PROJECT REPORT

Introduction

"This project focuses on sentiment analysis for Amazon musical instruments reviews. Sentiment analysis is a technique in natural language processing (NLP) used to classify the emotional tone of textual data—whether it is positive, negative, or neutral. The goal here is to automate sentiment classification based on customer reviews. By doing so, businesses can extract meaningful insights at scale to improve decision-making, and customers can make informed purchase choices."

"In this project, I explored the entire machine learning pipeline: data preprocessing, exploratory analysis, feature engineering, model building, and evaluation. Let me walk you through each step."

What is Sentiment Analysis?

*"Sentiment analysis identifies and classifies the sentiment expressed in text. For example:

- Positive: 'The instrument is amazing and works perfectly.'
- Negative: 'It broke after a week of use. Not worth the money.'
- Neutral: 'It's okay, but I expected better.'

"Here, I combined preprocessing techniques, exploratory data analysis, and machine learning methods to classify reviews into these sentiment categories."

Problem Statement

*"Amazon reviews often contain valuable feedback, but manually analyzing thousands of reviews is impractical. While ratings (1-5) provide some sentiment clues, textual reviews add depth and context. This project aims to:

1. Classify reviews into positive, negative, or neutral sentiment.
2. Analyze trends like how sentiment varies over time and how helpfulness correlates with sentiment.
3. Extract actionable insights from text data.

STEPS

1. **Import libraries –**

*"I used the following libraries to handle the workflow:

- pandas for data manipulation and numpy numerical computations.
- matplotlib and seaborn for creating visualizations.
- nltk and TextBlob for NLP tasks like text cleaning and polarity calculation.
- scikit-learn for machine learning and evaluation.
- imbalanced-learn for addressing class imbalance."*

"These libraries provided a comprehensive toolkit for preprocessing, feature extraction, and modeling."

2. Importing and Understanding the Dataset

This section loads a dataset of Amazon musical instrument reviews and provides basic information about it. It prints the dimensions (rows and columns) of the dataset and displays details about the data types and null values in each column.

Description of Columns

1. **reviewerID**: A unique identifier for the reviewer, e.g., A2SUAM1J3GNN3B.
2. **asin**: A unique identifier for the product being reviewed, e.g., 0000013714.
3. **reviewerName**: The name of the reviewer.
4. **helpful**: A measure of how helpful other users found the review, formatted as helpful/total helpful votes, e.g., 2/3.
5. **reviewText**: The full text of the review.
6. **overall**: The overall rating of the product, on a scale of 1 to 5.
7. **summary**: A short summary or title of the review, often derived from the full review text.
8. **unixReviewTime**: The time of the review in Unix timestamp format (seconds since January 1, 1970).
9. **reviewTime**: The human-readable date of the review (e.g., 2013-06-23).

3. Preprocessing and Cleaning -

a) Handling Missing and Null Values

This section focuses on handling missing (NaN) values in the dataset and preparing the data for further analysis. Missing values in the reviewText column are replaced with the word "Missing" to preserve the data,

[NaN stands for **"Not a Number"** and is used in data analysis to represent **missing or undefined values** in a dataset. In Pandas (Python's data analysis library), NaN values are often used as placeholders for data that is not available.]

b) Concatenating Review Text and Summary

The review text and summary columns are then combined into a single column called reviews and dropped those two columns to consolidate the textual data for sentiment analysis.

c) Creating sentiment column

This section creates a new column, **sentiment**, to classify reviews as **Positive**, **Neutral**, or **Negative** based on the overall rating given by users. Ratings greater than 3 are marked as Positive, ratings less than 3 are Negative, and ratings equal to 3 are Neutral. This preprocessing step is crucial for building a sentiment analysis model.

After that, we will count the number of reviews for each sentiment category (e.g., Positive, Neutral, Negative) in the dataset. This step provides an overview of the distribution of sentiment labels, which is essential for understanding class imbalances and planning model training.

d) Handling the TIME column

Initially, the reviewTime column was in an unstructured format, unsuitable for analysis. It was converted into a proper datetime format, and new features—year, month, and day—were extracted.

