OBJECTIVE

This task aims to utilize an open-source LLM to analyze and derive insights from a dataset that combines text data (such as product reviews) with other forms of metadata (like images or numerical data). You will extract meaningful features, analyze relationships, and build a predictive model. This task will evaluate your skills in multi-modal analysis, feature engineering, and predictive analytics.

Task:

1. Analyze the sentiment of each review and generate sentiment scores.
2. Extract key phrases, themes, or topics relevant to the reviews that could correlate with

product ratings.

1. Integrate textual data with numerical features from the metadata (e.g., ratings, number

of helpful votes).

1. Create a comprehensive feature set by combining:
   * Textual features (e.g., sentiment scores, key phrases).
   * Numerical features from metadata (e.g., product ratings).
   * If image data is available, employ a pre-trained model (e.g., a Convolutional Neural Network) to extract features from the images and combine them with text-based features.
   * Analyze the interaction between sentiment scores and other features to derive additional insights.

Goals: Develop a predictive model to forecast product ratings or sales using the engineered features.

Data Collection

### **Task: Data Collection**

**Objective:**  
The objective of this task is to collect a comprehensive dataset of product reviews from the Amazon Fine Food Reviews Dataset available on Kaggle. This dataset contains product reviews along with relevant metadata, including ratings, review text, and associated images. The dataset will be used for analysis and further processing in the project.

**Procedure:**

* **Install Dependencies:**  
  To access and download the dataset, the kagglehub package is used. This package allows seamless integration with Kaggle to download datasets directly.

!pip install kagglehub

* **Download Dataset:  
  The dataset is downloaded using the kagglehub package. The following Python code defines the destination path where the dataset will be saved and uses the dataset\_download() function to download the Amazon Fine Food Reviews dataset.**

**import kagglehub**

**# Define the destination path where you want to save the dataset**

**destination\_path = r"C:\Users\Novel kathor\Desktop\Task\_2\archive"**

**# Download the latest version of the dataset to the specified path**

**path = kagglehub.dataset\_download("snap/amazon-fine-food-reviews", destination\_path=destination\_path)**

**print("Path to dataset files:", path)**

The dataset\_download() function retrieves the dataset and saves it at the specified location on your local machine

* **Dataset Overview:**  
  The downloaded dataset contains the following columns:

|  |  |
| --- | --- |
| Column Name | Description |
| Id | |  | | --- | | Unique identifier for the review. |  |  | | --- | |  | |
| ProductId | Identifier for the product being reviewed. |
| UserId | |  | | --- | | Identifier for the user who wrote the review. |  |  | | --- | |  | |
| ProfileName | Profile name of the user who wrote the review. |
| HelpfulnessNumerator | Number of users who found the review helpful. |
| HelpfulnessDenominator | |  | | --- | | Total number of users who rated the review's helpfulness. |  |  | | --- | |  | |
| Score | Rating score given by the user (1-5). |
| Time | |  | | --- | |  |  |  | | --- | | Timestamp when the review was written. | |
| Summary | |  | | --- | | A brief summary of the review. |  |  | | --- | |  | |
| Text | **Full review text written by the user.** |

**Outcome:**  
Upon completion of this task, the dataset will be available at the specified location (C:\Users\Novel kathor\Desktop\Task\_2\archive). The dataset is now ready for further analysis, such as sentiment analysis, feature extraction, or recommendation system development.

Text Preprocessing

### **Task: Text Preprocessing**

**Objective:**  
The objective of this task is to clean and preprocess the review text from the Amazon Fine Food Reviews dataset. The steps include removing special characters, URLs, excessive whitespace, normalizing text, tokenizing, removing stop words, and applying lemmatization to ensure uniformity in the text data.

**Procedure:**

**Installing Dependencies:**  
The necessary Python libraries for text preprocessing, including re, pandas, spaCy, and nltk, are imported. Specific resources like stopwords and lemmatizers are also loaded.

!pip install spacy nltk tqdm

The required NLTK data for tokenization, stopwords, and lemmatization:

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

**Text Normalization:**  
A function normalize\_text() is defined to perform the following operations:

* **Remove URLs** from the text using regex.
* **Remove special characters** and **extra whitespace**.
* **Convert the text to lowercase** to ensure uniformity.
  + Common words that do not contribute significant meaning (e.g., "the," "and") were removed to focus on informative terms.

def normalize\_text(text):

text = re.sub(r'http\S+|www\S+|https\S+', '', str(text), flags=re.MULTILINE)

text = re.sub(r'[^a-zA-Z0-9\s]', '', text).strip()

return text.lower()

### **Preprocessing the Summary Column:** The preprocess\_summary() function handles text in the Summary column using spaCy for tokenization, lemmatization, and stopword removal.

**def preprocess\_summary(text):**

**text = normalize\_text(text)**

**text = handle\_negation\_and\_clean(text)**

**doc = nlp(text)**

**processed\_tokens = [token.lemma\_ for token in doc if token.pos\_ not in ['PUNCT', 'DET']]**

**return ' '.join(processed\_tokens)**

**Preprocessing the Text Column:**  
The preprocess\_text() function tokenizes the review text, removes stop words, and applies lemmatization using the NLTK WordNetLemmatizer.

def preprocess\_text(text):

tokens = word\_tokenize(text)

stop\_words = set(stopwords.words('english'))

tokens = [word for word in tokens if word not in stop\_words]

tokens = [lemmatizer.lemmatize(word) for word in tokens]

return ' '.join(tokens)

**Applying Preprocessing:**  
The preprocessing functions are applied to both the Summary and Text columns of the dataset. The dataset is then saved to a new file after processing.

df = pd.read\_csv(input\_file)

df[df.columns] = df[df.columns].progress\_apply(lambda x: x.apply(normalize\_text))

if 'Summary' in df.columns:

df['Cleaned\_Summary'] = df['Summary'].progress\_apply(preprocess\_summary)

if 'Text' in df.columns:

df['Cleaned\_Text'] = df['Text'].progress\_apply(preprocess\_text)

df.to\_csv(output\_file, index=False)

**Outcome:**  
After preprocessing, the dataset will have two new columns: Cleaned\_Summary and Cleaned\_Text, which contain the cleaned and normalized text for further analysis or model training. The processed data is saved to filtered\_data.csv.

Sentiment and Feature Extraction

### **Objective:** This task aims to analyze the sentiment of reviews and extract key phrases or themes that may correlate with product ratings. The process involves utilizing open-source language models (TextBlob and VADER) to determine sentiment and identify relevant key phrases in review text. Additionally, it integrates numerical metadata features like ratings and helpful votes to enrich the analysis.

#### ****Sentiment Analysis:****

Sentiment analysis is performed on the reviews using two methods: **TextBlob** and **VADER** (Valence Aware Dictionary and sEntiment Reasoner).

1. **TextBlob Sentiment Analysis:**  
   TextBlob is used to calculate the **polarity score** of each review, which ranges from -1 (negative sentiment) to 1 (positive sentiment). A score closer to 0 indicates neutral sentiment.

