00

InsightRush.

Comprehensive Product Review Analysis and Summarization Platform

Artificial Intelligence Lab (AZ5411)
Mini Project

Done by

Rameez Akther M -2022510025
Pragadeshwaran S -2022510005
Arun Kumar S -2022510039
Ayub Khan -2022510034

B.Tech AI&DS
IT Department
MIT Campus
Anna University

Table of Content

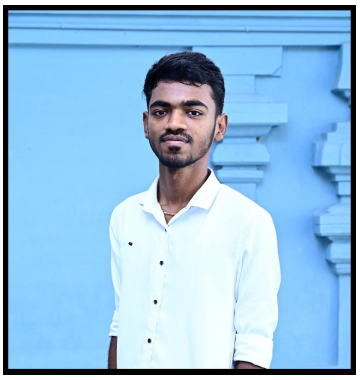
About Team

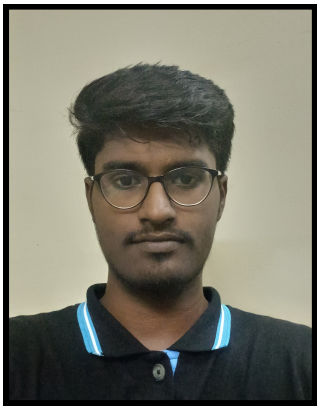
1. Rameez Akther - Team Lead

Rameez Akther leads the InsightRush team, focusing on backend integration and web scraping. With a solid background in Python and Flask, he ensures the seamless operation of the application's backend. Rameez's strategic vision and technical expertise have been instrumental in the project's success.

2. Pragadeshwaran - Model Development Specialist

Pragadeshwaran specializes in machine learning and natural language processing. He has developed the sentiment classification models using CNN, LSTM, and utilized pre-trained BERT. Pragadeshwaran's attention to detail and expertise in model training have significantly improved the accuracy of InsightRush's sentiment analysis.





3. Arun Kumar - Text Summarization Expert

Arun Kumar is responsible for the text summarization feature. His knowledge of NLP and implementation of the BartForConditionalGeneration model provide users with concise summaries of product reviews. Arun Kumar's work enhances user understanding and decision-making.

4. Ayub Khan - UI/UX Designer

Ayub Khan is the creative force behind InsightRush's user interface. His designs prioritize usability and accessibility, ensuring a seamless user experience. Ayub Khan's attention to detail and design aesthetics contribute to InsightRush's functionality and appeal.

Abstract

InsightRush is a web application developed using Python Flask, HTML, CSS, and JavaScript. The app enables users to input an Amazon or Flipkart product link to retrieve comprehensive product information, including reviews, price history, and detailed product data. The backend processes include scraping product reviews and performing sentiment classification on these reviews using CNN, LSTM, and BERT pretrained transformers. Reviews are classified into positive and negative sentiments. Subsequently, these classified reviews are summarized using the BartForConditionalGeneration model. The summarized reviews and other product details are then presented to the user. Additionally, all data is stored in a MySQL database, and users can access previous responses through a dedicated history tab.

Objectives

- Provide users with comprehensive and insightful analysis of Amazon and Flipkart products.
- Allow users to input a product link to retrieve detailed product information, including price history and reviews.
- Perform sentiment analysis on product reviews, classifying them as positive or negative using CNN, LSTM, and BERT models.
- Summarize classified reviews using the BartForConditionalGeneration model to present concise and relevant information.
- Help users make informed purchasing decisions based on synthesized summaries of product reviews.
- Securely store all data in a MySQL database.
- Offer a history tab for users to access previous analyses.

System Requirements

Hardware Requirements

- **Processor:** Intel i5 or equivalent
- **RAM:** Minimum 8 GB
- **Storage:** Minimum 10 GB of free disk space
- **Internet Connection:** Required for scraping product reviews and accessing external APIs

Software Requirements

- **Operating System:** Windows 10 or higher, macOS 10.15 or higher, Linux (any recent distribution)

- **Python:** Version 3.7 or higher
- **Flask:** Version 2.0 or higher
- **MySQL:** Version 8.0 or higher

Python Packages

- **Web Application Framework:**
  - **Flask:** For building the web application.

- **Scraping and Parsing:**
  - **requests:** For making HTTP requests to scrape product data.
  - **BeautifulSoup4:** For parsing HTML content.
  - **selenium:** For automated web scraping.

- **Database Interaction:**
  - **SQLAlchemy:** For ORM with SQL databases.
  - **mysql-connector-python:** For connecting to MySQL databases.

- **Data Manipulation and Analysis:**
  - **pandas:** For data manipulation and analysis.
  - **numpy:** For numerical operations.

- **Natural Language Processing:**
  - **nltk:** For various NLP tasks such as tokenization, stopwords removal, and frequency distribution.
  - **transformers:** For using BERT and BartForConditionalGeneration models.
  - **tensorflow and keras:** For deep learning models including CNN, LSTM, and BERT.

- **Visualization:**
  - **wordcloud:** For generating word clouds.
  - **matplotlib:** For creating plots and visualizations.

- **Utilities:**
  - **google-generativeai:** For integrating Google Generative AI.
  - **asyncio:** For asynchronous programming.
  - **threading:** For multithreading support.
  - **secrets:** For generating secure random numbers.
  - **re:** For regular expressions.

Browser Requirements

- **Web Browsers:** Latest versions of Google Chrome, Mozilla Firefox, Safari, or Microsoft Edge

Additional Requirements

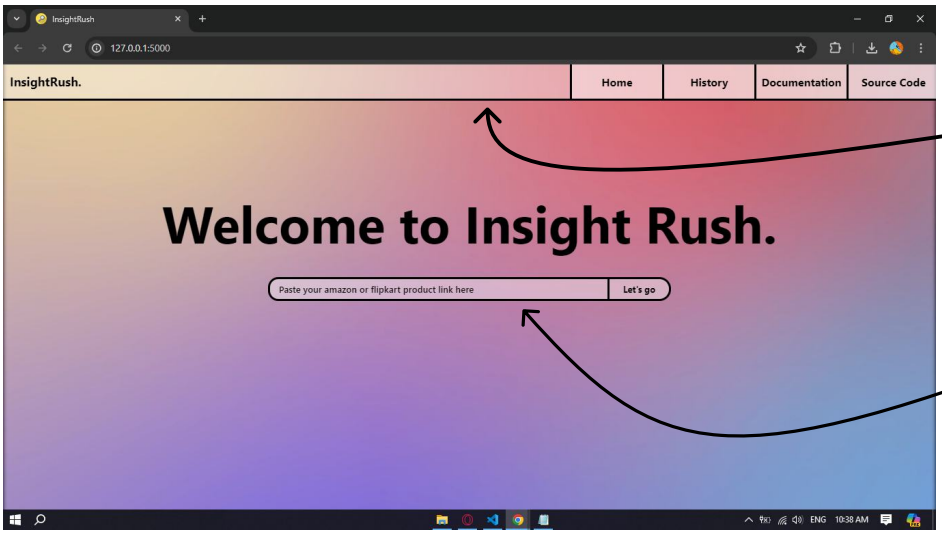
- **API Keys:** For any external APIs used for scraping or sentiment analysis (if applicable)
- **Database Configuration:** Access to a MySQL server for storing and retrieving data

These requirements ensure that the application runs smoothly and efficiently, providing users with a seamless experience.

Usage

Home Page

The home page serves as the entry point to the application. It features a user-friendly interface with a prominent link box where users can input Amazon or Flipkart product links. Additionally, there is a taskbar that facilitates easy navigation to different pages within the application. The taskbar includes buttons for "Home," "History," "Documentation," and "Source Code," providing users with quick access to essential features and information.



Home page button to take the user ot the home page

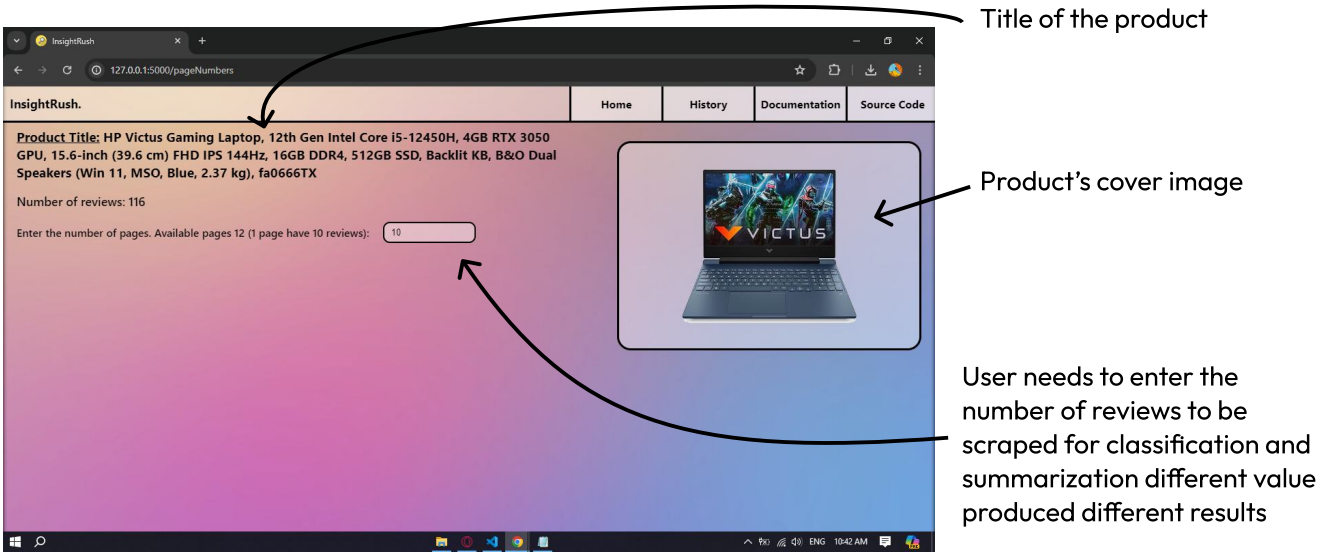
History page button to take the user to history page where the user get's access to already retrieved product data

Documentation page button which takes the user to an external drive folder which contains this document.

Source Code page button which takes the user to an external github repository where the project's source code is available.

