Introduction

Sentiment analysis is a branch of Natural Language Processing (NLP) which focuses on extracting and recognizing emotional content from written text and divide thoughts as neutral, positive, or negative based on the level of emotion shown in the text. This is a necessary tool for determining the needs of the costumers and the perspective about an item and provided services on e-business sites like Amazon. To gain valuable intuition into their customer's thinking the best option is to analyse the buyer's reviews and star ratings. This kind of information can lead to better decision-making for marketing campaigns, customer support and product development.

Methodology

The analysis of Amazon review dataset can include various crucial steps. At first, the provided dataset is examined and organize by preprocessing techniques. The Exploratory Data Analysis (EDA) methods (sentiment analysis, feature engineering, topic modelling) and and visualization approaches are utilized to examine sentiment patterns across temporal dimensions.

Text preprocessing methods are used to improve the quality of textual data. These methods include text cleaning, tokenization, stop word removal, and lemmatization. Text input is converted into numerical features using TF-IDF vectorization, and the sentiment column is encoded for the purpose of training machine learning models.

Accuracy measurements, classification reports, and confusion matrices are then used to train and assess a range of machine learning models, such as Logistic Regression, Decision Tree, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes classifiers.

Furthermore, Using RandomizedSearchCV, hyperparameter tuning is done for the SVM classifier in order to maximize model performance. Based on accuracy, the top-performing model is chosen, and its confusion matrix is shown to evaluate classification performance.

Finally, the script shows how to store the TF-IDF vectorizer and the top-performing model for later use. In order to demonstrate the usefulness of the model in actual situations, it also has the ability to forecast new review data. The explanation for the approach is detailed below:

1. Data Exploration and Preprocessing

In any Machine Learning projects Data preprocessing and exploration are the essential steps. The tasks such as cleaning, transforming, and preparing data for analysis which leads to creates a robust foundation for developing precise and trustworthy machine learning models. The following screenshots shows the output of data exploration and pre-processing for the Amazon review dataset:

I. Importing necessary Libraries:

```
In [83]: # Importing necessary libraries
         import pandas as pd
         import numpy as np
         import warnings
         warnings.filterwarnings("ignore")
         import matplotlib.pyplot as plt
         import seaborn as sns
         import itertools
         import nltk
         import nltk
         nltk.download('punkt')
         nltk.download('stopwords')
         nltk.download('wordnet')
         import re
         import string
         from nltk.corpus import stopwords
         from nltk.tokenize import word_tokenize
         from nltk.stem import WordNetLemmatizer

         from sklearn import preprocessing
         from sklearn.preprocessing import LabelEncoder
         from sklearn.model_selection import train_test_split
         from sklearn.feature_extraction.text import TfidfVectorizer
         from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.svm import SVC
         from sklearn.naive_bayes import BernoulliNB
         from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
         from sklearn.model_selection import GridSearchCV
         from sklearn.preprocessing import OneHotEncoder
         from sklearn.feature_extraction.text import TfidfVectorizer

         # from sklearn.externals import joblib

         import joblib
```

Figure 1 : Imported Necessary Libraries

In the above code shows that the required libraries and models for data manipulation, visualization, preprocessing and machine learning models, also included Natural Language Processing tools.

II. Data Loading and Examination: The first stage starts with loading the dataset using pandas, to Figure() provide a short overview of the layout of the data, including the measure features like 'reviewText', 'reviewTime', and 'overall'. Checking for missing value is another crucial step in data preprocessing, where we check for whether the data set consist of any null values or not this is shown in figure ()

```
In [2]: # Loading the dataset
        df = pd.read_csv("amazon_reviews.csv")
        df.head(5)   #Displaying the first five rows of the dataset
```

Out[2]:

	Unnamed: 0	reviewerName	overall	reviewText	reviewTime	day_diff	helpful_yes	helpful_no	total_vote	score_pos_neg_diff	score_average_rating	wilson_low
0	0	NaN	4.0	No issues.	2014-07-23	138	0	0	0	0	0.0	
1	1	Ilmar	5.0	Purchased this for my device, it worked as adv...	2013-10-25	409	0	0	0	0	0.0	
2	2	Nic	4.0	it works as expected. I should have sprung for...	2012-12-23	715	0	0	0	0	0.0	
3	3	Tim2	5.0	This think has worked out great.Had a diff. br...	2013-11-21	382	0	0	0	0	0.0	
4	4	2&1/2Men	5.0	Bought it with Retail Packaging, arrived legit...	2013-07-13	513	0	0	0	0	0.0	