### def analyze\_sentiment\_textblob(text):

### try:

### blob = TextBlob(text)

### return blob.sentiment.polarity

### except Exception as e:

### print(f"Error with TextBlob: {e}")

### return 0

**VADER Sentiment Analysis:**  
VADER is used to analyze the sentiment with a **compound score**, which ranges from -1 (negative) to 1 (positive). This score accounts for intensity and valence of the sentiment expressed.

def analyze\_sentiment\_vader(text):

try:

scores = vader.polarity\_scores(text)

return scores['compound']

except Exception as e:

print(f"Error with VADER: {e}")

return 0

The sentiment analysis is applied to each review, and the results are saved in two new columns: **Sentiment\_TextBlob** and **Sentiment\_VADER**.

# Process each row with progress bar

for text in tqdm(df['Cleaned\_Text'], desc="Processing", unit="texts"):

textblob\_scores.append(analyze\_sentiment\_textblob(text))

vader\_scores.append(analyze\_sentiment\_vader(text))

df['Sentiment\_TextBlob'] = textblob\_scores

df['Sentiment\_VADER'] = vader\_scores

**Result :**

|  |  |
| --- | --- |
| Sentiment\_TextBlob | Sentiment\_VADER |
| 0.425 | 0.9413 |
| 0.216666667 | -0.1027 |
| 0.187 | 0.8624 |

#### ****Extract key phrases, themes, or topics relevant to the reviews that could correlate with product ratings.****

Key phrases or noun chunks are extracted from the reviews using **spaCy**. The spaCy model en\_core\_web\_sm is used to process the text, and noun chunks (important phrases) are extracted from each review.

def extract\_phrases(text):

doc = nlp(text)

return [chunk.text for chunk in doc.noun\_chunks]

These key phrases are then added to the dataset in a new column called **Key\_Phrases**.

df['Key\_Phrases'] = df['Cleaned\_Text'].progress\_apply(extract\_phrases)

#### ****Integrating Metadata:****

The numerical metadata, such as product **ratings** and **number of helpful votes**, can be directly integrated with the textual features (sentiment scores and key phrases) for further analysis. This integration can help uncover correlations between sentiment, key phrases, and product ratings.

For example, product ratings can be merged with sentiment data to explore whether positive or negative sentiment correlates with higher or lower ratings.

#### ****Final Output:****

The final dataset includes:

* **Sentiment scores** from both TextBlob and VADER.
* **Key phrases** extracted from each review.
* **Metadata** such as ratings and helpful votes (if present in the dataset).

The results are saved in a CSV file for further analysis.

df.to\_csv("dataset/output\_with\_key\_phrases.csv", index=False)

**Result:**

|  |
| --- |
| Key\_Phrases |
| ['several vitality canned dog food product', 'good quality product', 'stew processed meat smell', 'better labrador finicky', 'product'] |
| ['product', 'labeled jumbo', 'peanutsthe peanut', 'small sized unsalted sure error vendor'] |
| ['century light', 'gelatin nut case', 'filbert', 'tiny square', 'liberally coated powdered sugar tiny mouthful heaven chewy', 'familiar story', 'c lewis lion witch wardrobe', 'seduces', 'brother sister witch'] |

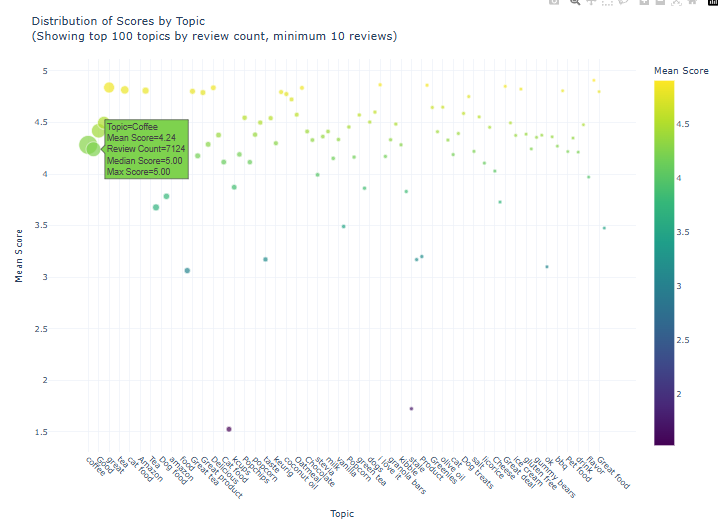
**Extracting themes, topics :**

The task is to extract key phrases, themes, or topics relevant to reviews, aiming to correlate them with product ratings. This approach helps in understanding customer feedback better and improving product offerings. The implementation uses a T5 transformer model to generate single-word topics and themes for each review text. It leverages PyTorch, Hugging Face Transformers, and efficient batching to process large datasets in minimal time while monitoring memory usage.

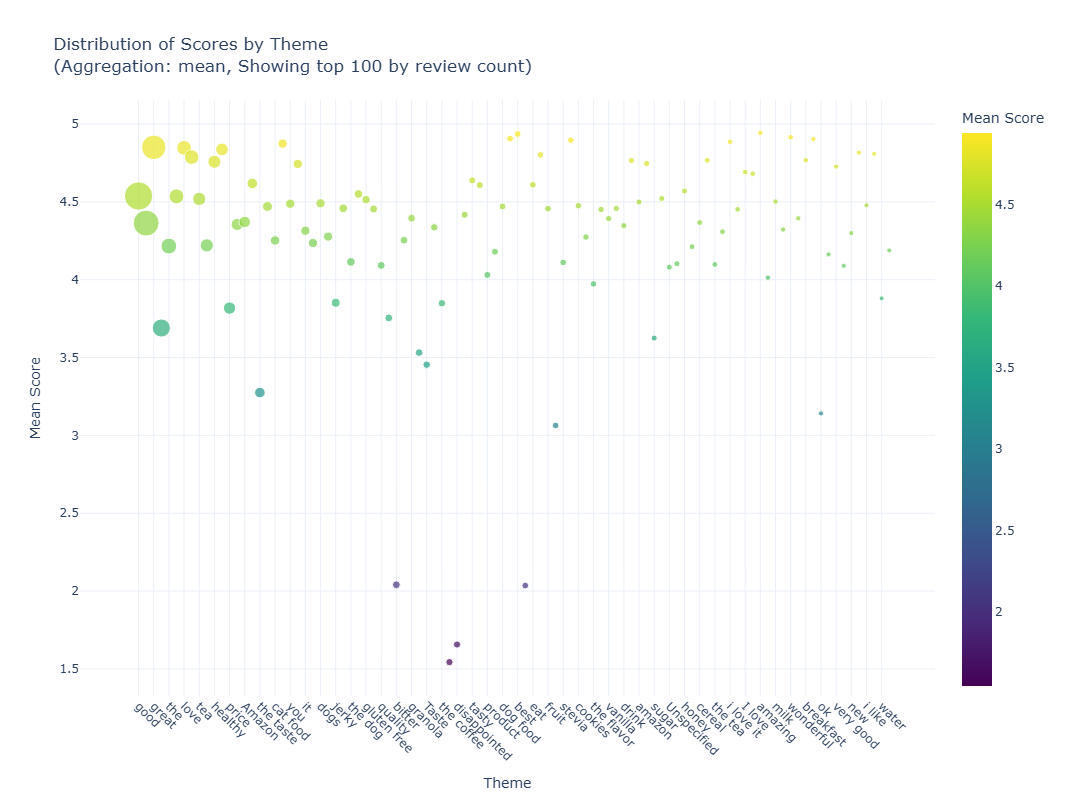
#### ****Key Features****

1. **Model**: The google/flan-t5-base model is fine-tuned to extract meaningful topics and themes.
2. **Dataset Handling**: A custom Dataset class prepares prompts and input for tokenization.
3. **Batch Processing**: Uses PyTorch DataLoader for efficient batch processing.
4. **Memory Monitoring**: GPU and RAM usage are monitored for optimizing resources.
5. **Incremental Saving**: Periodically saves results to avoid loss of progress during interruptions.