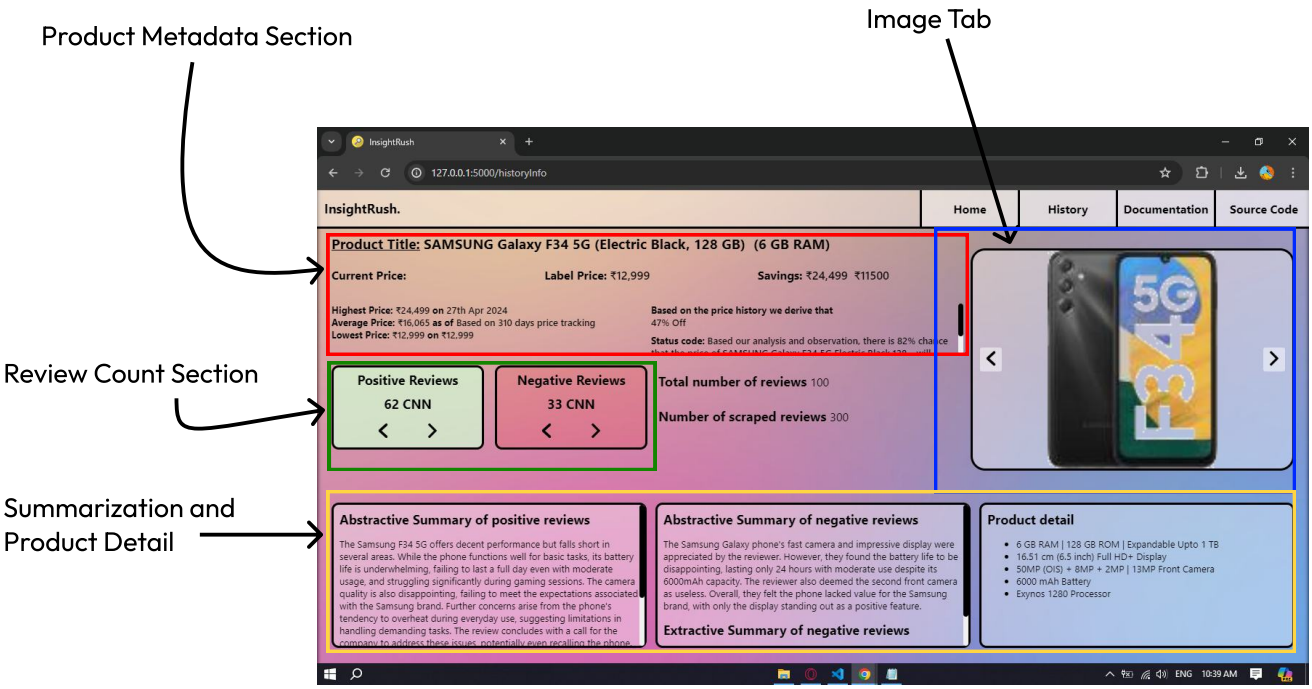
Number of reviews entry

After entering the product link, the user is directed to a page where they can specify the number of pages to scrape. This step is crucial as it allows users to control the amount of data retrieved, considering that not all reviews can be scraped and reviewed due to practical constraints.



Display Content Page

Upon specifying the number of pages, the application navigates to the display content page, where the information about the product is presented in a structured manner. This page serves as the central hub for analyzing the product, offering various insights and summaries to aid users in making informed decisions.

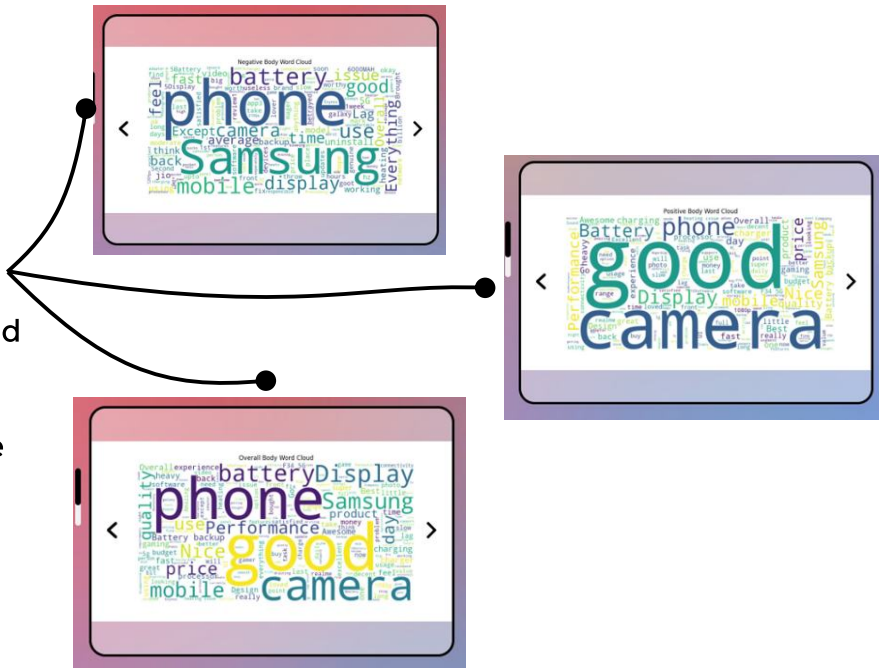


1. Product Metadata

This section provides essential details about the product, such as its title, current price, label price, savings, highest price, average price, lowest price, and suggestions based on price. These details help users understand the product's pricing dynamics and make informed purchasing decisions.

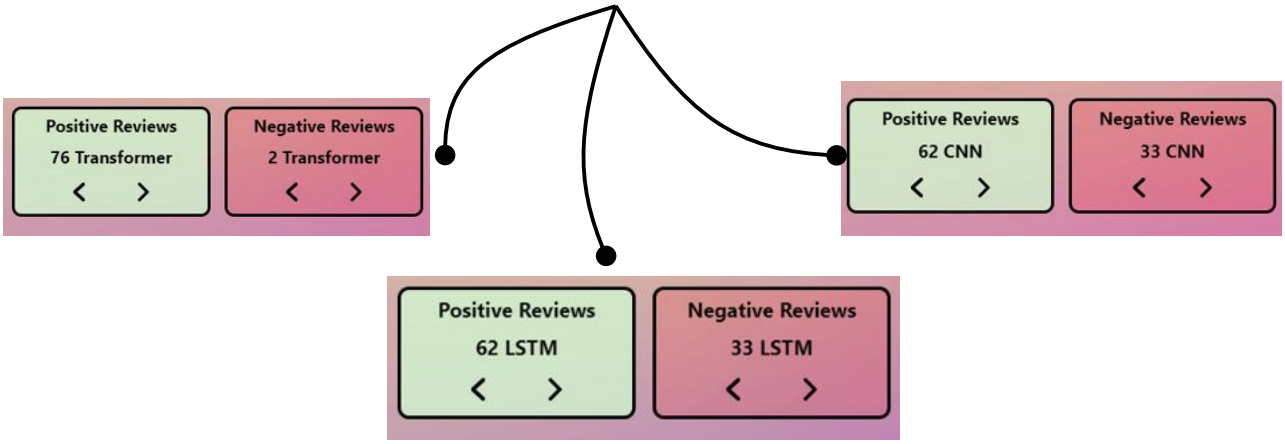
2. Images Tab

The images tab displays product images and word clouds generated from the reviews. Users can switch between different images and word clouds using arrow buttons, allowing them to visualize the product and the sentiment of its reviews.



3. Review Count

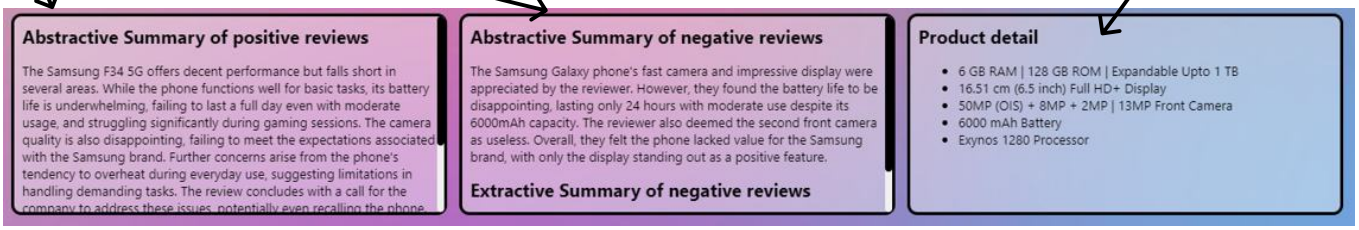
This section presents the count of positive and negative reviews classified by different models. Users can navigate through the counts using arrow buttons, gaining insights into the overall sentiment of the reviews.



4. Summarization and Product Details

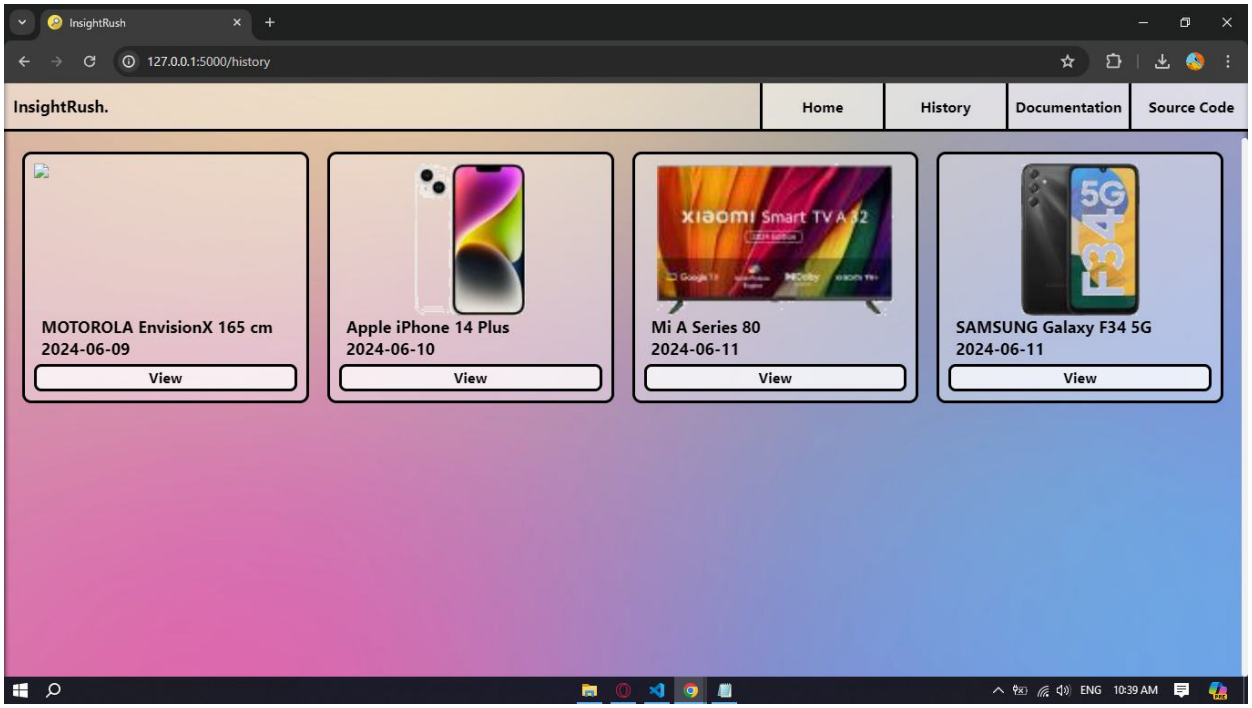
The last section is dedicated to summarizing the reviews and providing additional product details. It includes three boxes:

- The first box contains abstractive and extractive positive summaries, offering concise insights into the positive aspects of the product.
- The second box contains abstractive and extractive negative summaries, highlighting the negative aspects of the product.
- The last box provides detailed product information, helping users gain a comprehensive understanding of the product's features and specifications.