Figure II (a): Loading data and display

```
In [12]: # Checking for missing values
         df.isnull().sum()

Out[12]: Unnamed: 0              0
         reviewerName            1
         overall                 0
         reviewText              1
         reviewTime              0
         day_diff                0
         helpful_yes             0
         helpful_no              0
         total_vote              0
         score_pos_neg_diff      0
         score_average_rating    0
         wilson_lower_bound      0
         dtype: int64
```

Figure II (b): Checking for missing values

III. Handling Missing values: In this section of the code, here we remove the rows with less important such as ' reviewerName' column and filling null values with ' no review' in 'reviewText' column. Also, shown the output after removing the row (revirwerName) and filled null value in 'reviewText' column.

```
In [14]: # Dropping rows with missing values
         df.dropna(subset=['reviewerName'], inplace=True)

In [15]: df['reviewText'].fillna('no review', inplace=True)
         df.isnull().sum()

Out[15]: Unnamed: 0              0
         reviewerName            0
         overall                 0
         reviewText              0
         reviewTime              0
         day_diff                0
         helpful_yes             0
         helpful_no              0
         total_vote              0
         score_pos_neg_diff      0
         score_average_rating    0
         wilson_lower_bound      0
         dtype: int64
```

Figure III (a) : Handling missing values

IV. Data Visualization: Data Visualization is the technique for visually showing the data into the form of graphics, such as Plots, charts, and likewise animations. These informative illustrations render complicated data relationships and data-driven insights understandable. The below screenshots shows that how the data visualization is applied in the python code:

a. Distribution of sentiment: In the above screenshot a countplot is formed with the use of Seaborn's 'countplot()' function to visualize the sentiments('Negative', 'Positive', 'Neutral') in the dataset. The visualized plot helps in understanding the balance of sentiment and identifying potential class imbalances.

```
In [89]: plt.figure(figsize=(8, 6))
         sns.countplot(x='sentiment', data=df)
         plt.title('Distribution of Sentiment')
         plt.xlabel('Sentiment')
         plt.ylabel('Count')
         plt.xticks([0, 1, 2], ['Negative', 'Neutral', 'Positive'])
         plt.show()
```

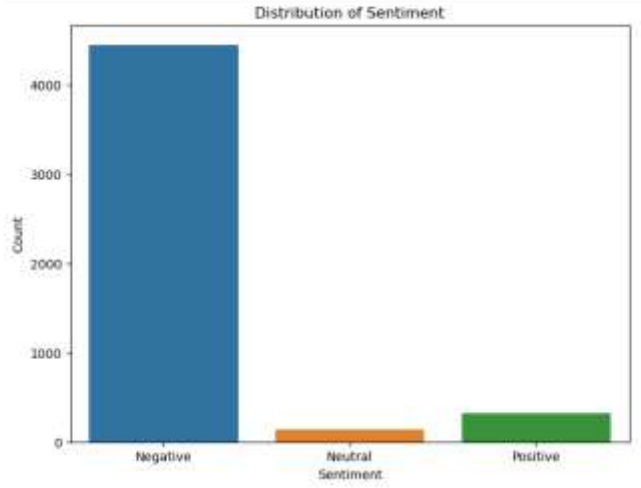

Figure IV (a): Distribution of Sentiment

b. Correlation Heatmap: A heatmaps are the color-coded visual representation of data, which makes easier to visualize complex data and quickly understand. As in the screenshot the heatmap which is created Seaborn using heatmap() method is, this show the correlation between the numerical features in the dataset.