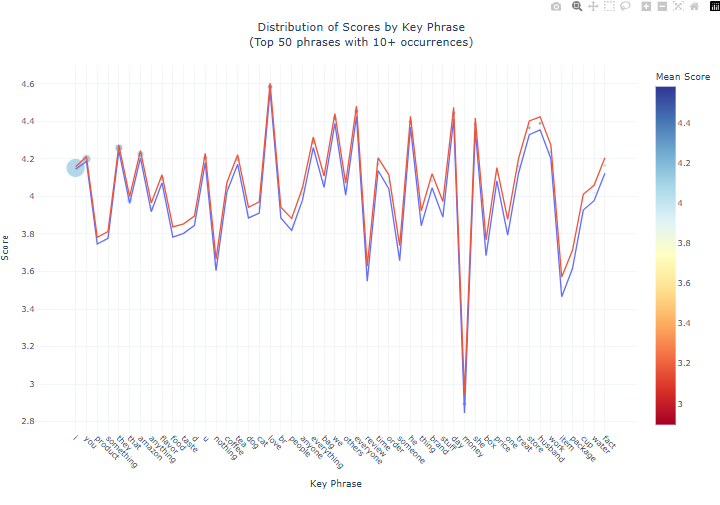
The scatter plot visualizes the distribution of scores by topic, highlighting the most reviewed topics with detailed metrics for easy interpretation.



**Theme Visualization:** A scatter plot showing aggregated scores for themes, with bubble size and color indicating review count and score, respectively.



### This script performs key phrase analysis on a dataset and visualizes the results using an interactive scatter plot. The focus is on statistical summaries of scores associated with key phrases, including mean, median, confidence intervals, and occurrence counts.



### **Integrate Textual Data With Numerical Features From The Metadata (E.G., Ratings, Number**

### **Of Helpful Votes).**

To enhance the feature set of our dataset, we integrate textual data with numerical features from the metadata, such as ratings and helpfulness scores. The process involves reading and processing data in chunks, applying text vectorization, and scaling the features. Below is a step-by-step breakdown of the process:

#### 1. ****Data Reading and Chunking****

We begin by reading the dataset in chunks to efficiently handle large datasets without memory issues. This is achieved using the pandas.read\_csv() function, specifying the columns that include both numerical and textual features. The data is processed in chunks of 10,000 rows to ensure scalability.

#### 2. ****Numerical Feature Processing****

The dataset includes numerical features such as:

* **Score**: A rating score for the product.
* **HelpfulnessNumerator** and **HelpfulnessDenominator**: Used to calculate the helpfulness ratio of a review.
* **Sentiment scores** from **TextBlob** and **VADER**: Represent the sentiment expressed in the review text.

n addition to these features, we calculate a new derived feature, the **Helpfulness Ratio**, by dividing the **HelpfulnessNumerator** by the **HelpfulnessDenominator** (with 0 denominators replaced by 1 to avoid division by zero).

#### 3. ****Textual Feature Processing****

The textual data, consisting of cleaned text reviews, is processed using a TfidfVectorizer. The TfidfVectorizer converts the text into numerical features, considering the term frequency and inverse document frequency (TF-IDF) of words in the text. Key parameters include:

* **Max Features**: Limit the number of features to the top 100 based on TF-IDF.
* **Stop Words**: Exclude common English words that do not contribute much to the meaning.
* **Min/Max Document Frequency**: Ensure terms appear in at least two documents and at most 95% of the documents.

The resulting text features are stored in a sparse matrix format to conserve memory.

#### 4. ****Feature Scaling****

After extracting text and numerical features, we scale the numerical features using **StandardScaler**, which normalizes them to have zero mean and unit variance. This is essential for machine learning algorithms that are sensitive to feature scales.

Additionally, we scale the text features. Since the text features are sparse (containing many zero values), we apply scaling to only the non-zero values in each feature column.

#### 5. ****Combining the Features****

Finally, we combine the scaled numerical features and text features into a single dataframe. This creates a final feature matrix that can be used for machine learning models.

#### 6. ****Data Storage****

The processed features are stored in a CSV file (processed\_features\_final\_v4.csv), which includes both the scaled numerical and textual features.

### Dataset Summary

* **Total Number of Rows in the Final Dataset**: 568,454
* **Total Number of Features**: 106
* **Expected Number of Features**: 106

### Distribution of Numerical Features:

The distribution of numerical features is summarized below:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Feature | Count | Mean | Std | Min | 25% | 50% | 75% | |  | | --- | |  |  |  | | --- | | Max | |
| Score | 568,454 | 0.000 | 1.000 | -2.429116 | -0.139799 | -0.623306 | 0.623306 | 0.623306 |
| |  | | --- | | HelpfulnessNumerator |  |  | | --- | |  | | 568,454 | 0.000 | 1.000 | -0.228353 | -0.228353 | -0.228353 | 0.033547 | 113.174300 |
| HelpfulnessDenominator | 568,454 | 0.000 | 1.000 | -0.268864 | -0.268864 | -0.148233 | |  | | --- | | -0.027602 |  |  | | --- | |  | | 111.073700 |
| Sentiment\_TextBlob | 568,454 | -0.000 | 1.000 | -5.456631 | |  | | --- | | -0.610190 |  |  | | --- | |  | | -0.030739 | 0.596447 | 3.294807 |
| Sentiment\_VADER | 568,454 | 0.000 | 1.000 | -4.114899 | -0.125829 | -0.422169 | |  | | --- | | 0.618405 |  |  | | --- | |  | | 0.758817 |
| Helpfulness\_Ratio | 568,454 | 0.000 | 1.000 | -0.882688 | -0.882688 | |  | | --- | | -0.882688 |  |  | | --- | |  | | 1.281496 | 5.609863 |

### Verification of Scaling of Numerical Features:

The scaling of the numerical features was verified and is as follows:

* **Score**: Mean = 0.000000, Std = 1.000001
* **HelpfulnessNumerator**: Mean = 0.000000, Std = 1.000001
* **HelpfulnessDenominator**: Mean = 0.000000, Std = 1.000001
* **Sentiment\_TextBlob**: Mean = -0.000000, Std = 1.000001
* **Sentiment\_VADER**: Mean = 0.000000, Std = 1.000001
* **Helpfulness\_Ratio**: Mean = 0.000000, Std = 1.000001

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Feature | Count | Mean | Std | Min | 25% | 50% | 75% | |  | | --- | |  |  |  | | --- | | Max | |
| text\_feature\_0 | 568,454 | 0.000 | 0.238 | 1.996215 | 0.000 | 0.000 | 0.000 | 5.305102 |
| text\_feature\_1 | 568,454 | 0.000 | 0.343 | 1.963583 | 0.000 | 0.000 | 0.000 | 5.512063 |
| text\_feature\_2 | 568,454 | 0.000 | 0.220 | 1.919386 | 0.000 | 0.000 | 0.000 | 5.004655 |
| text\_feature\_3 | 568,454 | 0.000 | 0.309 | 1.915893 | 0.000 | 0.000 | 0.000 | 4.897740 |
| … | … | … | … | … | … | … | … | … |
| text\_feature\_99 | 568,454 | 0.000 | 0.292 | -1.901997 | 0.000 | 0.000 | 0.000 | 5.614159 |

### Sparsity of Text Features:

* **Sparsity**: 0.9149

This indicates that a significant portion of the text features are sparse (i.e., have a value of zero).