History Page

The history page serves as a repository of previously reviewed product details. Users can revisit and review the information stored in the MySQL database through this tab, enabling them to track their analysis and make comparisons between different products.



About the Dataset

Overview
- Contains 34,686,770 Amazon reviews from 6,643,669 users on 2,441,053 products.
- Subset includes 1,800,000 training samples and 200,000 testing samples for each polarity sentiment.

Origin
- Reviews are from Amazon spanning 18 years, up to March 2013.
- Dataset includes product and user information, ratings, and plaintext reviews.

Description
- Constructed by considering review scores 1 and 2 as negative, and 4 and 5 as positive, while ignoring score 3.
- Class 1 represents negative sentiment and class 2 represents positive sentiment.
- Each class has 1,800,000 training samples and 200,000 testing samples.

Format
- Available as CSV files `train.csv` and `test.csv`, containing training and testing samples respectively.
- Columns: `polarity` (1 for negative, 2 for positive), `title` (review heading), `text` (review body).
- Text is escaped using double quotes (`"`), internal double quotes are escaped by 2 double quotes (`""`), and new lines are escaped by a backslash followed by `n` character (`\n`).

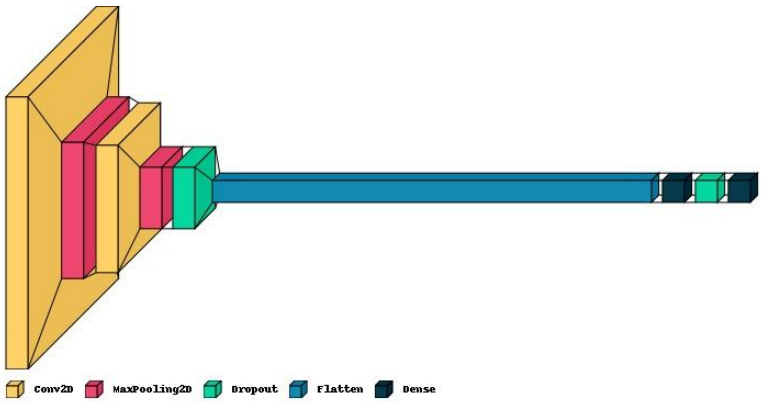
Citation
- Dataset is constructed by Xiang Zhang (xiang.zhang@nyu.edu).
- Used as a text classification benchmark in the paper: Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for Text Classification. Advances in Neural Information Processing Systems 28 (NIPS 2015).

Classification Models

CNN

This Convolutional Neural Network (CNN) model is designed for binary text classification. It begins with an embedding layer that converts input sequences of word indices into dense vectors of size 100, capturing semantic relationships in a continuous vector space. A Conv1D layer follows, applying 256 filters with a kernel size of 5 and ReLU activation to detect local patterns in the text. The Global Max Pooling layer then reduces each feature map to its maximum value, effectively down-sampling the data while retaining crucial features. A fully connected Dense layer with 256 neurons and ReLU activation learns higher-level representations from the pooled data.

To prevent overfitting, a Dropout layer sets 50% of its inputs to zero at each update during training. This is followed by a Batch Normalization layer, which normalizes the output, speeding up training and providing regularization. The final output layer is a single neuron with a sigmoid activation function, producing a probability score for binary classification. The model is compiled with the binary cross-entropy loss function, the Adam optimizer, and accuracy as the performance metric. Additionally, the training process incorporates `EarlyStopping` and `ReduceLROnPlateau` callbacks. `EarlyStopping` halts training if validation loss doesn't improve for three epochs, while `ReduceLROnPlateau` reduces the learning rate by a factor of 0.1 if validation loss stagnates for two epochs, with a minimum learning rate of 0.00001. These callbacks help in optimizing training efficiency and preventing overfitting, ensuring robust model performance.



CNN Model Architecture Diagram Generated using VisualKeras

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 100, 100)          1000000
_________________________________________________________________
conv1d (Conv1D)              (None, 96, 256)           128256
_________________________________________________________________
global_max_pooling1d (Global (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 1,195,329
Trainable params: 1,194,817
Non-trainable params: 512
_________________________________________________________________

<keras.engine.sequential.Sequential at 0x2a3bee4db70>
```

CNN Model Summary

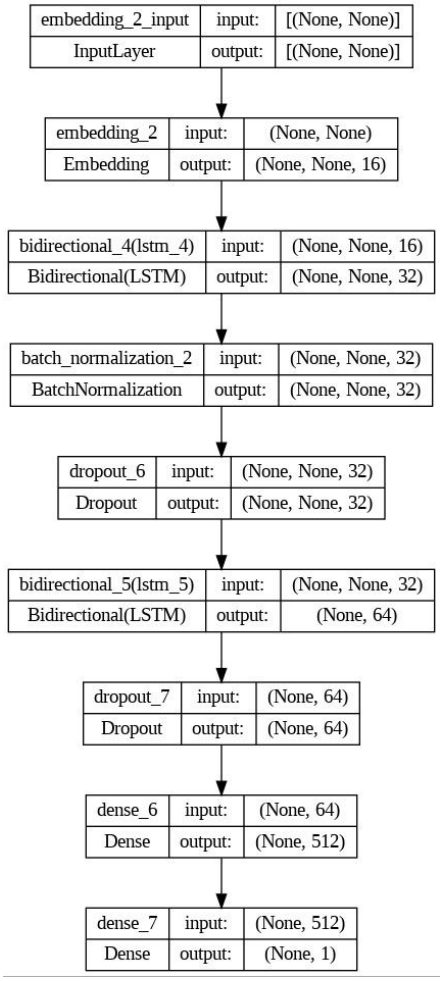
LSTM

This model is a Recurrent Neural Network (RNN) with Bidirectional Long Short-Term Memory (LSTM) layers designed for binary text classification. The model starts with an embedding layer that maps the input sequences of word indices into dense 16-dimensional vectors, enabling the capture of semantic relationships. Following this, the first Bidirectional LSTM layer with 16 units processes the sequences in both forward and backward directions, improving context understanding, and returns the full sequences (`return_sequences=True`). This output is normalized by a Batch Normalization layer, enhancing training speed and stability. A Dropout layer with a 50% rate is then applied to mitigate overfitting by randomly setting half of the inputs to zero during training. Next, another Bidirectional LSTM layer with 32 units processes the data, further capturing intricate temporal patterns. Another Dropout layer is applied to maintain regularization. The Dense layer with 512 neurons uses ReLU activation and includes L2 regularization to penalize large weights, encouraging the model to generalize better. Finally, a Dense layer with a single neuron and sigmoid activation provides the probability score for binary classification.

The model is compiled using the binary cross-entropy loss function, the Adam optimizer with an exponential decay learning rate schedule, which starts at 0.001 and decays by a factor of 0.9 every 10,000 steps, ensuring gradual learning rate reduction to stabilize training. The training includes an EarlyStopping callback that monitors validation loss and halts training if it doesn't improve for three consecutive epochs, restoring the best weights to prevent overfitting. The model is trained on `data_train` with labels from `df_train['label']` for up to 50 epochs, with a batch size of 32, and validated on `data_val` with labels from `df_test['label']`, using the defined early stopping callback to optimize performance and prevent overfitting.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 16)          160000
_________________________________________________________________
bidirectional (Bidirectional (None, None, 32)          4224
_________________________________________________________________
batch_normalization (BatchNo (None, None, 32)          128
_________________________________________________________________
dropout_6 (Dropout)          (None, None, 32)          0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 64)                16640
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               33280
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 513
=================================================================
Total params: 214,785
Trainable params: 214,721
Non-trainable params: 64
```

LSTM Model Summary

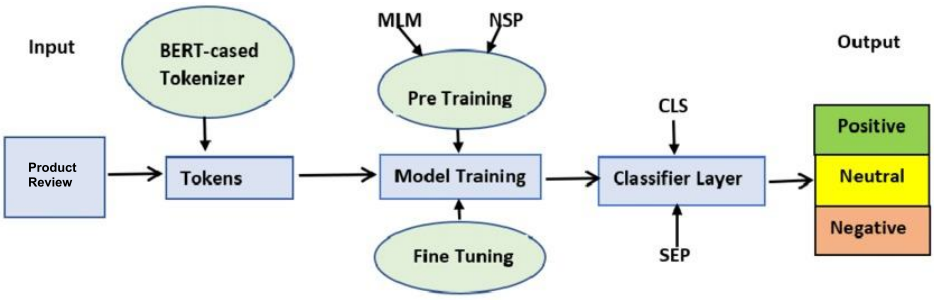


LSTM Model Architecture

BERT

The model AutoModelForSequenceClassification.from_pretrained("MarieAngeA13/Sentiment-Analysis-BERT") is a fine-tuned version of BERT (Bidirectional Encoder Representations from Transformers), specifically tailored for sentiment analysis. BERT, a transformer-based model, utilizes self-attention mechanisms to process input sequences in parallel, capturing long-range dependencies in text. Its bidirectional nature allows it to read text from both directions simultaneously, understanding context more comprehensively than unidirectional models. Pretrained on a vast corpus of texts like books and Wikipedia articles, BERT learns general language representations, which makes it effective for a wide range of NLP tasks. The specific model "MarieAngeA13/Sentiment-Analysis-BERT" has undergone fine-tuning on a sentiment analysis dataset, allowing it to learn the nuances of sentiment-specific tasks and adjust its pretrained weights for improved performance on such data. Designed for sequence classification, this model assigns labels to input sequences, identifying sentiments such as positive, negative, or neutral. Utilizing the Hugging Face Transformers library, the AutoModelForSequenceClassification class simplifies the use of BERT by providing easy APIs for loading, fine-tuning, and deploying the model. The from_pretrained method fetches the fine-tuned model's weights and configuration, making it ready for immediate use in inference or further training.

