Τεχνικές Εξόρυξης Δεδομένων (2024-2025)
Εργασία: Ανάλυση Ηλεκτρονικού Εμπορίου (e-commerce) από προϊόντα της Amazon.

This project involves e-commerce analysis using the Amazon Product Dataset. The project is divided into two parts: data exploration & feature engineering, followed by machine learning tasks including clustering, classification, recommendation system, and sentiment analysis. The first should be completed by ***04/05/2025***, and the second part by ***01/06/2025***. From the dataset, you will select **only 5 product categories** for practical feasibility. Y**ou have to form groups, max 3 students per group for this project.**

Part 1 - Pre-processing

The dataset for this project is provided in JSON format, and your first task will be to extract the relevant data and save it in CSV format to facilitate analysis. The dataset you will be working with is available on Hugging Face. This dataset contains a collection of e-commerce product reviews, including product details, ratings, review texts, and other metadata. There are also pre-built scripts and tools for processing Amazon review data into CSV format. For this task you can use the *datasets* library from Hugging face to load and filter the dataset (*hint: use the load_dataset function with the parameter streaming=True*).

Task 1: Data Exploration and Feature Engineering

- **Dataset Preparation:**
    - Extract data for any five categories that you like (e.g., Electronics, Books, Home & Kitchen). Visit the **dataset link** and download only the JSON files for the categories that you plan to use. Parse the JSON files and create the csv file(s) that you are going to use for the rest of the Tasks (For the 5 product categories selected by each student, create 5 different CSV files.). The dataset is quite large, even for the 5 categories. Start with a smaller subset by limiting the numbers of rows downloaded per category, making

sure your code works, before getting more rows for the analysis part of this project. Clean the data by handling missing values, normalizing prices, and preprocessing text (more on the text preprocessing techniques in Part2).

- **Ratings and Reviews** (Visualize using Matplotlib, Seaborn, or Plotly. You can use histograms, box plots, scatter plots, bar charts, word clouds, etc)**:**
    - What is the distribution of product ratings within each of the 5 selected categories? Are there any categories with significantly higher or lower average ratings?
    - Identify products with a high number of reviews but low ratings. What are some common keywords or phrases in the reviews for these products?
    - For each of the 5 selected categories, identify the top 5 best-selling (highest review count) products. What are their key attributes (features)?
    - How have average product ratings evolved over time within each category? Create line plots to show the average rating "trend" for each category over months or years. Are there any noticeable patterns or seasonal variations?

Task 2: Feature Engineering with Sentiment Scores and Ratings

You can choose from three alternative methods for combining text sentiment, VADER, Hugging Face models, and review ratings to create a final sentiment score. These approaches will help create more nuanced features for machine learning tasks like sentiment classification, recommendation, or customer segmentation.

Alternative 1: Weighted Combination of Text Sentiment and Rating

In this approach, you will combine sentiment extracted from the review text using **VADER** or a **Hugging Face model** with the numerical **rating** provided by the user. This approach emphasizes blending both the subjective opinion from the text and the explicit satisfaction level indicated by the rating.

Steps:

1. **Sentiment Extraction from Review Text**:

    - **VADER Sentiment**: Use the VADER sentiment analyzer to derive a sentiment score from the review text. This score typically ranges from -1 (negative) to +1 (positive).

- **Hugging Face Sentiment Model**: Alternatively, you can use a pre-trained sentiment model from Hugging Face (e.g., **DistilBERT**, **RoBERTa**) for a more context-sensitive sentiment classification. The model will classify the review as **positive**, **negative**, or **neutral**.

2. **Normalize the Rating**:
    - Convert the numerical **rating** (1 to 5 stars) into a normalized scale from 0 to 1:

$$\text{Normalized Rating} = \frac{\text{Rating} - 1}{4}$$

3. This makes the rating comparable with the sentiment scores.

4. **Calculate the Final Sentiment Score**:
    - Combine the sentiment score from the text with the normalized rating using a weighted average:

$$\text{Final Sentiment Score} = w_1 \times \text{Text Sentiment} + w_2 \times \text{Normalized Rating}$$

5. where w1 and w2 are weights that reflect the importance of text sentiment and rating, respectively. You can experiment with different weight values.

Alternative 2: Rating-Adjusted Sentiment

This approach involves adjusting the sentiment score based on the rating to ensure that the numerical rating reflects the intensity of sentiment more strongly.

1. **Sentiment Extraction from Review Text**:
    - Use **VADER** or **Hugging Face Sentiment Models** to extract sentiment scores from the review text, which will be later classified as positive, neutral, or negative.

2. **Adjust Sentiment Based on Rating**:
    - For reviews with higher ratings (e.g., 4 or 5 stars), amplify the sentiment score to reflect stronger positive sentiment. For lower ratings (e.g., 1 or 2 stars), adjust the sentiment score downward, making it more negative, even if the text is neutral or mildly positive.
    - **Adjusting Method**:
        - For ratings of **4 or 5**, increase the sentiment score by a factor (e.g., adding 0.2).
        - For ratings of **1 or 2**, decrease the sentiment score by a factor (e.g., subtracting 0.2).
3. **Final Sentiment Score**:
    - Take the *adjusted sentiment score* and produce the final sentiment label.