### Data Processing Summary:

* The dataset consists of **568,454 rows** with **106 features**, which include both numerical and text-based features.
* All features were **scaled and processed** to standardize their distributions.
* **Sparsity** was calculated for the text features, showing a value of **0.9149**, indicating that most of the text features contain a high proportion of zero values.
* The data was **processed in chunks of 10,000 rows** to optimize memory usage. However, the final dataset contains all **5,68,454 rows**, combining the processed data from each chunk.

Multi-Modal Feature Engineering

### **Objective**

The goal of this task is to create a comprehensive feature set by combining various types of features to enhance the predictive capability of machine learning models. This includes:

* **Textual Features**: Extracting sentiment scores and key phrases.
* **Numerical Features**: Utilizing metadata like product ratings and helpfulness scores.
* **Interaction Analysis**: Exploring the relationships between sentiment scores and other features to derive deeper insights.

### **Methodology**

To achieve the objective, the following steps were implemented:

1. **Loading Datasets**:
   * Dataset 1: Contains user reviews, sentiment scores, and metadata.
   * Dataset 2: Contains processed text features such as TF-IDF vectors.
2. **Feature Engineering**:
   * **Basic Features**: Extracted directly from the dataset, such as sentiment scores and helpfulness metrics.
   * **Helpfulness Ratio**: Calculated to represent the ratio of helpful votes to total votes for a review.
   * **User Statistics**: Aggregated user-level statistics, including average scores, review count, and score standard deviation.
   * **Product Statistics**: Aggregated product-level statistics, including average scores, review count, and score standard deviation.
   * **Text Features**: Added advanced text features from the second dataset, such as TF-IDF vectors for textual analysis.
3. **Data Cleaning**:
   * Missing values were handled by filling them with column-wise means.
4. **Output**:
   * A new feature set was saved as new\_features.csv.
   * The target variable (Score) was saved separately as target.csv.

### **Code Explanation**

#### ****Key Steps in the Code****:

1. **Import Libraries**: Essential libraries like pandas, numpy, sklearn, and tqdm were used for data processing, feature engineering, and efficiency.
2. **Function: engineer\_features(df1, df2)**:
   * **Basic Features**:
     + Extracted key columns: Sentiment\_TextBlob, Sentiment\_VADER, HelpfulnessNumerator, and HelpfulnessDenominator.
   * **Helpfulness Ratio**:

Computed using:

helpfulness\_ratio = HelpfulnessNumerator​/HelpfulnessDenominator+1

* + - 
  + **User-Level Statistics**:
    - Aggregated using groupby on the UserId column to calculate metrics like mean score, review count, and standard deviation.
  + **Product-Level Statistics**:
    - Similarly, aggregated metrics at the ProductId level.
  + **Textual Features**:
    - Extracted additional text-based features (e.g., TF-IDF scores) from df2.

1. **DataFrame Creation**:
   * Combined all features into a single DataFrame and handled missing values by replacing them with column-wise means.
2. **Saving Outputs**:
   * The engineered feature set was saved to a CSV file, ensuring compatibility with downstream machine learning pipelines.

### **Results**

* **Number of Features Generated**:  
  The function outputs a dataset with all the newly engineered features. The total number of features generated depends on the input datasets but includes user- and product-level aggregations, text features, and calculated metrics.
* **Output Files**:
  + **Feature Set**: Saved as new\_features.csv, containing the engineered features.
  + **Target Variable**: Saved as target.csv, containing the Score column for supervised learning.

Predictive Modeling

The objective is to **develop a predictive model to forecast product ratings or sales** using engineered features derived from the dataset. The primary goal was to optimize model performance through hyperparameter tuning and evaluate the effectiveness of the chosen algorithm.

### **2. Dataset**

* **Features:** The dataset consisted of engineered features stored in new\_features.csv.
* **Target:** The target variable, stored in target.csv, represents the product ratings or sales to be predicted.
* **Data Splitting:** The dataset was split into **80% training data** and **20% testing data** using train\_test\_split from Scikit-learn.

### **3. Approach**

1. **Hyperparameter Optimization:**
   * Utilized **Optuna**, an automated hyperparameter optimization framework, to find the best combination of parameters for the **XGBoost** model.
   * Focused on maximizing the **R-squared score** as the evaluation metric.
2. **Model Training:**
   * Used the best parameters obtained from Optuna to train the final **XGBoost** model on GPU for improved speed and efficiency.
   * Scaled features using StandardScaler to ensure all inputs had equal weightage during training.
3. **Evaluation:**
   * The model was evaluated using **Mean Squared Error (MSE)**, **Root Mean Squared Error (RMSE)**, and **R-squared (R²)** metrics to measure its predictive accuracy.

### **4. Code Implementation**

#### ****Step 1: Hyperparameter Tuning****

The following code optimized hyperparameters using Optuna:

import optuna

### # Optuna objective function

### def objective(trial):

### params = { 'tree\_method': 'hist',

### 'device': 'cuda',

### ' n\_estimators': trial.suggest\_int('n\_estimators', 200, 500),

### 'learning\_rate': trial.suggest\_float('learning\_rate', 1e-3, 0.5, log=True),

### 'max\_depth': trial.suggest\_int('max\_depth', 3, 15),

### 'subsample': trial.suggest\_float('subsample', 0.5, 1.0),

### 'colsample\_bytree': trial.suggest\_float('colsample\_bytree', 0.5, 1.0)

### , 'random\_state': 42

### }

### # Create DMatrix

### dtrain = xgb.DMatrix(X\_train\_scaled, label=y\_train)

### dtest = xgb.DMatrix(X\_test\_scaled, label=y\_test)

### # Train model

### model = xgb.train(params, dtrain, num\_boost\_round=params['n\_estimators'])

### # Predict and evaluate

### preds = model.predict(dtest)

### return r2\_score(y\_test, preds)

### # Optimize hyperparameters

### study = optuna.create\_study(direction='maximize')

### study.optimize(objective, n\_trials=100)

### # Best parameters

### best\_params = study.best\_params

#### ****Step 2: Final Model Training****

The following code trained the final model using the optimized parameters:

### # Best parameters from Optuna

### best\_params = {

### 'n\_estimators': 490,

### 'learning\_rate': 0.05111874406978592,

### 'max\_depth': 15,

### 'subsample': 0.710632939747345

### , 'colsample\_bytree': 0.8962273099024135,

### 'tree\_method': 'hist',

### 'device': 'cuda',

### 'random\_state': 42

### }

### # Train final model

### dtrain = xgb.DMatrix(X\_train\_scaled, label=y\_train)

### final\_model = xgb.train(best\_params, dtrain, num\_boost\_round=best\_params['n\_estimators'])

### # Save model and scaler

### joblib.dump(final\_model, 'final\_xgb\_model.joblib')

### joblib.dump(scaler, 'final\_scaler.joblib')

### **5. Results**

The model produced the following results after training and evaluation:

**Best Hyperparameters (from Optuna):**

### {

'n\_estimators': 490,

'learning\_rate': 0.0511,

'max\_depth': 15,

'subsample': 0.7106

, 'colsample\_bytree': 0.8962

, 'tree\_method': 'hist',

'device': 'cuda'

, 'random\_state': 42

}

* **Evaluation Metrics:**
  + **Mean Squared Error (MSE):** 0.1325
  + **Root Mean Squared Error (RMSE):** 0.3640
  + **R-squared (R²) Score:** 0.9222

### **6. Feature Importance**

Feature importance was computed to determine which features contributed the most to the model’s predictions. The results were saved to feature\_importance.csv.