Then we added date,month and year columns into the table. The year helped analyze trends in sentiment over time, while the month revealed seasonal patterns. The day provided insights into daily review activity. These transformations made the temporal data useful for uncovering trends and patterns.

e) Finding Helpfulness of Reviews

From the main dataframe we can see the helpful feature with values in list [a,b] format. It says that a out of b people found that review helpful. But with that format, it could not add value to the machine learning model and it will be difficult to decrypt the meaning for the machine.

From the helpful column, I calculated a helpful rate:

helpful rate =Helpful Votes/Total Votes

"This quantified how useful a review was to other customers and served as an additional feature."

From this, we added helpful_rate column in the table.

f) Text Cleaning

The reviewText column contained raw, unstructured data with noise such as punctuation, special characters, and numbers, which do not contribute to sentiment analysis. To clean the text:

1. **Punctuation and Special Characters Removal**: Punctuation and symbols (e.g., @, #, !) were removed to focus only on meaningful words.
2. **Lowercasing**: All text was converted to lowercase, ensuring uniformity (e.g., 'Great' and 'great' are treated the same).
3. **Stopword Removal**:

   Coming to stop words, general nltk stop words contains words like not,hasn't,would'nt which actually conveys a negative sentiment. If we remove that it will end up contradicting the target variable(sentiment). So I have curated the stop words which doesn't have any negative sentiment or any negative alternatives. Stopwords, such as "the", "and", and "is," which don't contribute to sentiment, were removed using a custom stopword list.
4. **Numbers Removal**: Numbers were excluded as they are generally irrelevant in textual sentiment classification.
5. **Result**: The cleaned text was simpler, uniform, and more relevant for downstream analysis and modeling.

3. **Story generation and visualisation –**

In this section I had done complete exploratory data analysis on texts as well as other factors to understand what are all features which contributes to the sentiment.

Prior analysis assumptions:

1.Higher the helpful rate the sentiment becomes positive.

2.There will be many negative sentiment reviews in the 2013 and 2014 year

3.There will be more reviews at the starting of a month.

These assumptions will be verified with our plots also we will do text analysis a lot.

a) Sentiments vs Helpful_rate

sentiment	helpful_rate
Negative	0.307559
Neutral	0.275687
Positive	0.260505

This section examines the relationship between sentiment (positive, neutral, negative) and mean of helpfulness rates of reviews. From the table, It shows that negative reviews have a higher average helpfulness rate compared to neutral and positive reviews.. Further analysis, such as using a violin plot, will explore the distribution of helpfulness rates across sentiments.

b) Sentiments vs Helpful_rate violin plot

This violin plot visualizes the relationship between sentiment categories (positive, negative, neutral) and helpfulness rates. Contrary to the mean values observed earlier, the plot reveals that **positive reviews have a wider spread and higher concentration of helpfulness rates**, suggesting they are often considered more helpful. The plot emphasizes that visual analysis can sometimes provide better insights than relying solely on measures of central tendency. The assumption that positive reviews are more helpful is validated.

This (average for negative may be high but most helpful_rated review are for positive) may happen. For example –

Here are numerical examples to explain the discrepancy between the **mean values** and the **violin plot insights** for **Sentiments vs Helpful Rate**:

---

1. Negative Reviews with Higher Average Helpfulness

Example Dataset (Negative Reviews):

Review	Helpful Votes	Total Votes	Helpful Rate
R1	10	10	1.0
R2	20	25	0.8
R3	0	15	0.0
Average	**30**	**50**	**0.6**

- **Observation**: Even though some negative reviews have a helpful rate of 0, the **average helpful rate** is skewed upward by a few highly helpful reviews (e.g., 1.0 and 0.8 in R1 and R2).
- Negative reviews tend to be fewer but more detailed and critical, making them more helpful overall.