Tools and Libraries:

- **VADER**: nltk.sentiment.vader for sentiment analysis.
- **Hugging Face**: Pre-trained models from the transformers library such as **DistilBERT**, **RoBERTa**, or **BERT** for advanced sentiment classification.
- **Python Libraries**: pandas, numpy, and scikit-learn for data manipulation and machine learning model training.

Task 3: Feature Engineering with Price Metrics (Optional)

1. **Price-per-Feature Metrics**:
    - **Objective**: Create features that allow price comparison across products by normalizing price values against their unique attributes.
    - **Methods**: For products with various features (e.g., size, color, material), calculate the price per unit of feature. For example, a product like a "leather sofa" may be priced differently depending on its size (2-seater vs. 5-seater), and this can be normalized by dividing the price by the size to compare value for money.
    - **Tools**: Simple arithmetic operations or feature extraction tools like pandas can be used to compute these metrics.
2. **Normalized Ratings**:
    - **Objective**: Address the potential bias of high-rating counts by creating a normalized score that adjusts ratings based on the volume of reviews.

- **Methods**: Calculate the normalized rating by adjusting the product's average rating with respect to the number of reviews it has. A product with a high rating but few reviews may be more volatile, whereas one with many reviews will provide more stable, reliable data. Calculate a weighted rating using the formula:

$$WeightedRating = AverageRating * log(ReviewCount + 1)$$

By creating these additional features, the dataset becomes more meaningful and allows subsequent machine learning models to learn better patterns from the data. **Feature importance** can later be assessed using techniques like Random Forest or Gradient Boosting Machines to determine the most relevant features for a particular model.

Part 2 - Learning Tasks

Objective:
Apply machine learning techniques for clustering, recommendation systems, and sentiment analysis.

Task 1: Clustering for Product Grouping

Group similar products within categories based on features like price, description, and ratings.

1. **Preprocessing :**
   - Clean and prepare the text data for model training by standardizing the input (Convert text to lowercase, remove punctuation, perform stemming…) . **Tools**: you can use nltk or spaCy for preprocessing tasks.
   - Scale numerical features (price, ratings) to prevent them from dominating the clustering.
2. **Vectorization**

- Use **TF-IDF** for vectorization of product descriptions (don’t forget to experiment with the max_features parameter to limit the vocabulary size).
- Combine numerical features and TF-IDF vectors into a single feature matrix.

3. **Clustering**: Perform clustering using **one clustering** method (e.g., K-Means with elbow method, DBSCAN). Visualize clusters using dimensionality reduction techniques (PCA, t-SNE).

Evaluation Metric:
Silhouette Score: Measures how similar items within a cluster are compared to items in other clusters. A higher score indicates better-defined clusters. Analyze the characteristics of the products within each cluster to gain insights. If you are using K-means, use the elbow method to help determine the optimal number of clusters.

Task2: Recommendation System

A recommendation system suggests products to users based on past behavior and preferences. This task involves two major techniques: **Collaborative Filtering** and **Content-Based Filtering**. You can use products from only one category for the recommendation system. *Optional - Use products from multiple categories for the recommendation system.*

Data Preparation:

- For CF, the data needs to be in a suitable format (user-item interactions, ratings). You will need the following fields from the dataset: user_id, asin, rating
- For CBF, the preprocessed product descriptions and other relevant attributes are needed (asin, text, price, category, title). You should preprocess the text, and title. You should also normalize the price. Finally, one hot encoding, or another method to convert the category into numerical features, if products from multiple categories are being used.

Collaborative Filtering using scikit-learn, implement a user-based and item-based collaborative filtering (CF) system that generates personalized book recommendations for a given user. Based on the user's past purchases and ratings, the CF system should recommend the products with their predicted scores. Evaluate the recommendations by showing the top-K recommended products and their predicted ratings.

- **User-Based Collaborative Filtering**:
    - Create a user-item matrix from the training data, filling missing values with 0.

    - Calculate the cosine similarity between users using sklearn.metrics.pairwise.cosine_similarity
    - For each user-item pair in the test set:
        - Find the K most similar users from the training set.
        - Calculate a predicted rating based on the weighted average of ratings from the similar users.
- **Item-Based Collaborative Filtering**
    - Create a user-item matrix from the training data, filling missing values with 0.
    - Transpose the matrix, and then calculate the cosine similarity between items using cosine similarity
    - For each user-item pair in the test set:
        - Find the K most similar items from the training set (Try starting with K=5).
        - Calculate a predicted rating based on the weighted average of ratings from the similar items.