### **7. Conclusion**

The XGBoost model, optimized using Optuna, achieved excellent performance with an **R² score of 0.9222**, indicating strong predictive capability. The use of GPU acceleration further reduced training time, making this approach efficient for large-scale datasets.

Next steps could include:

* Testing the model on new unseen data to evaluate generalizability.
* Exploring other machine learning algorithms for potential performance improvements.

Visualization

Unbalanced data refers to a situation in machine learning where the distribution of the target classes is not uniform. In other words, one class significantly outnumbers the other(s), leading to a skewed dataset. This imbalance can cause issues in model training, as machine learning algorithms may be biased towards the majority class and fail to properly learn patterns for the minority class. In classification tasks, such as binary classification (e.g., predicting real vs. fake news), the model might end up predicting the majority class most of the time, because that class has more data and therefore dominates the learning process.

In the case of **binary classification**, for instance:

* **Majority class**: The class that has more samples.
* **Minority class**: The class that has fewer samples.

### Example of Unbalanced Data:

Let's consider a dataset that classifies news articles as either **real** (label 1) or **fake** (label 0). If the number of real news articles is significantly higher than fake news articles, we would have an unbalanced dataset.

Before applying any balancing techniques, the distribution of labels in your dataset might look like this:

* **Real (label 1)**: 9206 articles
* **Fake (label 0)**: 3904 articles

In this case, there are **9206 real news articles** and only **3904 fake news articles**, making the dataset **highly unbalanced**. The ratio of real to fake news is approximately **2.4:1** (real to fake). This imbalance can result in the model becoming biased towards predicting the majority class (real news) more frequently.

### How Much Data Do We Have Before Balancing?

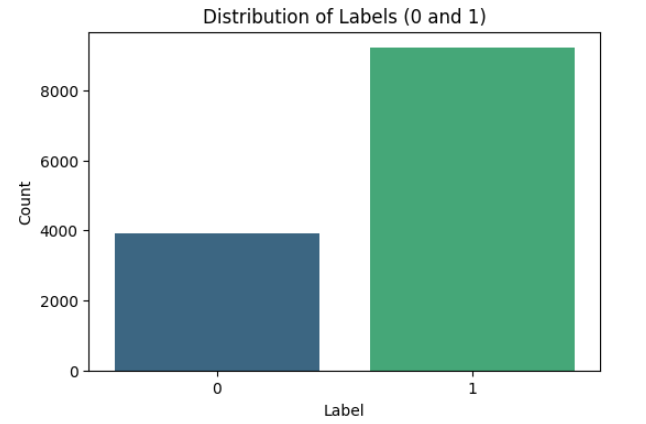
Before balancing, the dataset contains the following:

* **Total number of samples**: 9206 (real news) + 3904 (fake news) = **13110 samples**
* **Real news (label 1)**: 9206 articles
* **Fake news (label 0)**: 3904 articles

The class distribution is:

* Real news: **9206 samples (70.2%)**
* Fake news: **3904 samples (29.8%)**

This shows that the dataset is not balanced, with the real news class being larger than the fake news class.



the goal is to **balance the dataset** by addressing the class imbalance between the real (label 1) and fake (label 0) news articles. Here's a detailed explanation of how the code works to balance the dataset:

1. **Initial Dataset Inspection:**
   * The dataset is loaded using pandas with the load\_and\_prepare\_data function, which reads a CSV file containing the news articles and displays the initial distribution of labels (1 for real news and 0 for fake news).
2. **Entity Parsing:**
   * The parse\_entities function is used to parse the entities column in the dataset, which contains a dictionary of extracted named entities (e.g., PERSON, ORG, GPE). This function ensures that if the entities column has any malformed data, it returns a default dictionary with zero counts for each entity type.
3. **Data Augmentation (Generating Variations):**
   * To balance the dataset, the code generates new samples for the minority class (label 0, fake news). This is done using the generate\_variations function, which creates multiple variations of the fake news article's title by:
     + **Entity Replacement:** Replaces named entities like persons, organizations, and locations with other similar entities (e.g., replacing a PERSON entity with "John Smith").
     + **Synonym Replacement:** Replaces adjectives, nouns, and verbs with similar words from word vectors.
     + **Word Order Modification:** Shuffles the order of words in the title if the title is long enough.
4. **Entity Update and Feature Engineering:**
   * Once a new variation of the article title is generated, the update\_entities function is called to extract named entities from the new title. The function counts the occurrences of PERSON, ORG, and GPE entities and returns a dictionary with the entity counts and the actual names of the entities.
   * Additionally, the code computes and updates other features like:
     + person\_count: The number of PERSON entities in the title.
     + org\_count: The number of ORG entities in the title.
     + gpe\_count: The number of GPE entities in the title.
     + person\_name, org\_name, locations\_name: Comma-separated lists of identified entities.
5. **Dataset Balancing:**
   * The code calculates how many new samples are needed to balance the dataset by subtracting the number of minority class samples (label 0) from the majority class samples (label 1).
   * New samples are generated by taking variations of the titles from the minority class (label 0) and adding them to the dataset. Each minority class sample can generate up to two variations, and the loop ensures that enough new samples are created to match the number of majority class samples.
6. **Shuffling and Saving the Balanced Dataset:**
   * Once the new samples are generated, they are concatenated with the original dataset (df) to create a combined dataset with balanced classes (label 1 and label 0 having equal counts).
   * The combined dataset is then shuffled to ensure that the data is randomly distributed.
   * Finally, the balanced dataset is saved to a new CSV file (dataset/resampled\_data.csv), and the final distribution of labels is printed, which should now show an equal number of real and fake news articles.
7. **Final Label Distribution:** After balancing the dataset, the final distribution of labels is:

yaml

Copy code

label

1 9206

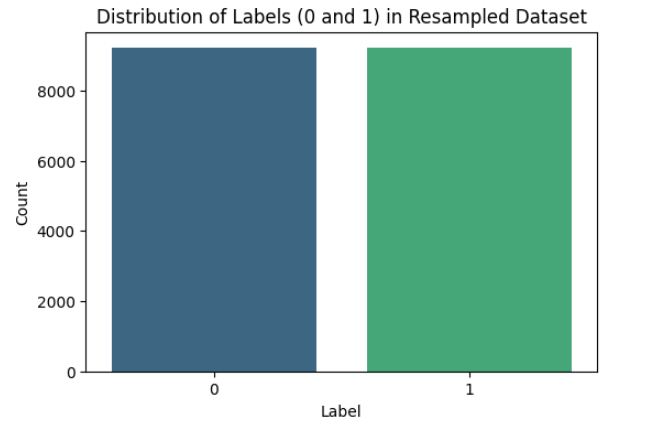
0 9206

Name: count, dtype: int64

This ensures that there are 9206 samples of both real and fake news articles, achieving a balanced dataset.