---

2. Positive Reviews with Wider Spread

Example Dataset (Positive Reviews):

Review	Helpful Votes	Total Votes	Helpful Rate
R4	50	50	1.0
R5	10	50	0.2
R6	0	30	0.0
R7	5	20	0.25
Average	**65**	**150**	**0.433**

- **Observation**: Positive reviews have a wider range of helpful rates. While some (like R4) are extremely helpful (1.0), many are unhelpful or generic (e.g., R6 and R7).
- This **spread of helpfulness rates** creates a wider violin shape but lowers the mean value compared to negative reviews.

---

3. Neutral Reviews with Consistently Low Helpful Rates

Example Dataset (Neutral Reviews):

Review	Helpful Votes	Total Votes	Helpful Rate
R8	1	10	0.1
R9	2	20	0.1
R10	0	30	0.0
Average	**3**	**60**	**0.05**

- **Observation**: Neutral reviews consistently have low helpful rates because they don't provide strong opinions or useful details.

Explaining the Discrepancy

- **Mean Values**:
    - Negative reviews have a higher mean helpfulness because detailed, critical reviews are often marked as helpful, skewing the average.
    - Positive reviews have a lower mean helpfulness because many are generic, even though some are very helpful.
    - Neutral reviews are consistently low in helpfulness.
- **Violin Plot Insights**:
    - The plot shows **positive reviews** have a higher concentration of very helpful reviews (helpful rate close to 1.0) but also a large spread of unhelpful ones, which balances out the mean.
    - **Negative reviews** show less spread, with many reviews marked as helpful but fewer unhelpful ones.

This numerical explanation highlights why negative reviews might have a higher mean helpfulness rate, but positive reviews appear more concentrated in the violin plot.

c) Year vs Sentiment count

In this block we will see how many reviews were posted based on sentiments in each year from 2004 to 2014 -

From the plot we can clearly see the rise in positive reviews from 2010. Reaching its peak around 2013 and there is a dip in 2014, All the review rates were dropped at this time. Negative and neutral reviews are very low as compared to the positive reviews. Our second assumption is wrong !

d) Day of month vs Reviews count – Bar graph

Let's check if there are any relationship between reviews and day of month

The review counts are more or less uniformly distributed. There isn't much variance between the days. But there is a huge drop at the end of month. Our third assumption is wrong ! We sd never trust your instincts unless we do EDA.

e) Creating Few More Features for Text Analysis

Polarity:
We use **TextBlob** to calculate the polarity of each review. Polarity values range between [-1, 1], where -1 indicates negative sentiment, 0 indicates neutral sentiment, and 1 indicates positive sentiment.

Review Length:
This feature calculates the total number of characters in a review, including letters and spaces, providing an idea of how detailed the review is.

Word Count:
This measures the total number of words in a review, helping identify verbosity patterns and providing insights into how much detail is expressed in the text.

f) Sentiment Polarity Distribution (polarity vs count) - Histogram

What We Did:
We plotted a histogram of the polarity scores calculated using the TextBlob library. Polarity values range from -1 to 1, representing the sentiment of each review, with -1 being highly negative and 1 being highly positive.

Insights:

1. There more positive reviews than negative ones.
2. This visualization confirms that the dataset has a significant number of positive reviews.

---

g) Review Rating Distribution (Review Rating vs count) - Histogram

What We Did:
We visualized the distribution of overall ratings (1 to 5) given by customers. This plot shows how frequently each rating appears in the dataset.

Insights:

1. The majority of the ratings are 5 stars, followed by 4, 3, 2, and 1 stars.
2. This confirms that the dataset is skewed toward higher ratings, reflecting more satisfied customers.

   Right-skewed distribution means that the majority of the data points are concentrated on the left side (lower values), while the right side (higher values) has a long "tail" with fewer data points.

h) Review Text Length Distribution (Review Length vs Count) - Histogram

What We Did:
We plotted the distribution of the lengths of reviews, measured by the total number of characters in each review. This includes letters, spaces, and punctuation.

Insights:

1. The distribution is right-skewed, with most reviews having lengths between 0 and 1000 characters.
2. A few outliers have significantly longer review lengths, which may indicate highly detailed feedback.
3. This distribution highlights that the majority of reviews are concise, with detailed reviews being less frequent.