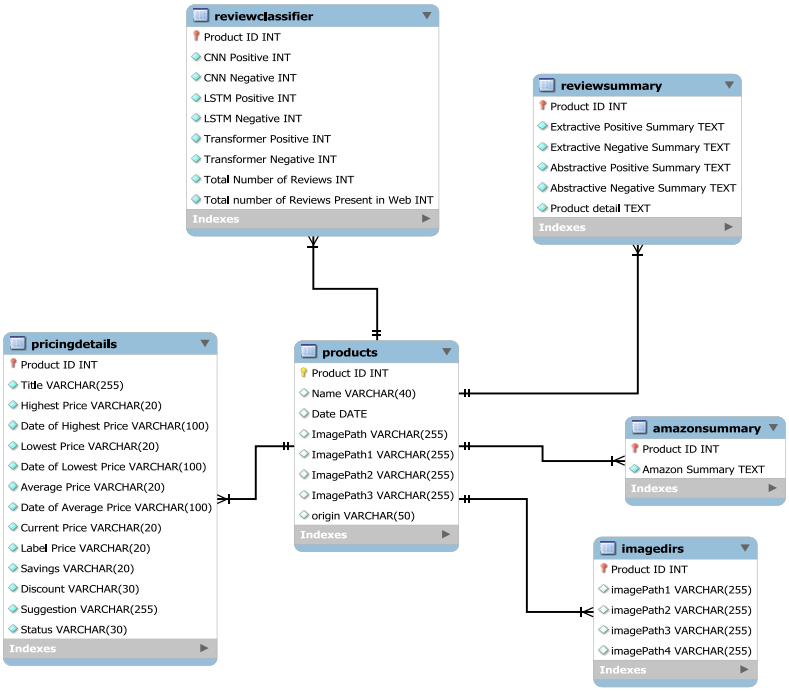
In practice, the input text is tokenized into word pieces, which are processed by BERT's transformer layers. The final hidden state of the classification token ([CLS]) is then passed through a classification head, which outputs probabilities for each sentiment class. This integration of advanced language modeling and fine-tuning makes the model highly effective for sentiment analysis tasks.



High level Architecture Diagram of BERT

ER Diagram

The following ER Diagram represents the database schema for storing and managing the retrieved information of products. Each table is designed to efficiently store different aspects of the product data, ensuring a structured and scalable approach to data management.



Performance Measure

Model\Performance Measure	Precision	Recall	F1 Score	Accuracy
CNN	0.54	0.88	0.67	0.65
LSTM	0.60	0.67	0.63	0.65
BERT	0.73	1.00	0.84	0.85

Table that summarizes various performance parameters of the implemented models
*values are calculated using 10 positive and 10 negative product reviews

CNN

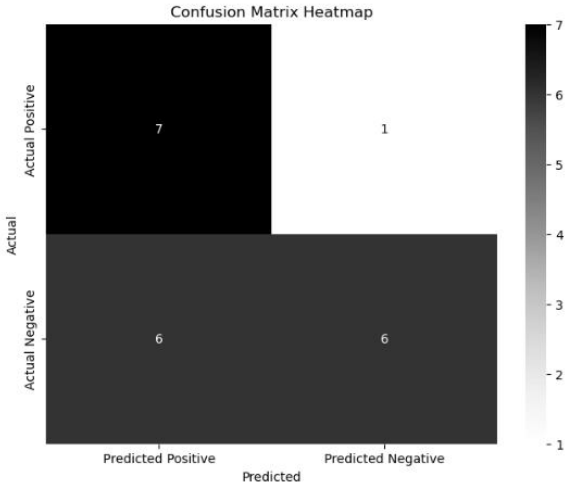
The CNN model has a relatively high recall, indicating it is good at identifying positive cases, but it has a lower precision, meaning there are more false positives. The overall accuracy is moderate, and the F1 score shows a balance between precision and recall.

LSTM

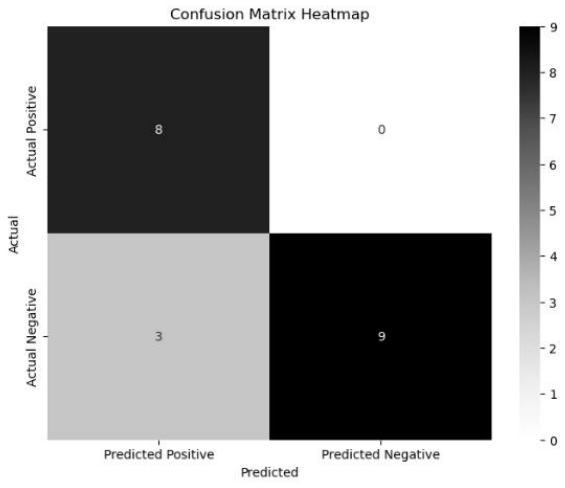
The LSTM model shows a more balanced precision and recall compared to CNN, though both are moderately low. The F1 score is slightly lower than the CNN, and the overall accuracy is the same as CNN. This indicates the LSTM model is more balanced but not significantly better in overall performance.

BERT

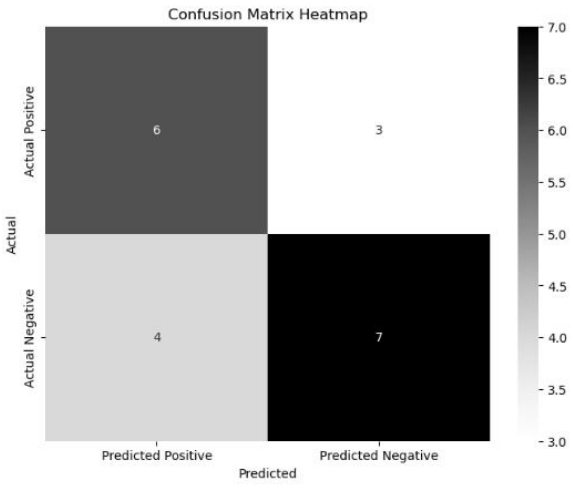
The BERT model outperforms both CNN and LSTM models across all metrics, with the highest precision, perfect recall, and the highest F1 score and accuracy. This makes BERT the most reliable and effective model for sentiment classification in this context.



Confusion matrix of CNN



Confusion matrix of BERT



Confusion matrix of LSTM

Future Works

1. **Reduced Waiting Time:** Implement optimizations to reduce the time it takes to gather and display product information. This could involve optimizing the scraping process, improving backend processing efficiency, or using caching mechanisms to store and retrieve data more quickly. Currently it takes about 7-8 minutes to display the product's detail.

2. **Enhanced UI Experience**: Improve the user interface to make it more visually appealing and user-friendly. This could include redesigning the layout, adding interactive elements, and improving the overall user experience to make it more engaging and intuitive.

3. **User Isolation and Profile Creation**: Implement a user login system to allow users to create profiles and personalize their experience. This would enable users to save and revisit products, view their browsing history, and receive personalized recommendations based on their preferences. Currently no user login system is implemented.

Source Code

HTML Files

home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>InsightRush</title>
    <link rel="stylesheet" href="../static/css/home.css">
    <link rel="icon" href="../static/Images/favicon.png" type="image/x-icon">
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
    <video autoplay muted loop id="video-bg">
        <source src="..\static\Images\bg.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
        {% include 'taskbar.html' %}

        <div class="content">
            <div class="spaceTop"></div>
            <div class="logo">Welcome to Insight Rush.</div>
            <div>
                <form action="/submitURL" method="post">
                    <input type="search" placeholder="Paste your amazon or flipkart
product link here" class="linkBox" name="searchURL">
                    <button type="submit" class="searchButton">Let's go</button>
                </form>
            </div>
            <div class="flash-message-spacer">
                {% with messages = get_flashed_messages(with_categories=true) %}
                    {% if messages %}
                        {% for category, message in messages %}
                            {{ message }}
                        {% endfor %}
                    {% endif %}
                {% endwith %}
            </div>
            <script>
                $('.linkBox').focus(function() {
                    $(this).attr('placeholder', '');
                }).blur(function() {
```

```
                $(this).attr('placeholder', 'Paste your amazon or flipkart product
link here');
            });
        </script>
    </div>
</body>
</html>
```

page1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../static/css/preInfo.css">
    <link rel="icon" href="../static/Images/favicon.png" type="image/x-icon">
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <title>InsightRush</title>
</head>
<body>
    {% include 'taskbar.html' %}
    <video autoplay muted loop id="video-bg">
        <source src="..\static\Images\bg.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
    <div class="info">
        <div class="containTitle">
            <div class="containTitle2">
                <div>
                    <p class="prodTitle"><u>Product Title:</u> {{content[0]}}</p>
                </div>
                <div>
                    <p class="nopgs">Number of reviews: {{content[1]}}</p>
                </div>
                <div class="containTitle3">
                    <div class="block1">
                        <p class="getNoPgs">Enter the number of pages. Available pages
{{content[2]}} (1 page have 10 reviews): </p>
                    </div>
                    <div class="block2">
                        <form action="/pageNumbers" method="post">
                            <input type="text" placeholder="Number of pages" class="pgBox"
name="pgNum">

                        </form>
                    </div>
```

```
                    <div class="block3">
                    </div>
                </div>
                <div class="flash-message-spacer">
                    {% with messages = get_flashed_messages(with_categories=true) %}
                        {% if messages %}
                            {% for category, message in messages %}
                                {{ message }}
                            {% endfor %}
                        {% endif %}
                    {% endwith %}
                </div>
                <script>
                    $('.pgBox').focus(function() {
                        $(this).attr('placeholder', '');
                    }).blur(function() {
                        $(this).attr('placeholder', 'Number of pages');
                    });
                </script>
            </div>
        </div>
        <div class="imgPropContain">
            <img src="/image"alt="Product Image" class="imgProp">
        </div>
    </div>

</body>
</html>
```

page2Amazon.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../static/css/postInfo1.css">
    <link rel="icon" href="../static/Images/favicon.png" type="image/x-icon">
    <title>InsightRush</title>
</head>
<body>
    {% include 'taskbar.html' %}
    <video autoplay muted loop id="video-bg">
        <source src="..\static\Images\bg.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
```

```
    <div class="info">
        <div class="innerInfo">
                <div class="innerInfo2">
                <div class="block1">
                    <span><u>Product Title:</u> {{content.Title}}</span>
                </div>
                <div class="innerInfo3">
                    <div class="block2">
                        <span><b>Current Price:</b> {{content.CurrentPrice}}</span>
                    </div>
                    <div class="block3">
                        <span><b>Label Price:</b> {{content.LabelPrice}}</span>
                    </div>
                    <div class="block4">
                        <span><b>Savings:</b>
{{content.Savings}}  {{content.Discount}}</span>
                    </div>
                </div>
                <div class="innerInfo4">
                    <div class="block5">
                        <div>
                            <span><b>Highest Price:</b> {{content.HighestPrice}} <b>on</b>
{{content.DateofHighestPrice}}</span>
                        </div>
                        <div>
                            <span><b>Average Price:</b> {{content.AveragePrice}} <b>as
of</b> {{content.DateofAveragePrice}}</span>
                        </div>
                        <div>
                            <span><b>Lowest Price:</b> {{content.LowestPrice}} <b>on</b>
{{content.DateofLowestPrice}}</span>
                        </div>
                    </div>
                    <div class="block6">
                        <div>
                            <span><b>Based on the price history we derive that</b></span>
                        </div>
                        <div>
                            <span>{{content.Suggestion}}</span>
                        </div>
                        <div style="margin-top: 8px;">
                            <span><b>Status code:</b> {{content.Status}}</span>
                        </div>
                    </div>
                </div>
                <div class="innerInfo5">
                    <div class="innerInfo6">
```