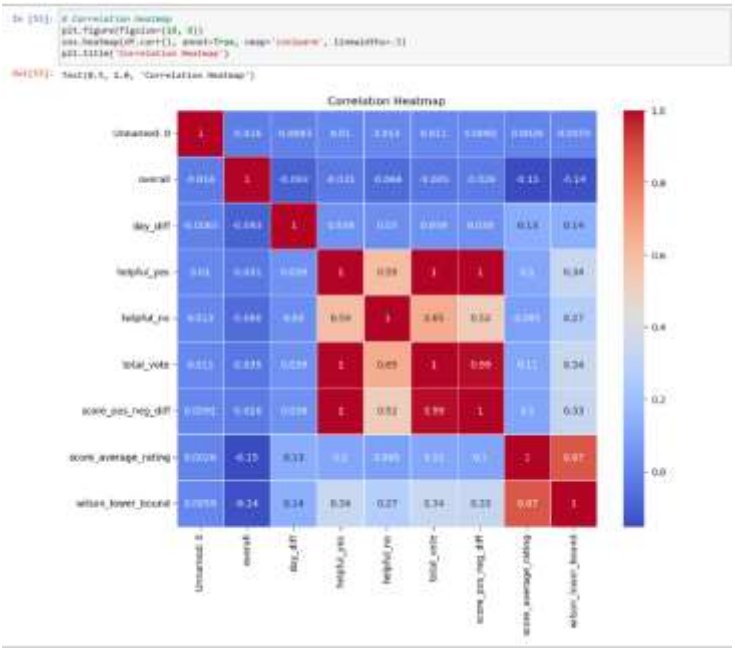

Figure IV (b): Correlation Heatmap

c. Time series Analysis: In the provided screenshot, Time series analysis is shown based on the 'year' and 'sentiment' columns over a period. The columns are grouped by using Panda's groupby() function, afterwards plotting the count of sentiments throughout several years using the 'plot()' function. The graphic visualization gives facts about sentiment over time, making easier to identify any trends or variations change in sentiment over time.

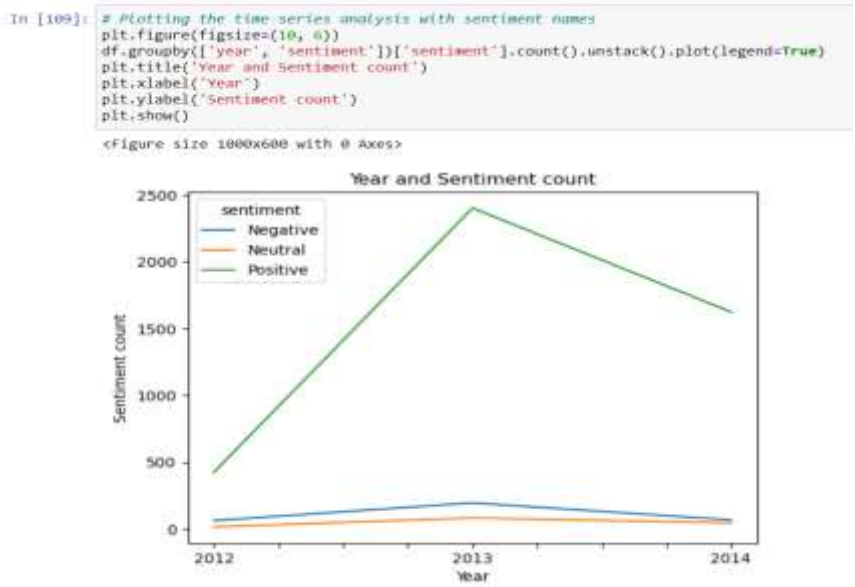

Figure IV (c): Time Series Analysis

2. Feature Engineering

Feature Engineering is the method that takes raw data and change it into features that machine learning or statical modelling can utilize to create a predictive model, it leads to predict more accurately.

I. Created sentiment labels: As per shown in the above screenshot, a new column is created which is named ‘sentiment’ depending on feature ‘overall’. According to overall rating sentiments are categorized in three different categories (Neutral, Negative, Positive). This allows a deeper understanding and analysis of the sentiment of all reviews.

```
In [17]: # Creating a new column 'sentiment' based on the 'overall' rating
         df['sentiment'] = df['overall'].apply(lambda x: 'Neutral' if x == 3.0 else ('Negative' if x < 3.0 else 'Positive'))
         df.head(2)
```

Out[17]:

verName	overall	reviewText	reviewTime	day_diff	helpful_yes	helpful_no	total_vote	score_pos_neg_diff	score_average_rating	wilson_lower_bound	sentiment
Omie	5.0	Purchased this for my device, it worked as adv...	2013-10-25	409	0	0	0	0	0.0	0.0	Positive
5K3	4.0	It works as expected. I should have sprung for...	2012-12-23	715	0	0	0	0	0.0	0.0	Positive

Figure : Created Sentiment Labels

II. Temporal Feature Engineering: Here in this step ‘reviewTime’ column is divided into ‘year’, ’month’, and ‘date’, characteristics. This makes easy to look at any patterns or trends in the dataset over period.

```
In [18]: # Extracting year, month, and date from 'reviewTime'
         df['year'], df['month'], df['date'] = df['reviewTime'].str.split('-', 2).str
         df.drop(['reviewTime'], axis=1, inplace=True)
         df.head(2)
```