---

i) Review Text Word Count Distribution (Word Count vs Count) - Histogram

What We Did:
We plotted the distribution of the word count for each review, representing the total number of words in a review.

Insights:

1. Similar to the review length distribution, the word count distribution is right-skewed, with most reviews containing between 0 and 200 words.
2. A small number of reviews exceed 200 words, showing that detailed reviews are less common but present.
3. This visualization indicates that the majority of reviews are brief, aligning with typical customer behavior.

h) N-Gram Analysis (Deep Text Analysis) -

We performed n-gram analysis to identify patterns and commonly used phrases in reviews. This included:

1. Unigrams: Single words (e.g., "great", "poor") to understand the most frequent terms used in reviews.
2. Bigrams: Two-word combinations (e.g., "highly recommend", "not worth") to capture short phrases that provide context.
3. Trigrams: Three-word combinations (e.g., "value for money", "would not recommend") to analyze more detailed expressions.

This analysis helped uncover repeated phrases in reviews, offering deeper insights into customer sentiment and behavior. It provided context-rich features for sentiment classification and trend analysis.

i) Word Cloud Analysis

Word clouds were generated for positive, negative, and neutral reviews to visualize the most frequently used words in each sentiment category.

- For positive reviews, words like "amazing", "quality", and "recommend" appeared prominently, reflecting satisfaction.

- In negative reviews, terms like "poor", "broken", and "disappointed" stood out, highlighting issues.
- Neutral reviews contained general words like "item", "product", "device", "instrument" with no strong sentiment.

Word clouds provided a quick, visual summary of the language patterns, making it easier to identify dominant themes in each sentiment. They also helped validate and interpret results from n-gram analysis.

4. Feature Extraction

- Feature extraction converts raw text data into numerical formats for machine learning models.
- **TF-IDF (Term Frequency-Inverse Document Frequency)** was used to represent text as numerical vectors.
- TF-IDF emphasizes unique words in reviews while reducing the weight of common terms across all reviews.
- This ensures the model focuses on important terms contributing to sentiment classification.
- Additional preprocessing, such as stemming and encoding the target variable, was performed prior to applying TF-IDF.
- These extracted features are now ready for use in machine learning models.

A) Encoding Target Variable - Sentiment

- Sentiment labels were encoded into numerical values using **LabelEncoder** (e.g., Positive → 2, Neutral → 1, Negative → 0).
- This transformation is necessary as machine learning models require numerical inputs for classification tasks.
- The distribution of sentiment classes was: 9622 positive, 772 neutral, and 467 negative reviews, reflecting a class imbalance.
- Encoding ensures that the sentiment labels are standardized and machine-readable.
- This step bridges the gap between textual labels and numerical representation.
- The encoded sentiment labels are now ready for modeling.

B) Stemming Reviews

- Stemming reduces words to their root forms by removing prefixes or suffixes (e.g., "going", "gone", "goes" → "go").
- **PorterStemmer** was used to apply stemming, ensuring consistency in the text data.
- Stopwords like "the", "is", and "and" were removed beforehand to retain meaningful terms.
- The cleaned and stemmed text data was stored in a new list called corpus.

- Stemming reduces vocabulary size while retaining the core meaning of words, simplifying feature extraction.
- This preprocessing step helps improve the performance and efficiency of the text-based model.

BUT, REVIEWS WILL BE IN TEXT FORM BUT COMPUTERS CAN'T UNDERSTAND TEXT FORM, SO WE S'D CONVERT THEM INTO NUMERICAL FEATURES FOR ANALYSIS.

C) TF-IDF (Term Frequency-Inverse Document Frequency)

- We used TF-IDF to convert the cleaned review text into numerical features for analysis.
- TF-IDF assigns higher weights to unique words within a document while down-weighting common words that appear across all reviews.
- The analysis used bigrams (two-word combinations) and considered the top 5000 most significant words in the corpus.
- After applying TF-IDF, the feature matrix had 5000 columns, as confirmed by the shape (n_samples, 5000).
- This step ensures that the model focuses on the most meaningful terms for sentiment classification.