```
                        <div class="innerinnerInfo6">
                            <div>
                                <span>Positive Reviews</span>
                            </div>
                            <div class="slider" id="slider1">
                                <!-- <span>{{content['Positive1']}}</span> -->
                                <div class="slide">{{content.pos1}} CNN</div>
                                <div class="slide">{{content.pos2}} LSTM</div>
                                <div class="slide">{{content.pos3}} Transformer</div>
                            </div>
                            <div>
                                <button class="prev1"><img src="..\static\Images\left.png"
style="width: 20px; height: 20px;object-fit: contain;margin-right: 30px;"></button>
                                <button class="next1"><img
src="..\static\Images\right.png" style="width: 20px; height: 20px;object-fit:
contain;"></button>
                            </div>
                        </div>
                    </div>
                    <div class="innerInfo7">
                        <div class="innerinnerInfo7">
                            <div>
                                <span>Negative Reviews</span>
                            </div>
                            <div class="slider" id="slider2">
                                <!-- <span>{{content['Negative1']}}</span> -->
                                <div class="slide">{{content.neg1}} CNN</div>
                                <div class="slide">{{content.neg2}} LSTM</div>
                                <div class="slide">{{content.neg3}} Transformer</div>
                            </div>
                            <div>
                                <button class="prev1"><img src="..\static\Images\left.png"
style="width: 20px; height: 20px;object-fit: contain;margin-right: 30px;"></button>
                                <button class="next1"><img
src="..\static\Images\right.png" style="width: 20px; height: 20px;object-fit:
contain;"></button>
                            </div>
                        </div>
                        <script src="../static/js/script.js"></script>
                    </div>

                    <script src="../static/js/script.js"></script>
                    <div class="innerInfo8">
                        <div class="amazonAITitle">
                            <span>Amazon's AI summarizer</span>
                        </div>
                        <div class="amazonAIP">
```

```
                            <p style="font-size: small; font-weight:
400;;">{{content.AmazonAI}}"</p>
                        </div>
                    </div>
                </div>
            </div>
        </div>
        <div class="imgPropContain">
            <img src="\image" class="sliderImage" alt="product-image">
            <img src="\image1" class="sliderImage" alt="wordcloud1">
            <img src="\image22" class="sliderImage" alt="wordcloud2">
            <img src="\image3" class="sliderImage" alt="wordcloud2">
            <button class="prev" onclick="prevSlide()"><img
src="..\static\Images\left.png" style="width: 20px; height: 20px;object-fit:
contain;"></button>
            <button class="next" onclick="nextSlide()"><img
src="..\static\Images\right.png" style="width: 20px; height: 20px;object-fit:
contain;"></button>
        </div>
        <script src="../static/js/script2.js"></script>
    </div>
    <div class="texts">
        <div class="innerText1">
            <span style="font-size: large;font-weight: bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Abstractive Summary of positive reviews
            </span>
            <p style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {{content.posSum}}
            </p>
            <span style="font-size: large;font-weight: bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Extractive Summary of positive reviews
            </span>
            <p style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {{content.EposSum}}
            </p>
        </div>
        <div class="innerText2">
            <span style="font-size: large;font-weight:bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Abstractive Summary of negative reviews
            </span>
            <p style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
```

```
                {{content.negSum}}
            </p>
            <span style="font-size: large;font-weight:bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Extractive Summary of negative reviews
            </span>
            <p style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {{content.EnegSum}}
            </p>
        </div>
        <div class="innerText3">
            <span style="font-size: large;font-weight: bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Total number of reviews
                <span style="font-size: medium;font-weight:normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                    {{content.NumRev}}
                </span>
            <br>
            </span>
            <span style="font-size: large;font-weight: bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Number of scraped reviews
                <span style="font-size: medium;font-weight:normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                    {{content.UsrNumRev}}
                </span>
            </span>
            <br><br>
            <span style="font-size: large;font-weight: bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Product detail
            </span>
            <ul style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {% for item in content.About %}
                <li>{{ item }}</li>
                {% endfor %}
            </ul>
        </div>
    </div>
</body>
</html>
```

page2Flipkart.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../static/css/postInfo2.css">
    <link rel="icon" href="../static/Images/favicon.png" type="image/x-icon">
    <title>InsightRush</title>
</head>
<body>
    {% include 'taskbar.html' %}
    <video autoplay muted loop id="video-bg">
        <source src="..\static\Images\bg.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
    <div class="info">
        <div class="innerInfo">
                <div class="innerInfo2">
                <div class="block1">
                    <span><u>Product Title:</u> {{content.Title}}</span>
                </div>
                <div class="innerInfo3">
                    <div class="block2">
                        <span><b>Current Price:</b> {{content.CurrentPrice}}</span>
                    </div>
                    <div class="block3">
                        <span><b>Label Price:</b> {{content.LabelPrice}}</span>
                    </div>
                    <div class="block4">
                        <span><b>Savings:</b>
{{content.Savings}}  {{content.Discount}}</span>
                    </div>
                </div>
                <div class="innerInfo4">
                    <div class="block5">
                        <div>
                            <span><b>Highest Price:</b> {{content.HighestPrice}} <b>on</b>
{{content.DateofHighestPrice}}</span>
                        </div>
                        <div>
                            <span><b>Average Price:</b> {{content.AveragePrice}} <b>as
of</b> {{content.DateofAveragePrice}}</span>
                        </div>
                        <div>
```

```
                        <span><b>Lowest Price:</b> {{content.LowestPrice}} <b>on</b>
{{content.DateofLowestPrice}}</span>
                    </div>
                </div>
                <div class="block6">
                    <div>
                        <span><b>Based on the price history we derive that</b></span>
                    </div>
                    <div>
                        <span>{{content.Suggestion}}</span>
                    </div>
                    <div style="margin-top: 8px;">
                        <span><b>Status code:</b> {{content.Status}}</span>
                    </div>
                </div>
            </div>
            <div class="innerInfo5">
                <div class="innerInfo6">
                    <div class="innerinnerInfo6">
                        <div>
                            <span>Positive Reviews</span>
                        </div>
                        <div class="slider" id="slider1">
                            <!-- <span>{{content['Positive1']}}</span> -->
                            <div class="slide">{{content.pos1}} CNN</div>
                            <div class="slide">{{content.pos2}} LSTM</div>
                            <div class="slide">{{content.pos3}} Transformer</div>
                        </div>
                        <div>
                            <button class="prev1"><img src="..\static\Images\left.png"
style="width: 20px; height: 20px;object-fit: contain;margin-right: 30px;"></button>
                            <button class="next1"><img
src="..\static\Images\right.png" style="width: 20px; height: 20px;object-fit:
contain;"></button>
                        </div>
                    </div>
                </div>
                <div class="innerInfo7">
                    <div class="innerinnerInfo7">
                        <div>
                            <span>Negative Reviews</span>
                        </div>
                        <div class="slider" id="slider2">
                            <!-- <span>{{content['Negative1']}}</span> -->
                            <div class="slide">{{content.neg1}} CNN</div>
                            <div class="slide">{{content.neg2}} LSTM</div>
                            <div class="slide">{{content.neg3}} Transformer</div>
```

```
                            </div>
                            <div>
                                <button class="prev1"><img src="..\static\Images\left.png"
style="width: 20px; height: 20px;object-fit: contain;margin-right: 30px;"></button>
                                <button class="next1"><img
src="..\static\Images\right.png" style="width: 20px; height: 20px;object-fit:
contain;"></button>
                            </div>
                        </div>
                        <script src="../static/js/script.js"></script>
                    </div>

                    <script src="../static/js/script.js"></script>
                    <div class="innerInfo8">
                        <span style="font-size: large;font-weight: bold;font-family:
'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;">
                            Total number of reviews
                            <span style="font-size: medium;font-weight:normal;font-family:
'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;">
                                {{content.NumRev}}
                            </span>
                        </span>
                        <br>
                        <span style="font-size: large;font-weight: bold;font-family:
'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;">
                            Number of scraped reviews
                            <span style="font-size: medium;font-weight:normal;font-family:
'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;">
                                {{content.UsrNumRev}}
                            </span>
                        </span>
                    </div>
                </div>
            </div>
        </div>
        <div class="imgPropContain">
            <img src="\image" class="sliderImage" alt="product-image">
            <img src="\image1" class="sliderImage" alt="wordcloud1">
            <img src="\image22" class="sliderImage" alt="wordcloud2">
            <img src="\image3" class="sliderImage" alt="wordcloud2">
            <button class="prev" onclick="prevSlide()"><img
src="..\static\Images\left.png" style="width: 20px; height: 20px;
object-fit:contain;"></button>
            <button class="next" onclick="nextSlide()"><img
src="..\static\Images\right.png" style="width: 20px; height: 20px;object-fit:
contain;"></button>
        </div>
```

```
        <div>

        </div>
        <script src="../static/js/script2.js"></script>
    </div>
    <div class="texts">
        <div class="innerText1">
            <span style="font-size: large;font-weight: bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Abstractive Summary of positive reviews
            </span>
            <p style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {{content.posSum}}
            </p>
            <span style="font-size: large;font-weight: bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Extractive Summary of positive reviews
            </span>
            <p style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {{content.EposSum}}
            </p>
        </div>
        <div class="innerText2">
            <span style="font-size: large;font-weight:bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Abstractive Summary of negative reviews
            </span>
            <p style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {{content.negSum}}
            </p>
            <span style="font-size: large;font-weight:bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Extractive Summary of negative reviews
            </span>
            <p style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {{content.EnegSum}}
            </p>
        </div>
        <div class="innerText3">
            <span style="font-size: large;font-weight: bold;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                Product detail
            <span>
```

```
            <ul style="font-size: small;font-weight: normal;font-family: 'Segoe UI',
Tahoma, Geneva, Verdana, sans-serif;">
                {% for item in content.About %}
                <li>{{ item }}</li>
                {% endfor %}
            </ul>
        </div>
    </div>
</body>
</html>
```

taskbar.html

```
<link rel="stylesheet" href="../static/css/home.css">
<div class="taskBar">
    <div class="taskBarLogo">InsightRush.</div>
    <div class="spaceBetween"></div>
    <div class="buttons">
        <a href="{{ url_for('index') }}"><button class="homeButton">Home</button></a>
        <a href="{{ url_for('history') }}"><button class="homeButton">History</button></a>
        <a href="https://github.com/RameezAkther/InsightRush_AIPoweredProductReviewer"
target="_blank"><button class="docButton">Documentation</button></a>
        <a
href="https://drive.google.com/drive/folders/1UTgXACEFKc5onzHnD3lhD2tKpGgBXBTb?usp=sharing
" target="_blank"><button class="sourceButton">Source Code</button></a>
    </div>
</div>
```

history.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>InsightRush</title>
    <link rel="stylesheet" href="../static/css/history.css">
    <link rel="icon" href="../static/Images/favicon.png" type="image/x-icon">
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>