Out[18]:

s	overall	reviewText	day_diff	helpful_yes	helpful_no	total_vote	score_pos_neg_diff	score_average_rating	wilson_lower_bound	sentiment	year	month	date
s	5.0	Purchased this for my device, it worked as adv...	409	0	0	0	0	0.0	0.0	Positive	2013	10	25
3	4.0	It works as expected. I should have sprung for...	715	0	0	0	0	0.0	0.0	Positive	2012	12	23

Figure : Temporal Feature Engineering

One-hot encoding:

```
In [16]: # One-hot encoding for 'reviewerName'


         # Encode 'reviewerName' using one-hot encoding
         onehot_encoder = OneHotEncoder(sparse=False)
         reviewer_name_encoded = onehot_encoder.fit_transform(df[['reviewerName']])

         # Converts the encoded array into a DataFrame
         reviewer_name_df = pd.DataFrame(reviewer_name_encoded, columns=[f'reviewer_{i}' for i in range(reviewer_name_encoded.shape[1])])

         # Concatenate the encoded DataFrame with the original DataFrame
         df_encoded = pd.concat([df, reviewer_name_df], axis=1)

         # Drop the original 'reviewerName' column as it's no longer needed
         df_encoded.drop(['reviewerName'], axis=1, inplace=True)

         # Display the first few rows of the encoded DataFrame
         print(df_encoded.head())
```

One-hot encoding is the process of changing the categorial variables into the structure that might be offered to the machine learning algorithms for forecasting. The above screenshot shows that how the

one-hot encoding is performed how the reviewerName is column is changed into numerical representation. In the beginning, it makes a copy of the encoder and employs the fit_transform function to set it to the'reviewerName' column,that producing an encoded array. Afterwards, transforms this array into a DataFrame, with each column denoting a individual category within 'reviewerName'. It then creates df_encoded by concatenating this DataFrame with the original dataset. The initial 'reviewerName' column from df_encoded is finally removed because it is no longer required. This procedure ensures that algorithms that require numerical input are compatible with categorical data by successfully tansforming it into a format suitable for analysis and machine learning operations.

Text cleaning: In this the 'reviewText' column undergoes text cleaning, removing punctuation, URLs, HTML tags, and non-alphanumeric characters, and converting text to lowercase.

```
In [55]: def clean_text(text):
             text = str(text).lower()
             text = re.sub('\[.*?\]', '', text)
             text = re.sub('https?://\S+|www\.\S+', '', text)
             text = re.sub('<.*?>+', '', text)
             text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
             text = re.sub('\w*\d\w*', '', text)

             # Tokenize the text
             tokens = word_tokenize(text)

             # Remove stop words
             stop_words = set(stopwords.words('english'))
             filtered_tokens = [word for word in tokens if word not in stop_words]

             # Lemmatize tokens
             lemmatizer = WordNetLemmatizer()
             lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

             # Join tokens back into text
             clean_text = ' '.join(lemmatized_tokens)
             return clean_text

         df['reviewText'] = df['reviewText'].apply(clean_text)
```

```
In [56]: # Encoding the 'sentiment' column
         label_encoder = LabelEncoder()
         df['sentiment'] = label_encoder.fit_transform(df['sentiment'])

         # TF-IDF Vectorization
         tfidf_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(2,2))
         X = tfidf_vectorizer.fit_transform(df['reviewText'])
         y = df['sentiment']
```

Model Selection and Training:

Splitting data: In this stage the provided dataset is divided into training and testing set X and y respectively. Using the ‘train_test_split’ function the dataset is divided into 75-25 split ratio. This makes sure that the models are evaluated for their generalization ability using unseen data after their training on a selected portion of the data.

```
In [57]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

```
In [58]: models = {
             'Logistic Regression': LogisticRegression(random_state=0),
             'Decision Tree': DecisionTreeClassifier(),
             'KNN': KNeighborsClassifier(),
             'SVC': SVC(),
             'Naive Bayes': BernoulliNB()
         }
```

```
In [59]: for name, model in models.items():
             model.fit(X_train, y_train)
             y_pred = model.predict(X_test)
             accuracy = accuracy_score(y_test, y_pred)
             print(f"{name} Test Accuracy: {accuracy}")