D) Handling Imbalance

- The dataset had a class imbalance, with more positive reviews compared to neutral and negative reviews.
- To address this, we applied **SMOTE (Synthetic Minority Oversampling Technique)**, which generates synthetic data for the minority classes.
- SMOTE creates new samples by interpolating between existing data points in the minority class.
- Balancing the dataset ensures the model does not favor the majority class during training.

E) Train-Test Split

- The balanced dataset was split into training and testing sets using an 75:25 ratio with the train_test_split function.
- The training set is used to build and train the machine learning model, while the test set is used to evaluate its performance.
- The split ensures that the model is trained on a sufficient amount of data while retaining enough for accurate evaluation.
- A random state was used to ensure the results.

5.Model Building –

The plot_confusion_matrix function visualizes a confusion matrix as a heatmap, showing how well a model classifies each class. It allows for normalization (to display percentages instead of raw counts) and labels the axes with true and predicted class names. Each cell's value is displayed inside the heatmap, with text color adjusted for readability. The heatmap highlights correct predictions on the diagonal and errors off-diagonal, making it easier to interpret model performance. This visualization is crucial for evaluating classification accuracy and identifying misclassification patterns.

A) Model Selection Block

We'll the train this data with various ML models and we'll select the best peforming model by using cross validaton. Let's consider all the classification algorithm and perform the model selection process.

What is Cross-Validation (CV)?

- **Cross-validation** is a method to evaluate a machine learning model's performance by splitting the dataset into multiple parts (folds).
- In this case, **cv=10** means the dataset is divided into **10 folds**:
    - The model is trained on **9 folds** and tested on the **remaining 1 fold**.
    - This process is repeated **10 times**, ensuring every fold is used once as a test set.
- The model's accuracy is computed for each fold, and the **average accuracy** across all 10 folds is reported.

Algorithms Tested

logreg_cv = LogisticRegression(random_state=0) # Logistic Regression

dt_cv = DecisionTreeClassifier() # Decision Tree

knn_cv = KNeighborsClassifier() # K-Nearest Neighbors

svc_cv = SVC() # Support Vector Classifier

nb_cv = BernoulliNB() # Naive Bayes

- **Logistic Regression**: A linear model that predicts the probability of a class using a weighted combination of features.
- **Decision Tree**: A tree-based model that splits data into subsets based on feature thresholds.
- **KNN (K-Nearest Neighbors)**: A distance-based classifier that assigns the class of the majority of the k-nearest data points.
- **SVC (Support Vector Classifier)**: A model that separates classes by finding the optimal hyperplane in high-dimensional space.
- **Naive Bayes**: A probabilistic classifier based on Bayes' theorem and assumes independence among features.

Results from the Code

The output shows the average accuracy for each model across 10 folds:

Model	**Test Accuracy (%)**
Logistic Regression	**88.10**
Decision Tree	81.65
K-Nearest Neighbors (KNN)	87.51
Support Vector Classifier (SVC)	87.54
Naive Bayes	80.83

Insights:

Logistic Regression has the **highest accuracy** (88.10%), making it the best-performing model.

Logistic Regression is computationally efficient and performs well with text features like TF-IDF vectors, where high-dimensional sparse data is common.

Conclusion

From this block:

- **Logistic Regression** is selected as the best model due to its **highest average accuracy (88.10%)** across 10-fold cross-validation.
- Cross-validation is a method for model evaluation, ensuring that the selected model generalizes well to new data. Hyperparameter tuning will be applied next to further improve Logistic Regression's performance.

b) Hyperparameter Tuning in Logistic Regression –

We have got 94% accuracy with HPT of LR, But for classification problems we need to get confusion matrix and check f1 score rather than accuracy .

These 5 points say how accuracy increases by HPT.

Optimizes Regularization: Hyperparameter tuning adjusts the regularization strength (C) to find the right balance between overfitting (low regularization) and underfitting (high regularization), improving model generalization.

Feature Selection: By testing penalty types (L1 and L2), tuning ensures that irrelevant or less significant features are either reduced or eliminated, improving model focus and accuracy.

Improves Cross-Validation Performance: By evaluating multiple parameter combinations with cross-validation, the model is better tailored to perform well on unseen data.