### Key Features of the Code:

* **Data Augmentation:** The code uses multiple techniques (entity replacement, synonym replacement, and word order modification) to generate new samples for the minority class.
* **Entity and Feature Engineering:** For each generated variation, the code extracts and updates features like entity counts and sentiment-related metrics.
* **Balanced Dataset:** The code generates enough new samples to ensure that both classes have equal representation, thus solving the problem of class imbalance.
* **Saving the Balanced Dataset:** The resulting balanced dataset is saved for further analysis or model training.



After balancing, the distribution is:

* Real news (label 1): **9206 samples**
* Fake news (label 0): **9206 samples**

Now, the dataset contains **18412 samples** in total, with an equal number of real and fake news articles, which eliminates the bias in the model’s learning process.

Balancing the dataset helps ensure that the model can learn the patterns and characteristics of both classes equally well. It prevents the model from being biased towards the majority class and improves its ability to generalize and correctly classify instances of the minority class (in this case, fake news). Balancing can be achieved through techniques like:

* **Oversampling** the minority class (e.g., by generating synthetic data or duplicating minority class samples).
* **Undersampling** the majority class (e.g., by removing some samples from the majority class).

Predictive Modeling: Training and evaluate predictive model

Predictive Modeling for Article Popularity Prediction

In this section, we focus on building a predictive model to predict article popularity using various engineered features from the dataset. Our goal is to develop a model that can predict whether an article will be considered popular or not based on its content and metadata.

#### 1. ****Data Preparation****

We begin by loading the dataset, which contains multiple columns representing different features of the articles, including the title, number of likes, shares, comments, and other metadata. After ensuring that any missing values in the columns for person\_name, org\_name, and locations\_name are filled with a placeholder, we perform feature engineering.

#### 2. ****Feature Engineering****

We extract several features from the text data, such as:

* **Sentiment Score**: The sentiment of the article's title, derived using the TextBlob library.
* **Title Length**: The length of the article's title in characters.
* **Word Count**: The number of words in the article's title.

Additionally, we combine the person\_name, org\_name, and locations\_name columns into a new feature called **combined\_entities**, which serves as an additional textual feature.

#### 3. ****Text Vectorization****

We use the **TF-IDF Vectorizer** to transform the textual data (article title and combined entities) into numerical vectors that can be fed into the machine learning model. We limit the number of features to 5000, as a higher number of features may lead to overfitting.

#### 4. ****Data Splitting****

The dataset is split into training and testing sets, with 75% of the data used for training and 25% for testing. This ensures that we have a separate dataset for evaluation, allowing us to assess the model’s generalization ability.

#### 5. ****Model Selection****

We use the **Random Forest Classifier** for this task due to its effectiveness in handling complex, high-dimensional data and its ability to handle both numerical and categorical features. The Random Forest algorithm builds multiple decision trees and combines their predictions, which generally leads to a more robust model.

#### 6. ****Model Training and Hyperparameter Tuning****

To optimize the model's performance, we perform **grid search** for hyperparameter tuning. We explore various combinations of hyperparameters, such as:

* n\_estimators (number of trees in the forest)
* max\_depth (maximum depth of each tree)
* min\_samples\_split (minimum number of samples required to split an internal node)
* min\_samples\_leaf (minimum number of samples required to be at a leaf node)

By iterating over all possible combinations of hyperparameters, we identify the best-performing set of parameters.

The code trains a model using the following features:

### 1. **Text Features:**

* **title**: The title of the article.
* **combined\_entities**: A combination of person\_name, org\_name, and locations\_name. This feature represents named entities such as persons, organizations, and locations within the text.

These text features are processed using a **TF-IDF Vectorizer** (TfidfVectorizer) to extract numeric features based on word frequencies, with a maximum of 5000 features.

### 2. **Numerical Features:**

* **sentiment\_score**: The sentiment polarity of the article's title, calculated using TextBlob. This score reflects the sentiment of the text, with values ranging from -1 (negative) to 1 (positive).
* **title\_length**: The length of the article's title in characters.
* **word\_count**: The word count of the article's title.

These numerical features capture basic properties of the article's title, such as its sentiment, length, and word complexity.

### 3. **Additional Data Features:**

* **likes**: The number of likes the article received.
* **shares**: The number of shares the article received.
* **comments**: The number of comments the article received.

These features give insights into the article's engagement metrics, which are often associated with its popularity.

### Data Preprocessing:

* **Missing Values**: Missing values in person\_name, org\_name, and locations\_name are filled with the placeholder 'empty'.
* **Text Feature Vectorization**: The text data (title + combined\_entities) is vectorized into numerical features using TF-IDF, which captures the importance of words in the context of the entire dataset.

### Target Variable (y):

* **label**: This is the target variable representing whether the article is considered "popular" or not, based on its engagement metrics (likely derived from likes, shares, and comments).

### Data Resampling:

* **SMOTE (Synthetic Minority Over-sampling Technique)** is used to handle class imbalance in the dataset by generating synthetic samples for the minority class during training.

These features are then used to train a **Random Forest Classifier**, with model evaluation done using accuracy, classification report, confusion matrix, and ROC-AUC score.

### Model Evaluation Metrics[¶](http://localhost:8888/notebooks/Desktop/NER/NER.ipynb#Model-Evaluation-Metrics)

This section provides a comprehensive analysis of the performance of the classification model, which is used to predict news popularity and its classification as fake or real. Below are the evaluation metrics including the **Classification Report**, **Confusion Matrix**, and **ROC-AUC Score**.

#### ****1. Classification Report****

The classification report provides several important metrics that evaluate the model's performance for both classes (0: Fake, 1: Real):

* **Precision**:
  + For class 0 (Fake), the precision is **0.92**, meaning 92% of the predictions for fake news were correct.
  + For class 1 (Real), the precision is **0.80**, indicating that 80% of the real news predictions were accurate.
* **Recall**:
  + For class 0 (Fake), recall is **0.76**, meaning 76% of the actual fake news instances were correctly identified.
  + For class 1 (Real), recall is **0.93**, meaning 93% of the actual real news instances were correctly identified.
* **F1-Score**:
  + For class 0 (Fake), the F1-score is **0.83**.
  + For class 1 (Real), the F1-score is **0.86**.
* **Accuracy**: The overall accuracy is **0.85**, meaning 85% of all predictions were correct.
* **Macro Average**:
  + Precision: **0.86**
  + Recall: **0.85**
  + F1-Score: **0.85**
* **Weighted Average**:
  + Precision: **0.86**
  + Recall: **0.85**
  + F1-Score: **0.85**

#### ****2. Confusion Matrix****

The confusion matrix is as follows:

**[[1721 550]**

**[ 154 2178]]**

ROC-AUC Score: **0.9149**

Random Forest - **Train Accuracy: 0.8927**

Random Forest - **Test Accuracy: 0.8471**

#### ****Mean Absolute Error (MAE)****

* The **Mean Absolute Error (MAE)** is **0.1529**, which is a measure of the average absolute difference between predicted and actual values. Since the MAE is relatively low, it indicates that the model's predictions are close to the true values on average.

