CODE IQ: IDEATION PHASE REPORT SUBMISSION

1. Team Name: The Qubits

2. Team Members

1. **Contestant 1 (Leader)**: Akhilesh T S

3. GitHub Repository

https://github.com/akhilesh1709/Code-IQ-Product-attribute-extraction

The github link also contains the output test results inside the folders "json_output_results"

The output results are also uploaded on the google drive link:
https://drive.google.com/drive/folders/1CgNtG0gB8lmzz2p-Srl7CgpHlmxgqsjr?usp=sharing

4. Track Chosen

- **Track Number**: 2
- **Track Name**: "AI for Product Information Extraction"

5. Idea and Approach Details

Solution Overview:

The solution aims to automate the extraction of product attributes such as brand, price, color, material, size, weight, features, and specifications from product descriptions in CSV format. This process will streamline the structuring of product information for e-commerce platforms, improving the efficiency of cataloging and enhancing the user experience.

The product descriptions often contain unstructured text, and our system uses Natural Language Processing (NLP) techniques to parse these descriptions and extract relevant product information. The extracted data is then structured in a JSON format, making it easy to integrate into a product catalog.

Technical Stack:

- **Programming Language**: Python 3.x
- **Libraries and Frameworks**:
    - **pandas**: Used for reading CSV files and handling data manipulation.
    - **re**: Used for regular expressions to match and extract relevant patterns from the text.
    - **spaCy**: (Optionally) for Named Entity Recognition (NER) if required for more advanced text processing.
    - **tqdm**: For displaying a progress bar while processing descriptions.
- **File Format**: JSON for structured output.

6. Attribute Extraction Logic

The logic for extracting product attributes from descriptions is based on predefined patterns that match common forms of these attributes in text. Below is a detailed explanation of how each attribute is extracted:

6.1. Product Name

- **Pattern Used**: `r"product name: (\w[\w\s]+)"`
- **Logic**:
    - The pattern attempts to match the product name by looking for phrases that are preceded by the text "product name:".
    - This simple pattern is used to capture the first part of the description where the product name typically appears.
    - **Challenges**: Variations in how product names are presented may require adjustments.

6.2. Color

- **Pattern Used**:
```
r"\b(black|white|red|blue|green|yellow|purple|pink|gray
|silver|gold|brown|orange|navy|maroon|turquoise|beige)\
b"
```
- **Logic**:
    - A list of common colors is used to create a word boundary regex.
    - The pattern captures any of the defined colors found in the description.
    - **Challenges**: This pattern only works for predefined colors. If additional colors are required, the pattern will need to be extended.

6.3. Material

- **Pattern Used**: `r"\b(cotton|polyester|leather|silk|wool|nylon|metal|plastic|glass|ceramic|wood|rubber|aluminum|stainless steel|faux leather|mesh)\b"`
- **Logic**:
  - Similar to color extraction, this pattern checks for specific materials (cotton, leather, metal, etc.).
  - The pattern uses word boundaries (`\b`) to match materials that are whole words and are surrounded by spaces or punctuation.
  - **Challenges**: The list of materials is static. If other materials are present, manual updates to the regex are necessary.

6.4. Brand

- **Pattern Used**: `r"\b(apple|samsung|lg|sony|dell|hp|nike|adidas|lenovo|canon|fujifilm|bose|jbl)\b"`
- **Logic**:
  - A predefined list of brand names is searched for in the description.
  - The regex checks for a match with well-known brand names, which are typically used in product descriptions.
  - **Challenges**: The brand list is static, and new brands may need to be added as the dataset grows.

6.5. Category

- **Pattern Used**: `r"\b(smartphone|laptop|camera|headphones|shoes|clothing|furniture|kitchen|electronics|home appliance|watch)\b"`
- **Logic**:
  - Categories are defined based on product types (e.g., smartphone, laptop, shoes).
  - This pattern allows the identification of the product's category by matching it with common product categories.

    - **Challenges**: Like other categories, additional categories may need to be added depending on the types of products being processed.

