# System Analysis & Design: Product Review Sentiment Analysis

### Project: Deployed Product Review Sentiment Analysis Application

* **Document Version:** 1.0
* **Date:** October 10, 2025
* **Author:** Abdelrahman

### 1. Introduction

#### 1.1. Problem Statement

In the digital marketplace, customer reviews are a critical source of business intelligence. However, the sheer volume of unstructured text data makes manual analysis inefficient, expensive, and unable to scale with business growth. Businesses require an automated solution to process and interpret customer feedback in near real-time to make informed decisions about product development, marketing, and customer service.

#### 1.2. Project Objectives

The objective of this project is to design, build, and deploy an end-to-end system that automates the sentiment analysis of product reviews. The key goals are to:

* **Develop** a web scraping component to automatically collect product review data.
* **Implement** a robust data processing pipeline to clean, preprocess, and prepare text for model ingestion.
* **Train** and fine-tune a deep learning model (BERT-based) to accurately classify the sentiment of review text as positive, neutral, or negative.
* **Build** an intuitive and interactive web application using Streamlit that allows a user to input text and receive an instant sentiment prediction.
* **Deploy** the final application to a cloud platform for accessible, public use.

### 2. System Architecture

This system is designed using a **Component-Based Pipeline Architecture**. This modular approach decouples the system's core functionalities—data collection, processing, modeling, and presentation—into independent, interchangeable components. This design enhances parallel development, simplifies unit testing, and improves overall system maintainability.

High-Level Architectural Flow:

Web Source → [1. Scraper] → Raw Data (CSV) → [2. Data Pipeline] → Cleaned & Tokenized Data → [3. Model Training] → Saved Model Artifact → [4. Streamlit Application] → End User

### 3. Data Flow & System Behavior

#### 3.1. Data Flow Diagram (DFD)

The DFD illustrates the path of data from its external source through various processes and storage components until it is presented to the end-user.

* **External Entities:** Web Source (e.g., E-commerce Site), End User
* **Processes:** 1. Scrape Data, 2. Clean & Process Text, 3. Train Model, 4. Analyze Sentiment
* **Data Stores:** Raw Reviews (File), Cleaned Reviews (File), Trained Model (File)

**DFD Process Breakdown:**

1. The **Scrape Data** process initiates a request to the **Web Source**, extracts review text, and stores it in the **Raw Reviews** data store.
2. The **Clean & Process Text** process reads data from **Raw Reviews**, performs all NLP preprocessing, and saves the structured output to the **Cleaned Reviews** data store.
3. The **Train Model** process loads data from **Cleaned Reviews** to generate and save the final **Trained Model** artifact.
4. The **End User** submits text through the Streamlit App. The app loads the **Trained Model** to **Analyze Sentiment** and immediately displays the classification result back to the user.

#### 3.2. Sequence Diagram (Live Application User Interaction)

This diagram details the chronological sequence of interactions when an end-user analyzes a review using the deployed Streamlit application.

1. **User** enters text into the input field on the Streamlit UI.
2. **User** clicks the "Analyze" button, triggering an event.
3. The **Streamlit UI** calls its internal predict() function, passing the input text as an argument.
4. The predict() function loads the pre-trained **BERT Model** from the saved artifact.
5. The predict() function invokes the **Data Pipeline** to preprocess the raw input text.
6. The **Data Pipeline** returns the cleaned and tokenized text in the format required by the model.
7. The predict() function passes the processed text to the **BERT Model** for inference.
8. The **BERT Model** returns a sentiment prediction (e.g., 'Positive') and a corresponding confidence score.
9. The **Streamlit UI** dynamically renders the prediction and score, presenting the final result to the **User**.

### 4. Component Design

The system is broken down into four primary components, each with a distinct responsibility and technology stack.

| **Component** | **Responsible Role** | **Key Technologies** | **Description** |
| --- | --- | --- | --- |
| **1. Data Collection** | Data Engineer | requests, BeautifulSoup4 | A Python script responsible for programmatically scraping review data from target websites and saving it into a structured format (e.g., CSV). |
| **2. Data Pipeline** | Data Scientist | pandas, NLTK / spaCy | A Python module containing functions for all text preprocessing: lowercasing, removing punctuation/stopwords, and tokenization for the BERT model. |
| **3. Model Training** | ML Engineer | PyTorch/TensorFlow, Hugging Face | A script that loads the cleaned data, fine-tunes a pre-trained transformer model (e.g., DistilBERT), evaluates its performance, and saves the final model artifact. |
| **4. Streamlit App** | App Developer | Streamlit | The main app.py script that builds the UI, loads the saved model, and orchestrates calls to the data pipeline to serve predictions to the end-user. |

### 5. UI/UX Wireframe

The user interface will be minimalist and function-oriented to ensure ease of use.

* **Title:** Product Review Sentiment Analyzer
* **Subtitle:** Enter a product review below to analyze its sentiment.
* **Component 1: Input Field:** A large text area (st.text\_area) allowing for multi-line user input.
* **Component 2: Action Button:** A primary button (st.button) labeled "Analyze Sentiment".
* **Component 3: Output Area:** A conditional display area that appears upon analysis. It will use colored banners to indicate the result (st.success for Positive, st.warning for Neutral, st.error for Negative) and clearly state the confidence score.

### 6. Deployment Strategy

The application will be deployed and hosted on **Streamlit Community Cloud**, which offers seamless integration with GitHub for continuous deployment.

**Deployment Process:**

1. Initialize a public GitHub repository for the project.
2. The repository must contain the following essential files:
   * app.py: The main Streamlit application script.
   * requirements.txt: A list of all Python library dependencies.
   * The serialized model artifact (or a script to download it from cloud storage like S3 or Hugging Face Hub if it is too large for GitHub LFS).
3. Connect the GitHub repository to the Streamlit Cloud dashboard.
4. Streamlit Cloud will automatically build the environment from requirements.txt and deploy the application from the specified branch (e.g., main).
5. A public URL will be generated, making the application live and accessible.

### 7. Technology Stack

* **Programming Language:** Python 3.9+
* **Data Science & NLP:** Pandas, NLTK, Scikit-learn, spaCy
* **Web Scraping:** Selenium
* **Machine Learning:** PyTorch / TensorFlow
* **Web Application Framework:** Streamlit
* **Version Control:** Git & GitHub
* **Development Environment:** VS Code, Jupyter Notebooks