Avoids Manual Guessing: Instead of manually selecting parameters, tuning exhaustively searches the parameter space, ensuring the best possible settings are chosen.

Handles Data Complexity: Different datasets have unique patterns; tuning adjusts hyperparameters to suit the specific data structure, leading to improved predictions and reduced errors.

Why Confusion Matrix and F1 Score Are Better Than Accuracy -

1. **Accuracy Can Be Misleading**
    - If a dataset has **90% positive reviews** and only **10% negative reviews**, a model can simply predict **all reviews as positive** and achieve **90% accuracy**. However, it completely fails to identify negative reviews, which is a big problem.
2. **Confusion Matrix Gives a Clearer Picture**
    - The **confusion matrix** shows the breakdown:
        - **True Positives (correct positives)**, **True Negatives (correct negatives)**, **False Positives (wrongly predicted positives)**, and **False Negatives (missed negatives)**.
    - For example: If a model predicts 95 positive reviews (90 correct + 5 wrong) and misses all 10 negative reviews, the confusion matrix clearly shows this failure, even if the accuracy is 90%.
3. **F1 Score Balances the Mistakes**
    - The **F1 score** looks at how well the model handles both:
        - **Precision** (How many of the predicted positives were actually correct?)
        - **Recall** (How many of the actual positives were captured by the model?)
    - For example: If the model predicts 95 positives but only 90 are correct, the precision is **90/95 = 94.7%**, and if it only captures 90 out of 100 actual positives, the recall is **90/100 = 90%**. The **F1 score combines these metrics** to give a balanced evaluation.

Example Summary:

Imagine you need to detect **spam emails** (10%) from regular emails (90%).
A model that predicts **all emails as non-spam** would get 90% accuracy but would completely fail at identifying spam. The confusion matrix and F1 score help you understand how well the model actually performs on **important cases like spam detection**.

Here are the formulas for the **four key parameters** derived from the confusion matrix:

1. **Accuracy:**
   - Formula:
$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$
   - **Interpretation:** Measures the overall correctness of the model.
2. **Precision (Positive Predictive Value):**
   - Formula:
$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
   - **Interpretation:** Indicates how many predicted positives (Yes) were actually positive.
3. **Recall (Sensitivity or True Positive Rate):**
   - Formula:
$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
   - **Interpretation:** Measures the model's ability to detect actual positives.
4. **F1 Score:**
   - Formula:
$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
   - **Interpretation:** Balances precision and recall, useful when dealing with imbalanced datasets.

c) Classification Metrics –

Here, we plotted confusion matrix as heatmap (a table with different colors for different metric) and ROC curve from the data.

Confusion Matrix

- The confusion matrix provides a **breakdown of model predictions** across the three classes: **Negative**, **Neutral**, and **Positive**.
- **Diagonal Elements**: Correct predictions for each class:
  - **Negative**: 2319
  - **Neutral**: 2196
  - **Positive**: 1850
- **Off-Diagonal Elements**: Misclassifications:
  - Example: 136 Positive reviews misclassified as Negative.

Classification report -

Class	Precision	Recall	F1-Score	Support
Negative	0.93	1.00	0.96	2326
Neutral	0.91	0.98	0.94	2232
Positive	0.99	0.84	0.91	2209

Key Insights

- Most predictions lie on the diagonal, indicating good performance.
- Misclassifications are relatively few, but the model struggles slightly with Positive reviews (136 misclassified as Negative and 223 as Neutral).

Macro Average F1-Score

$$\text{Macro F1-Score} = \frac{\sum \text{F1 (Class X)}}{N}$$

Where N = Total number of classes.

Weighted Average F1-Score

$$\text{Weighted F1-Score} = \frac{\sum(\text{F1 (Class X)} \times \text{Support (Class X)})}{\text{Total Support}}$$

Support

$$\text{Support (Class X)} = \text{Number of true samples belonging to Class X}$$

We got 94% accuracy.

D) ROC – AUC CURVE

This is a very important curve where we decide on which threshold to setup based upon the objective criteria.