6.6. Size & Dimensions

- **Pattern Used**: `r'(\d+(?:\.\d+)?)\s*(?:x|×)\s*(\d+(?:\.\d+)?)\s*(?:x|×)\s*(\d+(?:\.\d+)?)\s*(cm|inches|m)'`
- **Logic**:
    - This pattern looks for size dimensions presented in a typical format such as "10 x 5 x 2 cm" or "12×6×3 inches".
    - It captures the length, width, height, and unit (e.g., cm, inches).
    - **Challenges**: The size format may vary (e.g., space-separated instead of using "×"). The pattern assumes the common `x` or `×` as separators.

6.7. Weight

- **Pattern Used**: `r"(\d+(?:\.\d+)?)\s*(kg|g|grams|lbs|pounds)"`
- **Logic**:
    - This pattern captures weights with associated units (kg, g, lbs, etc.).
    - It handles both integer and floating-point numbers and supports different weight units.
    - **Challenges**: Units may appear in different formats (e.g., "g" vs. "grams"), so handling variations is crucial.

6.8. Price

- **Pattern Used**: `r"(₹|rs\.?|\$)?\s*(\d+(?:,\d+)*(?:\.\d+)?)"`
- **Logic**:
    - The price extraction pattern handles various currency symbols like ₹ (Indian Rupee), $ (Dollar), and rs (Indian Rupee abbreviated).
    - It captures the price value and can handle thousands separators and decimals.
    - **Challenges**: Currency formatting can vary; for example, the pattern currently handles ₹ and $ but may need extension for other currencies.

6.9. Rating & Reviews

- **Pattern Used**: `r"(\d\.\d)\s*\/\s*5"` and `r"(\d+)\s*reviews?"`
- **Logic**:
  - **Rating**: The pattern matches ratings on a scale of 1 to 5 (e.g., "4.5 / 5").
  - **Reviews**: This pattern matches the number of reviews, such as "150 reviews".
  - **Challenges**: Ratings and reviews may not always appear in this exact format. Variations such as "stars" instead of "/ 5" might require adjustments.

6.10. Warranty

- **Pattern Used**: `r"(\d{1,2})\s*(year|month)s?\s*warranty"`
- **Logic**:
  - This pattern is designed to match warranty information, capturing both the number of years or months (e.g., "2 years warranty").
  - **Challenges**: There may be variations in how the warranty period is mentioned, so extra regex may be needed.

6.11. Connectivity

- **Pattern Used**: `r"(wifi|bluetooth|4g|5g|nfc|usb)"`
- **Logic**:
  - The pattern looks for connectivity technologies such as Wi-Fi, Bluetooth, 4G, 5G, NFC, and USB.
  - **Challenges**: This extraction logic could be expanded to capture other technologies like Zigbee or NFC based on product types.

7. Decision Rationale:

The choice of Python and regular expressions (regex) was influenced by the need for a flexible, lightweight, and fast approach to extract product attributes from text. Python libraries like re and pandas are efficient for text processing and data manipulation, and JSON is a standard format for data exchange, making the output versatile for integration with e-commerce platforms.

We chose not to rely on pre-built NLP models for entity extraction (like spaCy's NER) because the product descriptions were relatively structured, and regex patterns could efficiently handle most cases. In the future, spaCy could be used for further enhancement if necessary.

8. Innovation Highlights:

The solution offers an innovative approach by automating the attribute extraction process, reducing the manual effort needed to process product descriptions. The integration of regex for detailed and flexible extraction, along with the ability to customize patterns for various product types, makes it stand out. Additionally, features like handling missing values, multiple attribute types (e.g., weight, price), and flexibility in the features list add to its uniqueness.

9. Methodology/Architecture Diagram

FlowChart / Process Flow:

1. **Data Input**: Input CSV files containing product descriptions.
2. **Preprocessing**: Text normalization, including removing extra spaces, converting to lowercase.
3. **Attribute Extraction**: Use regex patterns to extract attributes such as brand, color, price, size, etc.
4. **Data Structuring**: Organize extracted data into a structured format (JSON).
5. **Output**: Store and output the extracted data in JSON format for easy integration into a catalog system.