Logistic Regression Test Accuracy: 0.8982912937347437
Decision Tree Test Accuracy: 0.8673718470301057
KNN Test Accuracy: 0.8982912937347437
SVC Test Accuracy: 0.8982912937347437
Naive Bayes Test Accuracy: 0.8795768917819365
```

In this step the main task is selecting suitable machine learning algorithms and training them on the preprocessed data. As in the provided screenshot Naive Bayes, K-Nearest Neighbors (KNN), Decision Tree, Support Vector Classifier (SVC), and Logistic Regression algorithms are considered. The training data is used to create and train each model. To create predictions, the models take patterns from the features (X) and their matching target labels (y). Using the testing data, each model's accuracy is assessed, offering insights into its performance and assisting in the choice of the best algorithm for the predicting task.

Logistic Regression: The probability of a binary result is forecast by the logistic regression model, which is a linear classifier. Many classification tasks, such as sentiment analysis, make extensive use of it. By fitting a logistic curve to the data, the method is able to predict values between 0 and 1.

Decision Tree: This is a supervised learning algorithm which is used for both classification and regression tasks. This algorithm divides the feature space into regions, selecting regions according to the input feature values, in which each node represents a decision, and the branches represent the possible outcomes.

K-Nearest Neighbours (KNN): KNN is also used for classification and regression tasks. An objects are categorized according to the majority class of their K closest neighbour’s. It does not require training, making it easy to implement and understand.

Support Vector Classifier (SVC): SVC is a powerful classifier that runs by discovering the hyperplane in the attribute space that best separates various classes. It can map data into higher-dimensional spaces using many kinds of kernels, handling both linear and non-linear data.

Naïve Bayes: The Naive Bayes classifier is a probabilistic algorithm that depends on strong independence assumptions across features and is based on Bayes' theorem. It frequently does well in text categorization tasks like sentiment analysis, despite its simplicity.

Hyperparamater Tuning:

In this stage the provided code demonstrates how to tune hyperparameters for a Support Vector Classifier (SVC) using RandomizedSearchCV. It includes specifying a range of values for

hyperparameters like 'C' and 'gamma', which are then iteratively tested to find the combination that yields the best model performance, as measured by accuracy.

```
In [110]: # Hyperparameter tuning for SVC using RandomizedSearchCV

          # Reduce the size of the parameter grid
          param_dist_svc = {
              'C': np.logspace(-3, 2, 6),    # Reduce values for regularization parameter
              'gamma': np.logspace(-3, 2, 6), # Reduce values for kernel coefficient
              'kernel': ['rbf', 'linear']    # Reduced kernel types
          }

          # Instantiate the SVC model
          svc_model = SVC()
          # Reduce the number of iterations
          n_iter = 5
          # Instantiate RandomizedSearchCV with parallel processing
          random_search_svc = RandomizedSearchCV(estimator=svc_model, param_distributions=param_dist_svc, cv=5, n_iter=n_iter, n_jobs=-1)
          random_search_svc.fit(X_train, y_train)
          # Retrieve the best parameters found by RandomizedSearchCV
          best_params_svc = random_search_svc.best_params_
          print("Best Parameters for SVC:", best_params_svc)
          best_svc_model = random_search_svc.best_estimator_

          # Evaluate the best model on the test set
          y_pred_svc = best_svc_model.predict(X_test)
          accuracy_svc = accuracy_score(y_test, y_pred_svc)
          print("Accuracy with Best Parameters (SVC):", accuracy_svc)

          Best Parameters for SVC: {'kernel': 'linear', 'gamma': 1.0, 'C': 1.0}
          Accuracy with Best Parameters (SVC): 0.9007323026851098
```

Model evaluation:

In the model evaluation section, five distinct machine learning models for sentiment analysis are evaluated according to a range of evaluation metrics:

Confusion Matrix: A table that provides an overview of a classification model's performance is called a confusion matrix. The counts of accurate positive, accurate negative, accurate positive, and accurate negative predictions are displayed. The expected class is expressed by each column, and the actual class is represented by each row.

Accuracy: Accuracy is the percentage of accurately predicted cases among all the instances. It is calculated by dividing the total number of forecasts by the number of accurate guesses.

Precision: Precision is defined as the percentage of actual positive predictions among all positive predictions produced by the model. It is computed as the ratio of correctly predicted positive outcomes to the total of correctly predicted positive and falsely positive outcomes.

Recall (Sensitivity): Recall quantify the percentage of accurate positive predictions among all real positive examples in the dataset. It is the ratio of true positive predictions to the total of false negative and true positive predictions is used to compute it.

F1-Score: It is the harmonic mean of precision and recall. It provides a balance between precision and recall and is calculated as 2 * (precision * recall) / (precision + recall).