<body>
    {% include 'taskbar.html' %}
    <video autoplay muted loop id="video-bg">
        <source src="..\static\Images\bg.mp4" type="video/mp4">
        Your browser does not support the video tag.
    </video>
```

```
    <div class = "bound">
        <div class="grid-container">
            {% for product in content %}
            <div class="grid-item">
                <div class="block1">
                    <img src="/image2?{{ loop.index }}" class="imgProp">
                </div>
                <div class="block2">
                    <span class="prodName">{{ product[1] }}</span>
                    <span class="prodName">{{ product[2] }}</span>
                    <button class="go" onclick="viewProduct('{{ product[0]
}}')">View</button>
                </div>
            </div>
            {% endfor %}
            <script>
                function viewProduct(productID) {
                    $.ajax({
                        type: "POST",
                        url: "/history",
                        data: {productID: productID},
                        success: function(response) {
                            // Handle the response here if needed
                            console.log("Product ID sent to Flask: " + productID);
                            window.location.href = "/historyInfo";
                        },
                        error: function(err) {
                            console.error("Error sending product ID to Flask: " + err);
                        }
                    });
                }
                </script>
        </div>
    </div>
</body>
```

CSS Files

home.css

```
body{
    margin: 0px;
    padding-top: 60px;
}

#video-bg {
```

```css
    position: fixed;
    top: 0;
    left: 0;
    width: 100vw; /* Viewport width */
    height: 100vh; /* Viewport height */
    object-fit: cover; /* Cover the entire area */
    z-index: -1;
}

.taskBar{
    height: 50px;
    border-bottom: 3px solid;
    position: fixed;
    top:0px;
    left:0px;
    right:0px;
    display: flex;
    align-items: center;
    flex-direction: row;
    background-color: rgba(255, 255, 255, 0.4);
}

.homeButton:hover,.docButton:hover, .sourceButton:hover{
    cursor: pointer;
    background-color: black;
    color: white;
    border:0px;
    border-bottom: 0px;
    border-left: 3px solid;
    border-color: black;

}

.homeButton:active,.docButton:active, .sourceButton:active{
    cursor: pointer;
    background-color: grey;
    transition: background-color 0s;
}

.homeButton, .docButton, .sourceButton{
    display: inline-block;
    height: 50px;
    width: 135px;
    font-size: medium;
    font-weight: bold;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    margin: 0px;
```

```
    border: 0px;
    padding: 0px;
    margin-left: 0px;
    border-left: 3px solid;
    margin-right: -4px;
    background-color: rgba(255, 255, 255, 0.4);
    transition: background-color 0.2s ease;
}

.taskBarLogo{
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-size: large;
    font-weight: bold;
    padding-left: 10px;
}

.content{
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: space-between;
    height: 50vh;
}

.spaceBetween{
    flex: 1;
    min-width: 70px;
}

.buttons{
    flex-shrink: 0;
}

.spaceTop{
    flex:1;
}

.linkBox::placeholder{
    color:black;
    font-weight:600;
}

input[type="search"]::-webkit-search-cancel-button {
    -webkit-appearance: none;
    height: 15px;
    width: 15px;
    background: url('../Images/close.png') no-repeat center center;
```

```
    background-size: contain;
    cursor: pointer;
}

.linkBox{
    width: 50vw;
    outline: none;
    max-width: 500px;
    border: 3px solid;
    border-color: black;
    height: 35px;
    border-top-left-radius: 15px;
    border-bottom-left-radius: 15px;
    padding-left: 13px;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    color: black;
    background-color: rgba(255, 255, 255, 0.4);
}

.searchButton:hover{
    cursor: pointer;
    background-color: black;
    color:white;
}

.searchButton:active{
    background-color: grey;
    transition: background-color 0s;
}

.searchButton{
    height: 35px;
    width: 90px;
    border-right: 3px solid;
    border-top: 3px solid;
    border-bottom: 3px solid;
    border-left: 0px;
    border-color: black;
    background-color: rgba(255, 255, 255, 0.4);
    color: black;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-weight:bold;
    font-size: small;
    border-top-right-radius: 17px;
    border-bottom-right-radius: 17px;
    margin-left: -4px;
    transform: background-color 0.2s;
```

```
}

.logo{
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-size: 75px;
    font-weight: bolder;
    margin-bottom: 30px;
}

.flash-message-spacer {
    height: 30px;
    margin-top: 10px;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-size: medium;
    font-weight: 500;
    color: red;
}
```

preinfo.css

```
#video-bg {
    position: fixed;
    top: 0;
    left: 0;
    width: 100vw; /* Viewport width */
    height: 100vh; /* Viewport height */
    object-fit: cover; /* Cover the entire area */
    z-index: -1;
}

.info{
    margin-left: 20px;
    margin-right: 20px;
    display: flex;
    flex-direction: row;
}

.containTitle{
    flex:2;
}

.containTitle2{
    display: flex;
    flex-direction: column;
}
```

```css
.containTitle3{
    display: flex;
    flex-direction: row;
    align-items: center;
}

.prodTitle{
    font-size: larger;
    font-weight: 700;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    margin: 0px;
    margin-right: 60px;
}

.nopgs{
    font-size: large;
    font-weight: 500;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.getNoPgs{
    font-size: medium;
    font-weight: 500;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    display: inline;
    margin-right: 20px;
}

.pgBox::placeholder{
    color: grey;
}

.pgBox{
    border: 2px solid;
    height: 25px;
    width: 120px;
    border-radius: 10px;
    padding-left: 10px;
    background-color: rgba(255, 255, 255, 0.4);
    outline: 0;
}

.flash-message-spacer {
    height: 30px;
    margin-top: 10px;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-size: medium;
```

```
    font-weight: 500;
    color: red;
}

.imgPropContain{
    flex:1;
    height: 300px;
    margin-top: 20px;
    border: 3px solid;
    border-radius: 20px;
    overflow: hidden;
    background-color: rgba(255, 255, 255, 0.4);
}

.imgProp{
    width: 100%;
    height: 300px;
    display: block;
    object-fit: contain;
}


.block3{
    flex:1;
}
```

postInfo1.css

```
/* width */
::-webkit-scrollbar {
    width: 7px;
  }

  /* Track */
  ::-webkit-scrollbar-track {
    background: #f1f1f1;
    border-radius: 5px;
  }

  /* Handle */
  ::-webkit-scrollbar-thumb {
    background: black;
    border-radius: 5px;
  }

  /* Handle on hover */
```

```
  ::-webkit-scrollbar-thumb:hover {
    background: #555;
  }


span{
    margin:0;
}

#video-bg {
    position: fixed;
    top: 0;
    left: 0;
    width: 100vw; /* Viewport width */
    height: 100vh; /* Viewport height */
    object-fit: cover; /* Cover the entire area */
    z-index: -1;
}

button {
    background-color: transparent;
    border: none;
    padding: 5px 7px;
    cursor: pointer;
}

.info{
    margin-top: 0px;
    margin-left: 20px;
    margin-right: 20px;
    display: flex;
    flex-direction: row;
    height: 300px;
}

.innerInfo{
    flex:2;
}

.innerInfo2{
    display: flex;
    flex-direction: column;
}

.block1{
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-size: larger;
```

```
    font-weight: 700;
    margin-right: 60px;
}

.innerInfo3{
    display: flex;
    height: 35px;
    overflow: hidden;
    flex-direction: row;
    justify-content:space-around;
    margin-top: 20px;
    margin-bottom: 15px;
    font-size: medium;
    font-weight: 500;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.innerInfo4{
    display: flex;
    height: 70px;
    flex-direction: row;

    justify-content:space-around;
    font-size: medium;
    font-weight: 500;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.innerInfo5{
    display: flex;
    flex-direction: row;
    height: 115px;
    margin-top: 15px;
    font-size: large;
    font-weight: bold;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.innerInfo6{
    flex: 1;
    margin-right: 15px;
}

.innerinnerInfo6{
    display: flex;
    flex-direction: column;
    align-items: center;
```

```
    justify-content:space-around;
    border: 3px solid;
    border-radius: 10px;
    height: 110px;
    background-color: #d2ffd5b6;
    overflow: hidden;
}

.innerinnerInfo7{
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content:space-around;
    border: 3px solid;
    border-radius: 10px;
    height: 110px;
    background-color: rgba(247, 30, 30, 0.164);
    overflow: hidden;
}

.innerInfo7{
    flex:1;
    margin-right: 15px;
}

.innerInfo8{
    flex: 2;
    display: flex;
    flex-direction: column;
    margin-right: 10px;
    overflow: auto;
}

.amazonAIP{
    overflow: auto;
}



.block2{
    flex: 1;
    margin-right: 10px;
}

.block3{
    flex: 1;
    margin-right: 10px;
```

```
}

.block4{
    flex:1;
    margin-right: 10px;
}

.block5{
    flex:1;
    overflow: auto;
    margin-right: 10px;
    font-size: small;
}

.block6{
    flex:1;
    margin-right: 10px;
    overflow: auto;
    font-size: small;
}

.block7{
    border: 3px solid;
    border-radius: 10px;
    margin-right:10px;
}

.block8{
    display: flex;
    border: 3px solid;
    border-radius: 10px;
    margin-right:10px;

}

.prev {
    position: absolute;
    top: 50%;
    left: 10px;
    transform: translateY(-50%);
    background-color: rgba(255, 255, 255, 0.5);
    border: none;
    border-radius: 3px;
    padding: 5px;
    cursor: pointer;
}
```

```css
.next {
    position: absolute;
    top: 50%;
    right: 10px;
    transform: translateY(-50%);
    background-color: rgba(255, 255, 255, 0.5);
    border: none;
    border-radius: 3px;
    padding: 5px;
    cursor: pointer;
}

.imgPropContain{
    flex:1;
    margin-top: 20px;
    position: relative;
    height: 300px;
    border: 3px solid;
    border-radius: 20px;
    overflow: hidden;
    background-color: rgba(255, 255, 255, 0.4);
}

.sliderImage{
    width: 100%;
    height: 300px;
    display: block;
    object-fit: contain;
}