Content-Based Filtering:

- Convert product descriptions into **Word2Vec** vectors (it is better to download and use pre-trained embeddings).
- Calculate cosine similarity between product vectors.
- Recommend top-K products with high similarity scores.
- Optional: Evaluate the recommendations using Recall at K (which measures how many of the relevant items appear in the top-K recommended products).

Hybrid Approach: Weighted Average:

- Combine the recommendations from CF and CBF (For this approach, combine the results from collaborative and content-based filtering, using the asin field to connect the data)
- Method: Assign weights to the CF and CBF scores and combine them.

  Hybrid Score = (CF Score * CF Weight) + (CBF Score * CBF Weight)

  Example Weights: CF weight: 0.7, CBF weight: 0.3

- Evaluate the performance of the hybrid system. (experimentation is crucial to find the optimal weights for the hybrid system).

Evaluation: Analyze the strengths and weaknesses of each approach.

Task3: Classification task - sentiment analysis

Sentiment analysis aims to understand the emotional tone of **customer reviews**, **classifying them** as **positive**, **negative**, or **neutral**.

1. **Preprocessing :** Preprocessing steps will now be performed on the reviews.
2. **Feature Extraction (vectorization)**
    1. Create **TF-IDF** features like you did for the descriptions in the previous task.
    2. Use **Word2Vec,** or **FastText** for feature extraction on the reviews (it is better to use pre-trained embeddings).
    3. If you have completed Task 3 you can also append the numerical features you created to the word vectors and experiment to see if they improve performance on the sentiment analysis task. Try the sentiment analysis task without and with the numerical features.
3. **Best model**
    - **Objective**: Train different classifiers to predict sentiment labels from the features extracted from review texts.
    - **Approach**: Use the following classification models:
        - **Naive Bayes**
        - **KNN**
        - **Random forests**
        - **Optional: Deep Learning Models**: More advanced models like **LSTM** or **BERT** can be used for more accurate sentiment predictions, especially when working with large datasets. Use the available functions from Hugging face library and perform the sentiment classification task with deep learning models.
    - **Metrics for Evaluation**: **F1-Score**, which balances precision and recall. Create a table to showcase **results for all models and all feature sets**. (this is very important because it showcases the performance of your methods and gives you a way to compare different approaches) . All models will be

trained exclusively on the training dataset. Model performance will be evaluated on the held-out test dataset.

4. **10-Fold Cross-Validation:** This will provide a more robust estimate of model generalization (prevent overfitting). Evaluate and record the performance of each model using 10-fold cross-validation on the **training data**.

Evaluation Metrics:

- Calculate and record the following metrics:
  - Precision (Macro-average)
  - Recall (Macro-average)
  - F1-Score (Macro-average)
  - Accuracy

Results Presentation:

- Create a clear and well-organized table to showcase the results for all models and feature sets. This table should include the evaluation metrics from the 10-fold cross-validation on the train set.

Feature Set	Model	Precision	Recall	F1-Score	Accuracy
TF-IDF	KNN				
TF-IDF	Naive Bayes				
TF-IDF	Random Forest				
Embeddings	KNN				
Embeddings	Naive Bayes				
Embeddings	Random Forest				

- You then perform **a single, final evaluation on the test set**. For the test set, you should create a separate table (or section in your report) that shows the performance metrics (precision, recall, F1-score, accuracy) of each trained model.

Model	Precision	Recall	F1-Score	Accuracy
KNN				
Naive Bayes				
Random Forest				

Task 4: Frequent Pattern Mining (Bonuns)

Frequent pattern mining focuses on discovering patterns or associations in transactional data, where products or items are bought together frequently.

1. **Data Representation**
    - **Objective**: Represent product purchases as transactions, where each transaction consists of products bought together by a customer.
    - **Approach**: Using the **asin** (Amazon Standard Identification Number) and **user_id**, group transactions where users buy multiple products in one review or over time (Pandas' *groupby* and *merge* functions can be used to achieve this. Also Pandas' datetime functionality can be used to extract time-based features (month, day of week, etc.).).
    - **Example**: If user 1 buys a "phone case" and "screen protector" together, this pair would form a *transaction*.
2. **The Apriori Algorithm**
    - **Objective**: Apply the **Apriori algorithm** to identify **frequent itemsets** (sets of products that are often bought together).
    - **Approach**: The algorithm operates by generating candidate itemsets and pruning those that do not meet a minimum support threshold. These frequent itemsets can then be used to generate **association rules**, which describe the likelihood that products are purchased together (e.g., "If a user buys Product A, they are likely to buy Product B").
    - **Tools**: Python libraries like **mlxtend** provide simple implementations of the Apriori algorithm.
3. **Temporal Context**
    - **Objective**: Discover patterns that are specific to certain time intervals, such as seasonality or holiday promotions.

- **Approach**: Use the **timestamp** data to segment the dataset by time periods (e.g., months or seasons) and apply the Apriori algorithm to find patterns that appear frequently during specific times of the year.
- **Evaluation Metric**: *Lift*, which measures the strength of association rules, is used to evaluate whether products frequently bought together have a higher probability than random chance.