 0.84, 0.80]
}

df_avg_results = pd.DataFrame(avg_data)

for avg_type in ['Macro Avg', 'Weighted Avg']:
    plt.figure(figsize=(12, 8))
    for metric in ['Precision', 'Recall', 'F1-Score']:
        plt.plot(df_avg_results['Method'], df_avg_results[f'{avg_type} {metric}'],
    plt.title(f'{avg_type} Metrics Comparison')
    plt.xlabel('Method')
    plt.ylabel('Score')
    plt.ylim(0, 1)
    plt.legend(title='Metric')
    plt.xticks(rotation=45)
    plt.show()
```

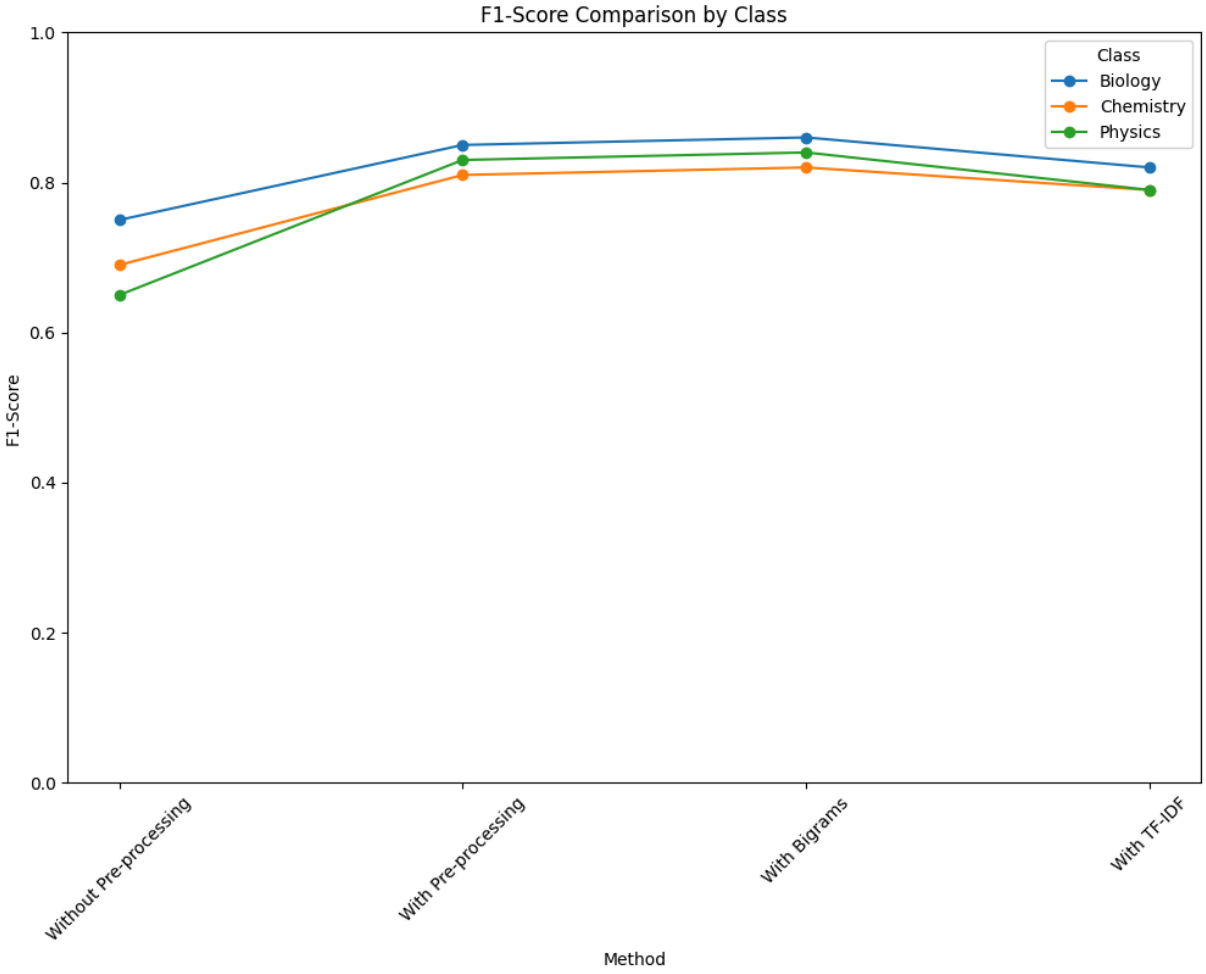
```
Comparison DataFrame:
                   Method  Accuracy  Biology Precision  Biology Recall  \
0  Without Pre-processing  0.714811               0.65            0.89
1     With Pre-processing  0.827869               0.87            0.82
2            With Bigrams  0.842371               0.85            0.88
3             With TF-IDF  0.801387               0.76            0.89

   Biology F1-Score  Chemistry Precision  Chemistry Recall  \
0              0.75                 0.75              0.64
1              0.85                 0.77              0.84
2              0.86                 0.80              0.83