.texts{
    flex: 1;
    height: 200px;
    display: flex;
    flex-direction: row;
    margin-top: 70px;
    margin-right: 20px;
    margin-left: 20px;
}

.innerText1{
    flex:1;
    border: 3px solid;
    margin-right: 12px;
    border-radius: 10px;
    padding-left: 8px;
    padding-top: 8px ;
```

```
    overflow: auto;
    background-color: rgba(255, 255, 255, 0.4);
    overflow: scroll;
}

.innerText2{
    flex:1;
    border: 3px solid;
    margin-right: 12px;
    border-radius: 10px;
    padding-left: 8px;
    padding-top: 8px ;
    overflow: auto;
    background-color: rgba(255, 255, 255, 0.4);
    overflow: scroll;
}

.innerText3{
    flex:1;
    border: 3px solid;
    border-radius: 10px;
    padding-left: 8px;
    padding-top: 8px ;
    overflow: auto;
    background-color: rgba(255, 255, 255, 0.4);
}
```

postInfo2.css

```
/* width */
::-webkit-scrollbar {
    width: 7px;
  }

  /* Track */
  ::-webkit-scrollbar-track {
    background: #f1f1f1;
    border-radius: 5px;
  }

  /* Handle */
  ::-webkit-scrollbar-thumb {
    background: black;
    border-radius: 5px;
  }
```

```css
  /* Handle on hover */
  ::-webkit-scrollbar-thumb:hover {
    background: #555;
  }


span{
    margin:0;
}

#video-bg {
    position: fixed;
    top: 0;
    left: 0;
    width: 100vw; /* Viewport width */
    height: 100vh; /* Viewport height */
    object-fit: cover; /* Cover the entire area */
    z-index: -1;
}

button {
    background-color: transparent;
    border: none;
    padding: 5px 7px;
    cursor: pointer;
}

.info{
    margin-top: 0px;
    margin-left: 20px;
    margin-right: 20px;
    display: flex;
    flex-direction: row;
    height: 300px;
}

.innerInfo{
    flex:2;
}

.innerInfo2{
    display: flex;
    flex-direction: column;
}

.block1{
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
```

```
    font-size: larger;
    font-weight: 700;
    margin-right: 60px;
}

.innerInfo3{
    display: flex;
    height: 35px;
    overflow: hidden;
    flex-direction: row;
    justify-content:space-around;
    margin-top: 20px;
    margin-bottom: 15px;
    font-size: medium;
    font-weight: 500;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.innerInfo4{
    display: flex;
    height: 70px;
    flex-direction: row;

    justify-content:space-around;
    font-size: medium;
    font-weight: 500;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.innerInfo5{
    display: flex;
    flex-direction: row;
    height: 115px;
    margin-top: 15px;
    font-size: large;
    font-weight: bold;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.innerInfo6{
    flex: 1;
    margin-right: 15px;
}

.innerinnerInfo6{
    display: flex;
    flex-direction: column;
```

```
    align-items: center;
    justify-content:space-around;
    border: 3px solid;
    border-radius: 10px;
    height: 110px;
    background-color: #d2ffd5b6;
    overflow: hidden;
}

.innerinnerInfo7{
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content:space-around;
    border: 3px solid;
    border-radius: 10px;
    height: 110px;
    background-color: rgba(247, 30, 30, 0.164);
    overflow: hidden;
}

.innerInfo7{
    flex:1;
    margin-right: 15px;
}

.innerInfo8{
    flex: 2;
    display: flex;
    margin-top: 10px;
    flex-direction: column;
    margin-right: 10px;
    overflow: auto;
}

.amazonAIP{
    overflow: auto;
}



.block2{
    flex: 1;
    margin-right: 10px;
}

.block3{
```

```
    flex: 1;
    margin-right: 10px;
}

.block4{
    flex:1;
    margin-right: 10px;
}

.block5{
    flex:1;
    overflow: auto;
    margin-right: 10px;
    font-size: small;
}

.block6{
    flex:1;
    margin-right: 10px;
    overflow: auto;
    font-size: small;
}

.block7{
    border: 3px solid;
    border-radius: 10px;
    margin-right:10px;
}

.block8{
    display: flex;
    border: 3px solid;
    border-radius: 10px;
    margin-right:10px;

}



.imgPropContain{
    flex:1;
    margin-top: 20px;
    position: relative;
    height: 300px;
    border: 3px solid;
    border-radius: 20px;
    overflow: hidden;
```

```
    background-color: rgba(255, 255, 255, 0.4);
}

.prev {
    position: absolute;
    top: 50%;
    left: 10px;
    transform: translateY(-50%);
    background-color: rgba(255, 255, 255, 0.5);
    border-radius: 3px;
    border: none;
    padding: 5px;
    cursor: pointer;
}

.next {
    position: absolute;
    top: 50%;
    right: 10px;
    transform: translateY(-50%);
    background-color: rgba(255, 255, 255, 0.5);
    border: none;
    border-radius: 3px;
    padding: 5px;
    cursor: pointer;
}

.sliderImage{
    width: 100%;
    height: 300px;
    display: block;
    object-fit: contain;
}

.texts{
    flex: 1;
    height: 200px;
    display: flex;
    flex-direction: row;
    margin-top: 70px;
    margin-right: 20px;
    margin-left: 20px;
}

.innerText1{
    flex:1;
    border: 3px solid;
```

```
    margin-right: 12px;
    border-radius: 10px;
    padding-left: 8px;
    padding-top: 8px ;
    overflow: auto;
    background-color: rgba(255, 255, 255, 0.4);
    overflow: auto;
}

.innerText2{
    flex:1;
    border: 3px solid;
    margin-right: 12px;
    border-radius: 10px;
    padding-left: 8px;
    padding-top: 8px ;
    overflow: auto;
    background-color: rgba(255, 255, 255, 0.4);
    overflow: auto;
}

.innerText3{
    flex:1;
    border: 3px solid;
    border-radius: 10px;
    padding-left: 8px;
    padding-top: 8px ;
    overflow: auto;
    background-color: rgba(255, 255, 255, 0.4);
}
```

history.css

```
/* width */
::-webkit-scrollbar {
    width: 7px;
  }

  /* Track */
  ::-webkit-scrollbar-track {
    background: #f1f1f1;
    border-radius: 5px;
  }

  /* Handle */
  ::-webkit-scrollbar-thumb {
    background: black;
```

```
    border-radius: 5px;
  }

  /* Handle on hover */
  ::-webkit-scrollbar-thumb:hover {
    background: #555;
  }

body{
    overflow: hidden;
}

.bound{
    height: 608px;
    overflow: scroll;
}

.grid-container {
    margin-top: 15px;
    margin-left: 20px;
    margin-right: 20px;
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(280px, 1fr)); /* Adjust the minimum
width (200px) as needed */
    grid-gap: 20px; /* Adjust the gap between grid items as needed */
}

#video-bg {
    position: fixed;
    top: 0;
    left: 0;
    width: 100vw; /* Viewport width */
    height: 100vh; /* Viewport height */
    object-fit: cover; /* Cover the entire area */
    z-index: -1;
}


.grid-item {
    background-color: rgba(255, 255, 255, 0.4);
    padding: 10px;
    text-align: center;
    border: 3px solid;
    height: 250px;
    border-radius: 10px;
    display: flex;
    flex-direction: column;
```

```
}

.block1{
    flex: 2;
    display: flex;
    overflow: hidden;
}

.imgProp{
    flex: 1;
    width: 100%;
    object-fit: contain;
    display: block;
    object-fit: contain;
}

.block2{
    flex: 1;
    display: flex;
    flex-direction: column;
    align-items: start;
}


.prodName{
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-size: large;
    font-weight: bold;
    margin-left: 8px;
}

.go:hover{
    cursor: pointer;
    background-color: black;
    color: white;
    border:0px;
    border-color: black;
}

.go:active{
    cursor: pointer;
    background-color: grey;
    transition: background-color 0s;
}

.go{
    border: 3px solid;
```

```
    background-color: rgba(255, 255, 255, 0.747);
    border-radius: 8px;
    margin-top: 5px;
    width: 100%;
    height: 30px;
    font-size: medium;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    font-weight: bold;
    transition: background-color 0.2s ease;
}
```

JavaScript Files

script.js

```
document.addEventListener('DOMContentLoaded', function() {
    const sliders = document.querySelectorAll('.slider');
    const prevButtons = document.querySelectorAll('.prev1');
    const nextButtons = document.querySelectorAll('.next1');

    sliders.forEach((slider, index) => {
        let currentSlideIndex = 0;
        const slides = slider.querySelectorAll('.slide');
        const totalSlides = slides.length;

        function showSlide(slideIndex) {
            slides.forEach((slide, i) => {
                if (i === slideIndex) {
                    slide.style.display = 'block';
                } else {
                    slide.style.display = 'none';
                }
            });
        }

        prevButtons[index].addEventListener('click', () => {
            currentSlideIndex = (currentSlideIndex - 1 + totalSlides) % totalSlides;
            showSlide(currentSlideIndex);
        });

        nextButtons[index].addEventListener('click', () => {
            currentSlideIndex = (currentSlideIndex + 1) % totalSlides;
            showSlide(currentSlideIndex);
        });
```

```
        // Show the initial slide
        showSlide(currentSlideIndex);
    });
});
```

script2.js

```
let currentSlide = 0;
const slides = document.querySelectorAll('.sliderImage');

function showSlide(index) {
    slides.forEach((slide, i) => {
        if (i === index) {
            slide.style.display = 'block';
        } else {
            slide.style.display = 'none';
        }
    });
    currentSlide = index;
}

function prevSlide() {
    currentSlide = (currentSlide - 1 + slides.length) % slides.length;
    showSlide(currentSlide);
}

function nextSlide() {
    currentSlide = (currentSlide + 1) % slides.length;
    showSlide(currentSlide);
}

// Show the first slide initially
showSlide(currentSlide);
```

Web Scraping Python files

getAmazonpgs.py

```
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.chrome.options import Options
```

```
def totalReview(productUrl):
    try:
        chrome_options = Options()
        chrome_options.add_argument('--headless')
        chrome_options.add_argument('--disable-gpu')
        chrome_options.add_argument("--silent")
        chrome_options.add_argument('--log-level=3')
        driver = webdriver.Chrome(options=chrome_options)
        productUrl = productUrl.replace("dp", "product-reviews") + "&pageNumber=" + str(1)
        driver.get(productUrl)
        WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.CSS_SELECTOR,
'div[data-hook="cr-filter-info-review-rating-count"]')))
        page_source = driver.page_source
        soup = BeautifulSoup(page_source, 'html.parser')
        reviews = soup.find('div', {'data-hook': 'cr-filter-info-review-rating-count'})
        total_reviews_text = reviews.text.strip().split(', ')[1].split(" ")[0]
        return int(total_reviews_text.replace(",",""))
    except:
        raise RuntimeError("Unable to scrape review count")
```

getFlipkartpgs.py