Here we plotted ROC for different classes which can help us understand which class was classified better.

Also we plot micro and macro averages on the roc curve.

Insights:

1. **Class-Specific ROC Performance**:
    - **Classes 2 and 0** have been classified well, as their **area under the curve (AUC)** is high.
    - The optimal threshold for maximizing **True Positive Rate (TPR)** and minimizing **False Positive Rate (FPR)** can be chosen between **0.6 and 0.8**.
2. **Micro vs. Macro Average Performance**:
    - **Micro Average** performs well, indicating good overall classification performance across all classes.
    - **Macro Average**, however, shows a lower score, highlighting variations in the model's ability to classify different classes.
3. **Understanding Micro and Macro Averages**:
    - **Macro Average**: Computes the metric (e.g., precision, recall) for each class independently and then averages them, treating all classes equally.
    - **Micro Average**: means combining all the true positives, false positives, and false negatives from all classes into one big group and then calculating the metrics (like precision, recall, or F1-score) as if there is just one class. It treats every single sample the same, regardless of which class it belongs to.
    - In multi-class classification with potential **class imbalance**, the **micro-average** is generally preferred.

Summary of ROC and AUC –

Better explanation in video

https://www.youtube.com/watch?v=4jRBRDbJemM

1. **ROC Curve**:
    - The **Receiver Operating Characteristic (ROC)** curve is a graphical representation of a classifier's performance across various thresholds.
    - The Y-axis shows the **True Positive Rate (TPR)** (Sensitivity), and the X-axis shows the **False Positive Rate (FPR)** (1 - Specificity).
    - The goal is to maximize TPR while minimizing FPR, which corresponds to a curve close to the **top-left corner** of the graph.
2. **Thresholds**:
    - Changing the threshold for classification adjusts the trade-off between true positives and false positives.
    - The ROC curve summarizes all possible thresholds, providing a complete picture of the model's performance.
3. **AUC (Area Under the Curve)**:

- **AUC** measures the area under the ROC curve and summarizes the model's overall ability to distinguish between classes.
- Higher AUC values (0.8−1.0) indicate better model performance, while 0.50.5 implies random guessing.

4. **Applications**:
   - **Evaluate Model Performance**: ROC curves help identify the best thresholds for decision-making.
   - **Compare Models**: AUC is used to compare different models' performance.
   - **Optimize Trade-offs**: Useful in cases where false positives and false negatives have different costs (e.g., fraud detection or medical diagnosis).

Example: Email Spam Detection

- Suppose you're building a spam detection system for emails:
  - **Positive Class**: Spam emails.
  - **Negative Class**: Non-spam emails.
- The ROC curve evaluates how well the model distinguishes spam from non-spam for various thresholds.
- A threshold having (high TPR) and (lower FPR) can be is optimal threshold by using the ROC curve and the AUC score .

Challenges Faced -

During the course of this project, several challenges arose, spanning data preprocessing, feature engineering, model training, and evaluation. Here are the challenges faced, explained in detail:

---

1. Handling Class Imbalance

- **Challenge**:
  - The dataset exhibited a significant imbalance in the distribution of sentiments. Positive reviews (ratings 4 and 5) were overrepresented compared to negative (ratings 1 and 2) and neutral (rating 3) reviews.
  - This imbalance led to a bias in the model, which tended to predict the majority class (positive reviews) more often, reducing its ability to correctly classify minority classes.
- **Impact**:
  - This bias resulted in poor performance metrics like precision and recall for negative and neutral classes, which are equally critical for decision-making.

- **Solution**:
    - I implemented **SMOTE (Synthetic Minority Oversampling Technique)** to generate synthetic samples for the minority classes. SMOTE creates new data points by interpolating between existing samples, effectively balancing the dataset.
    - This improved the model's ability to generalize and predict across all classes.