3              0.82                 0.78              0.80

   Chemistry F1-Score  Physics Precision  Physics Recall  Physics F1-Score
0                0.69               0.90            0.51              0.65
1                0.81               0.84            0.83              0.83
2                0.82               0.89            0.80              0.84
3                0.79               0.92            0.69              0.79
```

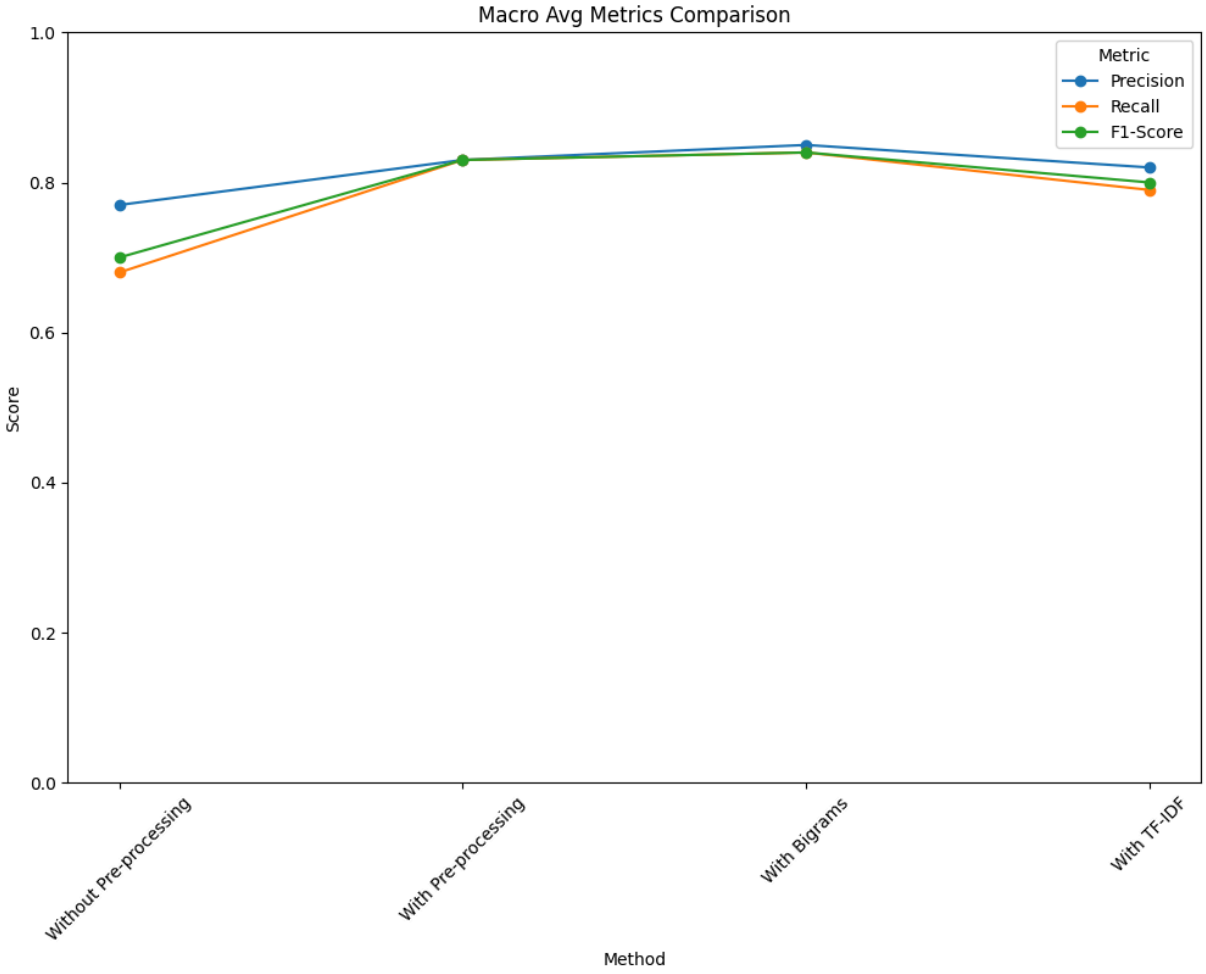
```
<ipython-input-41-3adc7a913c4d>:35: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Method', y='Accuracy', data=df_results, palette='viridis')
```

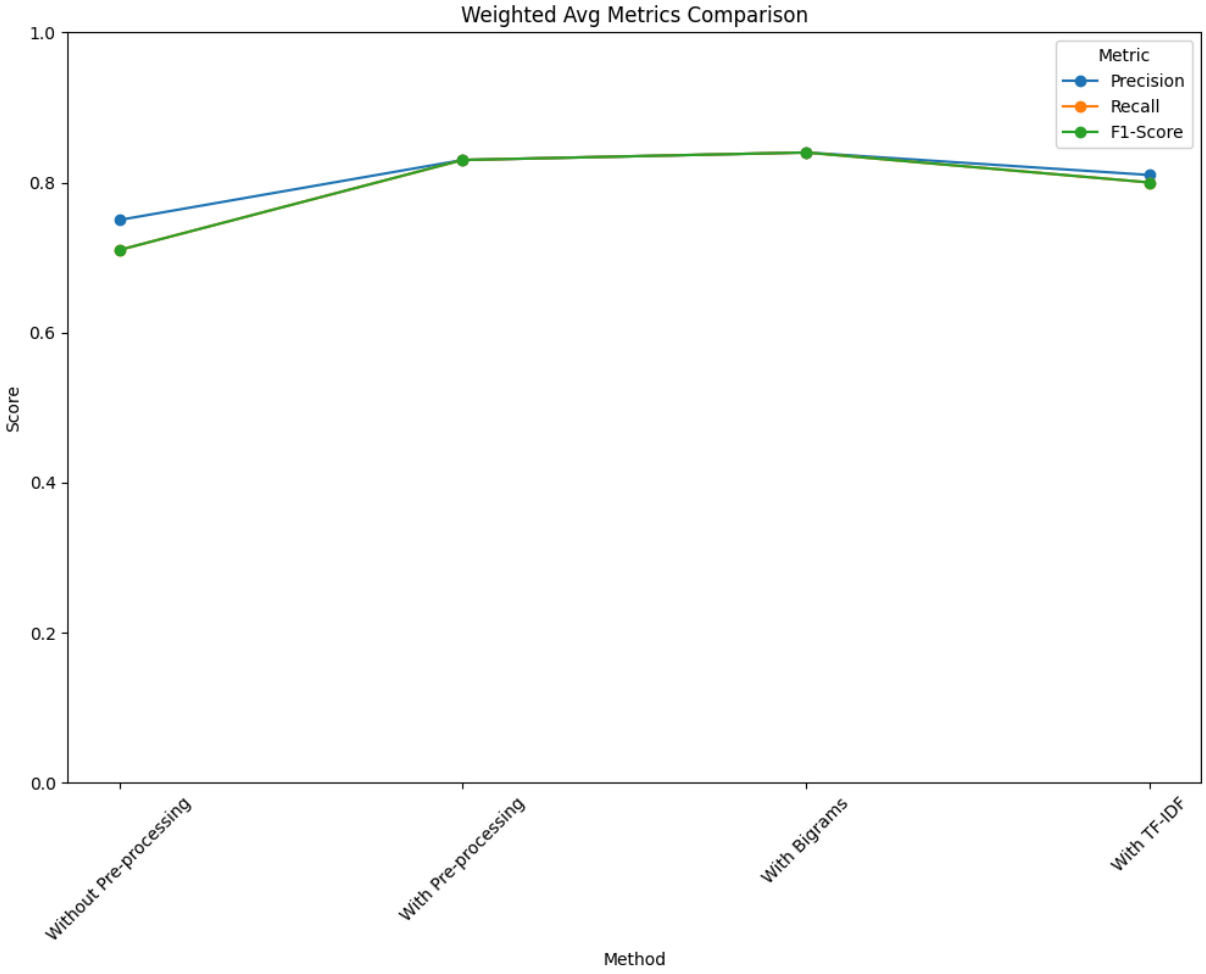



Precision Comparison by Class
1.0
0.8
0.6
0.4
0.2
0.0
Precision
Class
Biology
Chemistry
Physics
Without Pre-processing
With Pre-processing
With Bigrams
With TF-IDF
Method


Recall Comparison by Class
1.0
0.8
0.6
0.4
0.2
0.0
Recall
Class
Biology
Chemistry
Physics
Without Pre-processing
With Pre-processing
With Bigrams
With TF-IDF
Method


F1-Score Comparison by Class
Class
Biology
Chemistry
Physics
1.0
0.8
0.6
0.4
0.2
0.0
F1-Score
Without Pre-processing
With Pre-processing
With Bigrams
With TF-IDF