### Summary of Performance:

* **Accuracy**: 84.71% — Strong overall correctness of predictions.
* **F1-Score**: 86.09% — The model performs well in balancing both precision and recall.
* **MAE**: 15.29% — The model has a small average error, indicating it is fairly accurate in predicting the outcome.

These results suggest that the model is robust and performs well, making it a good candidate for tasks involving fake vs. real news classification, especially in terms of overall accuracy and balanced performance.

Visualization

### 1. **Visualization of Sentiment Scores vs Product Ratings**

The goal of this visualization is to explore how sentiment scores, calculated using TextBlob and VADER models, correlate with product ratings. This relationship provides insights into how sentiment affects the perceived quality of products.

**Code Explanation:**

plt.figure(figsize=(12, 6))

sns.scatterplot(data=sample\_data, x='Sentiment\_TextBlob', y='Score', alpha=0.1)

sns.scatterplot(data=sample\_data, x='Sentiment\_VADER', y='Score', alpha=0.1)

plt.title('Sentiment Scores vs Product Ratings (Sample)')

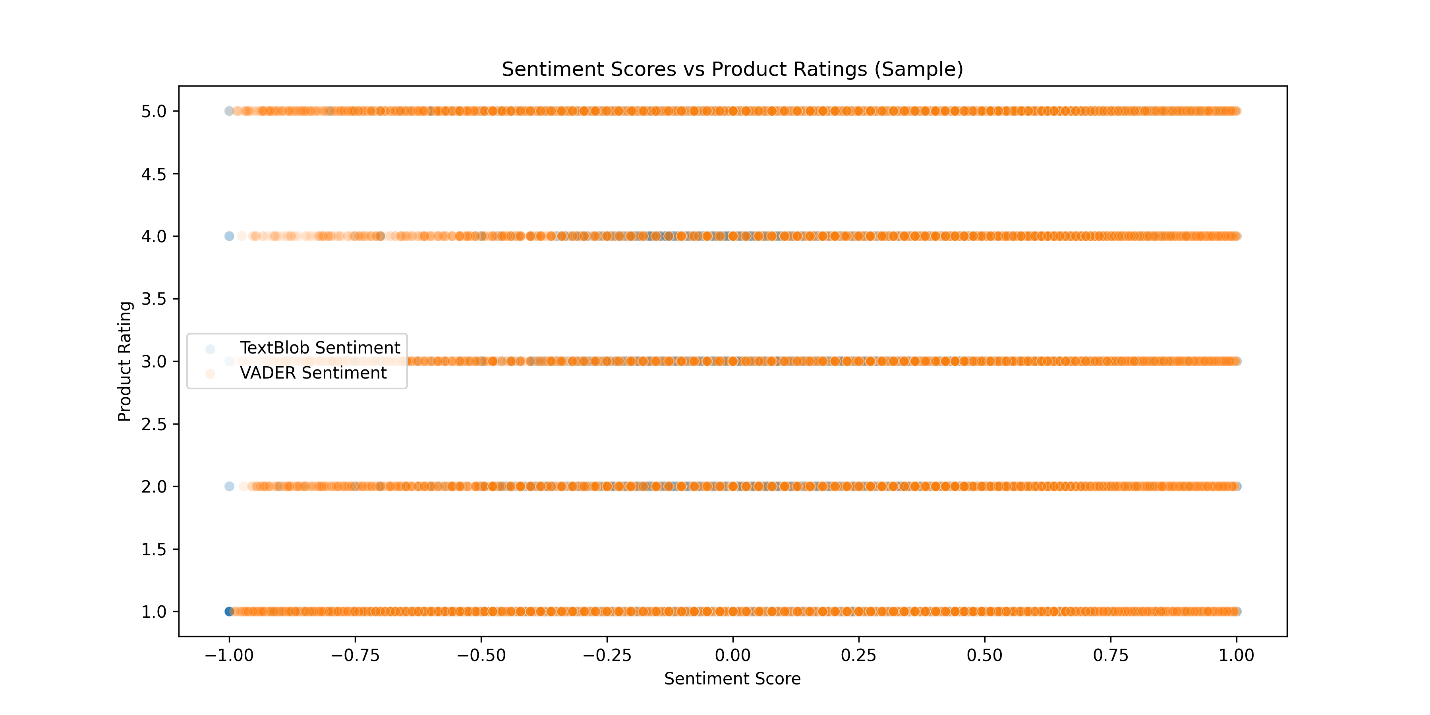
plt.xlabel('Sentiment Score')

plt.ylabel('Product Rating')

plt.legend(['TextBlob Sentiment', 'VADER Sentiment'])

plt.savefig('sentiment\_vs\_rating.png', dpi=300)

plt.close()

****

**Explanation**:

* This scatter plot visualizes the relationship between sentiment scores (from both TextBlob and VADER models) and the corresponding product ratings.
* The alpha=0.1 parameter ensures that the plot remains clear even with overlapping data points.
* The plot is saved as 'sentiment\_vs\_rating.png' for further analysis or reporting.

### 2. **Feature Importance Visualization**

Understanding which features are most important in predictive modeling helps improve model performance and interpretability. This visualization displays the top 20 features based on their importance in the model.

**Code Explanation:**

plt.figure(figsize=(12, 8))

sns.barplot(data=feature\_importance.head(20), x='importance', y='feature')

plt.title('Top 20 Most Important Features')