---

2. Neutral Sentiment Ambiguity

- **Challenge**:
    - Reviews labeled as "neutral" (rating 3) often lacked clear emotional tone, making it difficult for both humans and models to interpret them accurately.
    - For example:
        - A review like "It's okay, but not great" is neutral but could lean slightly negative or positive based on the context.
- **Impact**:
    - The model struggled to distinguish neutral reviews from slightly positive or slightly negative ones, reducing its classification accuracy for this class.
- **Solution**:
    - To address this, I explored polarity scores from TextBlob to better capture the subtle nuances in neutral sentiment. While this added some improvement, neutral classification remained a complex challenge.

---

3. Noise in Text Data

- **Challenge**:
    - Text data in the reviewText column contained various forms of noise:
        - Typos or spelling errors, e.g., "amzaing" instead of "amazing."
        - Slang and abbreviations, e.g., "gr8" instead of "great."
        - Domain-specific terms or jargon related to musical instruments, which were difficult for generic NLP tools to handle.
- **Impact**:
    - This noise affected the accuracy of text-cleaning and feature-engineering steps like stemming, tokenization, and TF-IDF vectorization, as the model could not correctly interpret misspelled or domain-specific terms.
- **Solution**:
    - I addressed this partially by:

        - Using stemming to reduce word variations.
        - Manually inspecting common noisy words and correcting them.
        - Leveraging polarity scores from TextBlob for additional insights.
    - However, the inclusion of a spell-checking or domain-specific vocabulary model could further improve preprocessing.

---

4. Helpfulness Metric Correlation

- **Challenge**:
    - The assumption that "higher helpfulness indicates positive sentiment" was not always valid. For instance:
        - A negative review like "Terrible product, broke in two days, do not buy!" could still have a high helpfulness ratio if it warned others about a critical flaw.
- **Impact**:
    - This assumption led to misleading correlations between helpfulness scores and sentiment labels.
- **Solution**:
    - I decided to treat helpfulness as a separate feature rather than directly correlating it with sentiment. However, a more nuanced analysis of helpfulness patterns would be beneficial.

Improvements That Can Be Done -

The project demonstrated strong results, but there is always room for improvement. Below are actionable enhancements that could elevate the performance and scope of this analysis:

---

1. Incorporate Deep Learning Models

- Deep learning models, such as **BERT**, **RoBERTa**, or **DistilBERT**, could significantly improve sentiment classification.
- **Why**: These transformer-based models capture the context and semantic meaning of words better than traditional methods like TF-IDF.
- **How**: Fine-tune a pre-trained BERT model on the Amazon reviews dataset to classify sentiments while capturing deeper linguistic patterns.

---

2. Use Word Embeddings

- Replace TF-IDF with dense word embeddings like **Word2Vec**, **GloVe**, or **FastText**.
- **Why**: Word embeddings provide a continuous vector representation of words, capturing semantic relationships and reducing sparsity.
- **How**: Train embeddings on the dataset or use pre-trained embeddings for feature extraction.

---

3. Improve Neutral Sentiment Classification

- Develop a hybrid approach that combines:
    - Polarity-based rules for text with low sentiment intensity.
    - Contextual models like BERT to handle ambiguous phrases.
- **Why**: Neutral reviews often have subtle expressions that are hard for simple models to classify accurately.

---

5. Better Noise Handling

- Integrate a **spell checker** to correct common typos before stemming or tokenization.
- **Why**: Correcting noisy data would improve the quality of features extracted from text.

---

6. Add Sentiment Shift Analysis

- Analyze sentiment changes within long reviews (e.g., positive at the start but negative toward the end).
- **Why**: Sentiment shifts can provide more nuanced insights into customer experiences.

What You Have Learned -

1. **End-to-End Workflow Mastery**:
    - Managing a complete machine learning pipeline, from data preprocessing to model deployment.
2. **Importance of Data Preprocessing**:
    - Cleaning noisy text data and handling class imbalance are critical to achieving meaningful results.
3. **Feature Engineering Matters**:
    - Derived features like polarity, review length, and helpfulness significantly improved model performance.
4. **Model Selection and Tuning**:

- Simpler models like Logistic Regression, when tuned properly, can outperform complex models for certain datasets.

5. **Insights Beyond Accuracy**:
    - Visualizations and exploratory analysis helped uncover business-relevant insights like temporal trends and review helpfulness patterns.