Method


Macro Avg Metrics Comparison
Metric
Precision
Recall
F1-Score
Score
1.0
0.8
0.6
0.4
0.2
0.0
Without Pre-processing
With Pre-processing
With Bigrams
With TF-IDF
Method



Commentary

The provided code snippet focuses on text preprocessing and model training using a Naive Bayes classifier. Here's a breakdown of the key steps:

Text Preprocessing:

The preprocess function performs essential text cleaning tasks: -Removing URLs to eliminate noise and irrelevant information. -Converting text to lowercase for consistency. -Removing punctuation to focus on word content. -Tokenizing the text into individual words for analysis. -Removing stop words (common words like "the", "and", etc.) to retain meaningful terms. -Lemmatizing words to reduce them to their root form, improving analysis. 2. **TF-IDF Vectorization with Bigrams**:

-The code uses TfidfVectorizer with ngram_range=(1, 2) to convert preprocessed text into numerical features. -This means it considers both individual words (unigrams) and pairs of consecutive words (bigrams) for a richer representation of the text.

3. **Model Training**:

-A Multinomial Naive Bayes model (MultinomialNB) is initialized. -The model is trained on the vectorized training data (X_train_ngram) and corresponding labels (y_train).

4. **Evaluation (Not Shown)**:

The code includes imports for accuracy_score and classification_report, suggesting that the next step would involve evaluating the model's performance on test data.

Observations

The code demonstrates good practice in text preprocessing, which is crucial for effective text analysis. Using bigrams can capture more context and potentially improve model performance. The choice of Multinomial Naive Bayes is suitable for text classification tasks.

Conclusion

The provided code snippet lays the groundwork for text classification using a Naive Bayes model. The next logical steps would be:

1. Model Evaluation: Use the imported metrics to assess the model's performance on the test data (X_test_ngram and y_test).
2. Hyperparameter Tuning: Explore different values for the alpha parameter of the MultinomialNB model to potentially improve results.
3. Further Analysis: Depending on the evaluation results, consider additional feature engineering, model selection, or error analysis to refine the solution.