Approach:

1. **Data Collection:**
    - The data is collected from datasets, containing reviews and metadata for electronics products.
    - The 'Electronics_5.json' file is used for reviews, and 'meta_Electronics.json' file is used for metadata.
    - The data is stored in DataFrames using Pandas and saved as pickle files for later use.
2. **Preprocessing:**
    - The metadata is filtered to include only headphones-related data based on the product titles.
    - Preprocessing steps include handling missing values, removing duplicates, and converting ratings into categories.
    - Text data in reviews is preprocessed using techniques like lowercasing, removing HTML tags, accents, contractions, special characters, and lemmatization.
    - Word clouds are generated for good and bad reviews to visualize common words.
3. **Analysis:**
    - Descriptive statistics are calculated, including the number of reviews, average rating score, number of unique products, etc.
    - Top 20 most and least reviewed brands are identified.
    - Most positively reviewed headphone is identified based on average ratings.
    - Count of ratings over five consecutive years is reported.
    - Word clouds are generated to visualize frequent words in good and bad reviews.
    - Distribution of ratings is visualized using a pie chart.
    - Years with maximum reviews and highest number of customers are identified.
4. **Classification Model:**
    - The reviews are categorized as 'Good', 'Average', or 'Bad' based on their ratings.
    - The data is split into training and testing sets.
    - Word2Vec embeddings are used to represent textual data.
    - Several classification models (Logistic Regression, KNN, Decision Tree, Random Forest, Neural Network) are trained and evaluated using Word2Vec embeddings.
    - Classification reports are generated to evaluate model performance in terms of precision, recall, and F1-score for each class.

Methodologies:

- Data preprocessing techniques such as lowercasing, tokenization, lemmatization, and handling missing values are employed to clean the textual data.
- Word clouds are used to visually represent the most frequent words in good and bad reviews.
- Word2Vec embeddings are utilized to represent textual data as numerical vectors for classification.

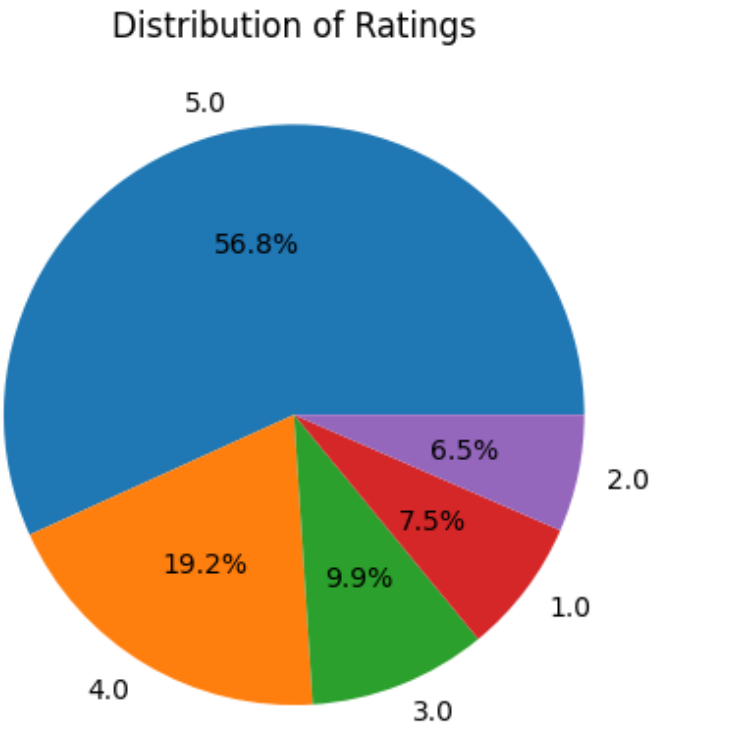
- Multiple classification models are trained and evaluated to predict the sentiment of reviews.

Assumptions:

- It's assumed that the ratings provided by users are reliable indicators of their sentiment towards the products.
- The preprocessing techniques applied to the textual data effectively clean and normalize the text for further analysis.
- Word2Vec embeddings capture semantic meanings of words and can effectively represent textual data for classification.

Results:

- Descriptive statistics provide insights into the overall distribution of ratings and the popularity of different brands.
- Word clouds visually represent common words in good and bad reviews, offering insights into customer sentiment.
- Classification models achieve varying levels of performance in predicting review sentiment, with Random Forest and Logistic Regression showing relatively higher accuracy and F1-scores.
- The approach provides a comprehensive analysis of the dataset, from exploratory data analysis to model building, to gain insights and predict review sentiments effectively.