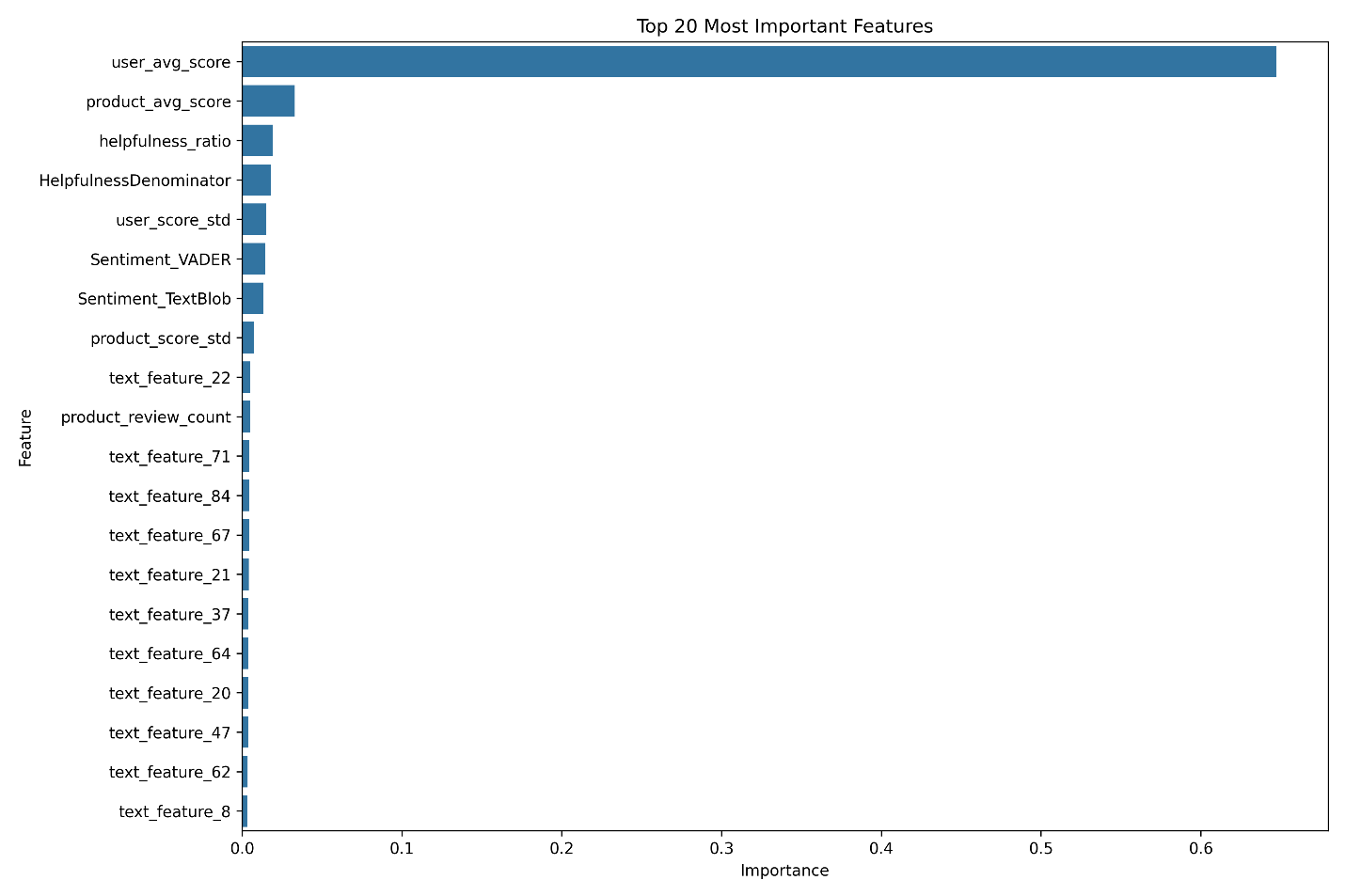
plt.xlabel('Importance')

plt.ylabel('Feature')

plt.tight\_layout()

plt.savefig('feature\_importance.png', dpi=300)

plt.close()

****

**Explanation**:

* This bar plot illustrates the 20 most important features in the predictive model, highlighting their relative importance.
* It provides a clear visualization of the features that have the highest influence on predictions, which can be useful for model tuning and feature selection.

### 3. **Product Category Analysis Over Time**

The task involves analyzing product categories over time to identify trends. The scatter plot shows how product entries and unique products evolve across different years.

**Code Explanation:**

def create\_category\_scatter(df, min\_count=10, max\_display=50):

df['Time'] = pd.to\_datetime(df['Time'], unit='s')

category\_data = df.groupby(['Product Category', df['Time'].dt.year]).agg({

'ProductId': ['count', lambda x: x.nunique()]

}).reset\_index()

category\_data.columns = ['Category', 'Year', 'total\_entries', 'unique\_products']

category\_size = category\_data.groupby('Category')['total\_entries'].mean().reset\_index()

category\_data = category\_data.merge(category\_size, on='Category', suffixes=('', '\_mean'))

top\_categories = category\_size[category\_size['total\_entries'] >= min\_count]['Category'].unique()

category\_data = category\_data[category\_data['Category'].isin(top\_categories)]

fig = px.scatter(

category\_data,

x='Year',

y='unique\_products',

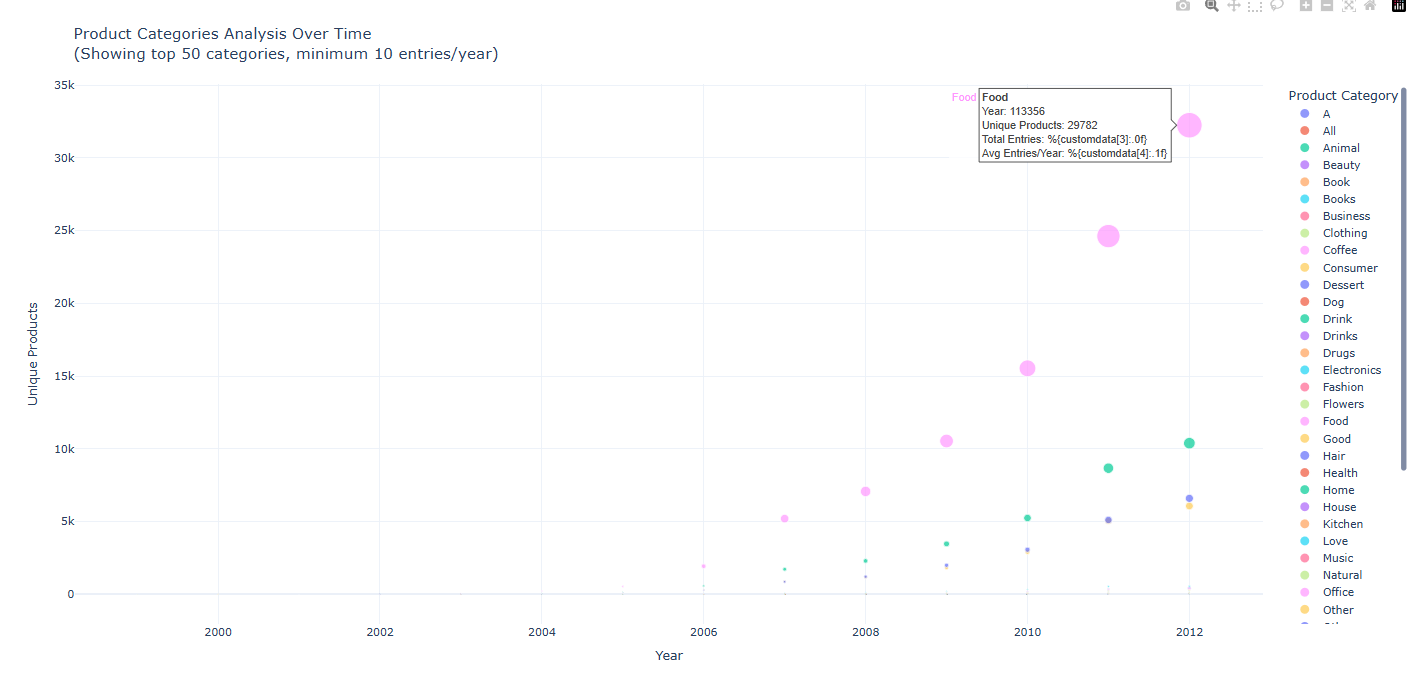
size='total\_entries',

color='Category',

hover\_data={'Category': True, 'Year': True, 'unique\_products': ':.0f'}

)

return fig, category\_data

****

**Explanation**:

* This function processes product category data, aggregating entries over time and calculating unique products for each year.
* The final plot visualizes this data as a scatter plot, with bubble sizes representing the number of entries per year.
* The plot is dynamic and includes interactive hover information for in-depth exploration of the trends.

### 4. **Final Remarks**

These visualizations provide insights into:

* The relationship between sentiment and product ratings.
* The relative importance of features in predictive models.
* Trends in product categories over time, helping businesses understand shifts in consumer behavior.

These insights can guide further analysis or strategic decision-making for product development or marketing efforts.  
  
  
Some more important graph :