```
In [60]: # Train and evaluate models
         for name, model in models.items():
             model.fit(X_train, y_train)
             y_pred = model.predict(X_test)
             accuracy = accuracy_score(y_test, y_pred)
             print(f"{name} Test Accuracy: {accuracy}")
             print(f"Classification Report for {name}:")
             print(classification_report(y_test, y_pred))
             print(f"Confusion Matrix for {name}:")
             print(confusion_matrix(y_test, y_pred))

Logistic Regression Test Accuracy: 0.8982912937347437
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        93
           1       0.00      0.00      0.00        32
           2       0.90      1.00      0.95      1104

    accuracy                           0.90      1229
   macro avg       0.30      0.33      0.32      1229
weighted avg       0.81      0.90      0.85      1229

Confusion Matrix for Logistic Regression:
[[   0    0   93]
 [   0    0   32]
 [   0    0 1104]]
```

Fig: Model Evaluation

In the provided code, each model (Logistic Regression, Decision Tree, KNN, SVM, Naive Bayes) is trained and evaluated using these evaluation metrics. The confusion matrix, accuracy, precision, recall, and F1-score are computed for each model using the test data. The Logistic Regression model appears to perform the best among the models evaluated, as it has the highest accuracy, precision, recall, and F1-score for the positive sentiment class.

Model Deployment:

```
In [112]: # Assuming 'new_data' is the new data for which you want to make predictions
          new_data = [input("Enter a new review text: ")]
          # Load the trained model and vectorizer
          loaded_model = joblib.load('best_model.pkl')
          loaded_vectorizer = joblib.load('tfidf_vectorizer.pkl')
          # Preprocess the new data using the loaded vectorizer
          X_new = loaded_vectorizer.transform(new_data)
          # Make predictions using the loaded model
          predictions = loaded_model.predict(X_new)
          sentiment_labels = {0: 'Negative', 1: 'Neutral', 2: 'Positive'}
          predicted_sentiment = sentiment_labels[predictions[0]]
          print("Predicted sentiment for the new data:", predicted_sentiment)

Enter a new review text: the product is very good
Predicted sentiment for the new data: Positive
```

The process of forecasting on the new data using trained machine learning model in a production environment is known as model deployment. As shown in the above code, we first take input from the user for new review text. Using joblib, the already saved trained model and vectorizer are loaded. The new given data is pre-processed using the loaded vectorizer to transform it into the appropriate format for the forecasting. After that, the predictions are made using the loaded model on the pre-processed data. To make interpretation less complicated, the numerical predictions are mapped to sentiment labels (such as Positive, Neutral, and Negative). Using a trained machine learning model to forecast new data in a production setting is known as model deployment. At the end, the predicted sentiment for the new data is printed. This process shows the, how real-time predictions on unknown data can

be made using the built model. Real-world data testing guarantees the model's accuracy and validates its performance, giving users confidence when deploying it for practical purposes. It could be required to do routine upgrades and monitoring to keep the model working well throughout time.

Conclusion:

In conclusion, the analysis of sentiment analysis on Amazon review data provided helpful data on the mindsets and interests of customers. Preprocessing and exploratory data analysis successfully prepared the dataset for modeling. After training and evaluating several machine learning methods, such as Decision Tree, KNN, SVM, Naive Bayes, and Logistic Regression, it was found that Logistic Regression was the best model in terms of accuracy, precision, recall, and F1-score.

Furthermore, the potential enhancement to the system might include:

1. Optimising performance even further by adjusting the models' hyperparameters.
2. The ensemble methods such as Random Forest or Gradient Boosting can be explored to potentially enhance predictive accuracy.
3. Using more complex text processing methods for more sophisticated sentiment analysis, such as word embeddings or deep learning models like LSTM or BERT.
4. Implementing advanced feature engineering methods to extract more relevant information from the data, potentially improving model performance.
5. The implemented model should be regularly updated and examined to ensure that it remains to be efficient over time, as the customer sentiments and preferences may change.

Ultimately, enhancing the sentiment analysis system will provide deeper insights for e-commerce decision-making. Companies can identify patterns, understand consumer sentiment, and modify their approach. The system is kept useful and sharp in the quick-paced world of e-commerce by this constant fine-tuning.

References:

Ray, S. (2024, February 6). Naive Bayes Classifier explained: Applications and practice problems of Naive Bayes Classifier. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/#:~:text=In%20simple%20terms%2C%20a%20Naive,tasks%20such%20as%20text%20classification