Word Cloud:

Models and their Accuracy:

```
Model: Logistic Regression
/opt/conda/lib/python3.10/site-packages/sklearn/linear_mo
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the
    https://scikit-learn.org/stable/modules/preprocessing
Please also refer to the documentation for alternative so
    https://scikit-learn.org/stable/modules/linear_model.
  n_iter_i = _check_optimize_result(
              precision    recall  f1-score   support

     Average       0.36      0.05      0.09     10256
         Bad       0.66      0.48      0.56     14438
        Good       0.83      0.97      0.90     78093

    accuracy                           0.81    102787
   macro avg       0.62      0.50      0.52    102787
weighted avg       0.76      0.81      0.77    102787
```

```
Model: KNN
              precision    recall  f1-score   support

     Average       0.25      0.15      0.19     10256
         Bad       0.57      0.40      0.47     14438
        Good       0.84      0.93      0.88     78093

    accuracy                           0.78    102787
   macro avg       0.55      0.49      0.51    102787
weighted avg       0.74      0.78      0.76    102787

Model: Decision Tree
              precision    recall  f1-score   support

     Average       0.22      0.24      0.23     10256
         Bad       0.41      0.42      0.41     14438
        Good       0.85      0.83      0.84     78093

    accuracy                           0.71    102787
   macro avg       0.49      0.50      0.49    102787
weighted avg       0.72      0.71      0.72    102787
```

Model: Random Forest

	precision	recall	f1-score	support
Average	0.84	0.09	0.16	10256
Bad	0.76	0.39	0.51	14438
Good	0.82	0.99	0.90	78093
accuracy			0.81	102787
macro avg	0.81	0.49	0.52	102787
weighted avg	0.81	0.81	0.77	102787

Model: Neural Network

	precision	recall	f1-score	support
Average	0.39	0.04	0.08	10256
Bad	0.68	0.45	0.54	14438
Good	0.83	0.98	0.90	78093
accuracy			0.81	102787
macro avg	0.63	0.49	0.50	102787
weighted avg	0.76	0.81	0.76	102787

Collaborative Filtering:

Attempt 1: Chunking Method

The chunking method used in the code aims to handle large datasets by splitting them into smaller, more manageable chunks. This approach can significantly reduce the memory requirements associated with processing large volumes of data, making it a valuable technique for memory-efficient data processing.

Methodology:

The methodology employed in the code involves several key steps:

1) Data Preprocessing:
   - Handling Missing Values: The code fills missing values in the dataset with the value 'Unknown' to ensure that the data remains consistent and usable.
   - Removing Duplicates: Duplicates within the dataset are removed, reducing redundancy and ensuring data accuracy.
2) Chunking the Data:
   - Definition of Chunk Size: The original dataframe is divided into chunks, each containing a defined number of rows (in this case, 10000).
   - Iteration: The code iterates over each chunk, storing them as pickle files for subsequent processing.
3) Creating the User-Item Rating Matrix:

The code later loads these chunks and creates a user-item rating matrix for each chunk. This involves pivoting the data to have users as rows, items as columns, and ratings as values.

Challenges Faced:

The implementation of the chunking method may have presented some challenges, including:

• Memory Management: Processing and storing large datasets often pose memory management challenges. The chunking method offers a solution to overcome this, but it requires careful management to ensure the integrity of the dataset.
• Data Consistency: Splitting the dataset into chunks and processing them separately may introduce challenges related to data consistency, especially when merging these chunks. Ensuring that the final user-item rating matrix accurately represents the original dataset requires thorough validation and testing.
• Computational Efficiency: While the chunking method can enhance memory efficiency, computational efficiency must also be considered. Iterative operations over large datasets can be computationally intensive and may require optimization for speed and performance.
• Incorporating these key points into your report will provide an insightful overview of the methodology explored and the challenges encountered in implementing the chunking method for data processing.

Attempt 2: k-Fold Method

The k-Fold Method implemented in the provided code aims to conduct model evaluation using the k-fold cross-validation technique. This method is widely used to assess the performance of machine learning models, particularly in scenarios with limited data. Here are some notable points from the code that can be included in your report:

Methodology:

1) k-Fold Cross-Validation:

The code implements k-fold cross-validation with a specified number of folds (k). This technique systematically splits the dataset into k smaller subsets, called folds, and iteratively evaluates the model k times, using each fold once as the validation set and the remaining k-1 folds as the training set.

2) User-Item Rating Matrix:

The code creates a user-item rating matrix using the validation set for each fold. This matrix represents users' ratings for different items and serves as the basis for similarity calculations and predictive modeling.

3) Similarity Matrix and Predictive Modeling:

The code calculates the similarity matrix and employs nearest neighbor-based methods to predict ratings for items. It considers the ratings given by similar users to make predictions for a particular user-item pair.

Challenges Faced:

The implementation of the k-Fold Method may have presented some challenges, including:

• Data Sparsity: Limited ratings for certain users or items can lead to data sparsity issues, affecting the accuracy of similarity calculations and predicted ratings.

• Model Overfitting: Ensuring that the model generalizes well and does not overfit to the training data across different folds is crucial in k-fold cross-validation.
• Optimal Parameter Selection: The code iterates over different values of k (number of neighbors) to find the best model performance. Selecting the optimal value of k involves trade-offs between bias and variance and requires careful consideration.
• By incorporating these points into your report, you can provide a comprehensive overview of the methodology deployed and the challenges encountered in implementing the k-Fold Method for model evaluation and user-item rating predictions.