```
import requests
from bs4 import BeautifulSoup
import re

def totalReviews(productUrl):
    try:
        productUrl = productUrl.replace("/p/", "/product-reviews/")
        resp = requests.get(productUrl)
        soup = BeautifulSoup(resp.text, 'html.parser')
        reviews_span = soup.find('span',{'class': 'Wphh3N'})
        if reviews_span:
            no = reviews_span.text
            no = no.replace(",","")
            numbers = re.search(r'(\d+)\s+Reviews', no).group(1)
            if numbers:
                if int(numbers) >= 300:
                    return 300
                else:
                    return int(numbers)
        return 0
    except:
        raise RuntimeError("Unable to get review count")
```

```
#print(totalReviews("https://www.flipkart.com/hisense-e7k-126-cm-50-inch-qled-ultra-hd-4k-smart-vidaa-tv-dolby-vision-atmos/p/itm62fd1e22110c2?pid=TVSGSZGZY6QMR5JH&lid=LSTTVSGSZGZY6QMR5JH1AIHVY&marketplace=FLIPKART&store=ckf%2Fczl&srno=b_1_1&otracker=browse&fm=organic&iid=en_YqA2-K3dy-kt1A1AkUEXy3XC2wgOX4KCXXCuvYsRRy4CsphHTMT0hzWNPP8aqMncKhOVm2IPio8jXkH2OqIqOPUFjCTyOHoHZs-Z5_PS_w0%3D&ppt=pp&ppn=pp&ssid=3kppifnzr40000001714544768493"))
```

amazonProductReview.py

```
import pandas as pd
from bs4 import BeautifulSoup
from .maxFile import get_max_numbered_csv
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.chrome.options import Options
import keyboard


reviewlist = []

def extractReviews(driver, reviewUrl, pageNumber):
    try:
        driver.get(reviewUrl)
        WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.CSS_SELECTOR, 'div[data-hook="review"]')))
        page_source = driver.page_source
        soup = BeautifulSoup(page_source, 'html.parser')
        reviews = soup.findAll('div', {'data-hook': 'review'})
        for item in reviews:
            review = {
                'productTitle': soup.title.text.replace("Amazon.in:Customer reviews: ", "").strip(),
                'Rating': item.find('i', {'data-hook': 'review-star-rating'}).text.strip(),
                'Review Title': item.find('a', class_='review-title').get_text(strip=True).split('stars')[1],
                'Review Body': item.find('span', {'data-hook': 'review-body'}).text.strip(),
            }
            reviewlist.append(review)
    except:
        raise RuntimeError("Unable to scrape reviews")

def get_review(prod, pg):
    productUrl = prod
    chrome_options = Options()
```

```
    chrome_options.add_argument('--headless')
    chrome_options.add_argument('--disable-gpu')
    chrome_options.add_argument("--silent")
    chrome_options.add_argument('--log-level=3')
    driver = webdriver.Chrome(options=chrome_options)
    pg = int(pg)
    try:
        reviewUrl = productUrl.replace("dp", "product-reviews") + "&pageNumber=" + str(1)
        #totalRev = totalReview(driver, reviewUrl)
        if pg == 0:
            #print("No customer reviewed the product!")
            driver.quit()
            return None
        else:
            #totalPg = totalRev // 10 + 1
            #print("\nTotal number of reviews available ",totalRev)
            #print("Total number of pages ",totalPg)
            #pg = int(input("Enter the number of pages you want to scrape (1 page will
have 10 reviews) : "))
            for i in range(1, pg + 1):
                if keyboard.is_pressed('q'):
                    print("Stopping scraping...")
                    break
                try:
                    reviewUrl = productUrl.replace("dp", "product-reviews") +
"&pageNumber=" + str(i)
                    extractReviews(driver, reviewUrl, i)
                except Exception as e:
                    print(f"Error scraping page {i}: {e}")
    except Exception as e:
        driver.quit()
        raise RuntimeError("Internal Error. ",e)
    finally:
        driver.quit()
    try:
        if reviewlist == []:
            raise RuntimeError("No data is collected")
        else:
            df = pd.DataFrame(reviewlist)
            dir = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data"
            prodName = int(get_max_numbered_csv(dir).replace(".csv",""))
            prodName = prodName + 1
            file_path = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data\\" +
str(prodName) + ".csv"
            df.to_csv(file_path)
            print("Reviews are successfully collected and saved as ", prodName ,".csv")
    except:
```

```
        raise RuntimeError("Unable to create CSV")

#test
#get_review("https://www.amazon.in/HP-i5-12450H-15-6-inch-Response-fa0666TX/dp/B0C2HZYM87/
ref=cm_cr_arp_d_pdt_img_top?ie=UTF8&th=1",5)
```

flipkartProductReviewPartial.py

```
import requests
import pandas as pd
from bs4 import BeautifulSoup
from .maxFile import get_max_numbered_csv

review_data = []

def extractReviews(reviewUrl):
    try:
        resp = requests.get(reviewUrl)
        soup = BeautifulSoup(resp.text, 'html.parser')
        review_divs = soup.find_all('div', {"class":"col EPCmJX Ma1fCG"})
        for div in review_divs:
            rating = None
            title = None
            review_description = None

            rating_elem = div.find('div', {"class":"XQDdHH Ga3i8K"})
            if rating_elem:
                rating = rating_elem.text.strip()

            title_elem = div.find('p', {"class":"z9E0IG"})
            if title_elem:
                title = title_elem.text.strip()

            review_desc_elem = div.find('div', {"class":"ZmyHeo"})
            if review_desc_elem:
                review_description = review_desc_elem.text.strip()

            if rating and title and review_description:
                review_data.append({
                    'Rating': rating,
                    'Review Title': title,
                    'Review Body': review_description[:-9]
                })
    except Exception as e:
        raise RuntimeError("Unable to scrape reviews")
```

```
def get_review(prod, totalPg):
    try:
        productUrl = prod
        reviewUrl = productUrl.replace("/p/", "/product-reviews/")

        #print("Number of reviews found ",totalRev)
        #print("Number of review pages : ",totalPg)

        totalPg = int(totalPg)
        #temp = int(input("Enter the number of review pages you want to scrape (1 page has
10 reviews) : "))
        for i in range(totalPg):
            try:
                reviewUrl = reviewUrl + "&page=" + str(i+1)
                extractReviews(reviewUrl)
            except Exception as e:
                print("Error occured: ",e)
    except Exception as e:
        raise RuntimeError("Internal Error. ",e)

    try:
        df = pd.DataFrame(review_data)
        dir = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data"
        prodName = int(get_max_numbered_csv(dir).replace(".csv",""))
        prodName = prodName + 1
        file_path = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data\\" +
str(prodName) + ".csv"
        df.to_csv(file_path)
        print("Reviews are successfully collected and saved as ", str(prodName) ,".csv")
    except:
        raise RuntimeError("Unable to create CSV")

# test run
#get_review("https://www.flipkart.com/motorola-envisionx-165-cm-65-inch-qled-ultra-hd-4k-s
mart-google-tv-dolby-vision-atmos/product-reviews/itm767828569c629?pid=TVSGRTDDGYA4HPWU&li
d=LSTTVSGRTDDGYA4HPWUZVEWYT&marketplace=FLIPKART",2)
```

productDetailGeneral.py

```
from bs4 import BeautifulSoup
from selenium.webdriver.chrome.options import Options
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

prodDetail = {}
```

```
def getSauce(prodLink):
    try:
        chrome_options = Options()
        chrome_options.add_argument('--headless')
        chrome_options.add_argument('--disable-gpu')
        chrome_options.add_argument("--silent")
        chrome_options.add_argument('--log-level=3')
        driver = webdriver.Chrome(options=chrome_options)
        driver.get("https://pricehistory.app")
        search_bar = driver.find_element("id","search")
        product_url = prodLink
        search_bar.send_keys(product_url)
        search_bar.send_keys(Keys.RETURN)
        time.sleep(7)
        html_content = driver.page_source
        driver.quit()
        return html_content
    except:
        driver.quit()
        raise Exception('Unable to get info in pricehistory.com')
    finally:
        driver.quit()

def getTitle(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) >
div.col-md-12.col-lg-7.col-xl-8 > div >
div.col-lg-9.col-md-9.col-sm-8.col-8.ph-title.my-0.pl-2.p-1 > h1')
        prodDetail['Title'] = t.text
    except:
        prodDetail['Title'] = " "

def getHighPrice(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) >
div.col-12.px-0.py-2.all-time-price-overview.small > div > div.col.bg-danger.text-light >
span.amount')
        prodDetail['Highest Price'] = t.text
    except:
        prodDetail['Highest Price'] = " "

def getHighPriceDate(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) > div:nth-child(9) > table >
tbody > tr:nth-child(2) > td')
        prodDetail['Date of highest price'] = t.text.strip()
```

```
    except:
        prodDetail['Date of highest price'] = " "

def getAvgPrice(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) >
div.col-12.px-0.py-2.all-time-price-overview.small > div > div.col.bg-warning.text-dark >
span.amount')
        prodDetail['Average Price'] = t.text.strip()
    except:
        prodDetail['Average Price'] = " "

def getAvgPriceDate(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) > div:nth-child(9) > table >
tbody > tr:nth-child(4) > td')
        prodDetail['Average Price as of'] = t.text.strip()
    except:
        prodDetail['Average Price as of'] = " "

def getLowPrice(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) >
div.col-12.px-0.py-2.all-time-price-overview.small > div > div.col.bg-info.text-light >
span.amount')
        prodDetail['Lowest Price'] = t.text.strip()
    except:
        prodDetail['Lowest Price'] = " "

def getLowPriceDate(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) >
div.col-12.px-0.py-2.all-time-price-overview.small > div > div.col.bg-info.text-light >
span.amount')
        prodDetail['Date of lowest price'] = t.text
    except:
        prodDetail['Date of lowest price'] = " "

def getCurrentPrice(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) >
div.col-md-12.col-lg-5.col-xl-4.ph-pricing.mt-2.mb-2.border.shadow-sm.p-2.bg-light >
div.ph-pricing-pricing')
        prodDetail['Current Price'] = t.text
    except:
        prodDetail['Current Price'] = " "
```

```
def getLabelPrice(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) >
div.col-md-12.col-lg-5.col-xl-4.ph-pricing.mt-2.mb-2.border.shadow-sm.p-2.bg-light >
div.ph-pricing-mrp')
        prodDetail['Label Price'] = t.text
    except:
        prodDetail['Label Price'] = " "

def getDiscount(soup):
    try:
        t = soup.select_one('body > div > div:nth-child(6) >
div.col-md-12.col-lg-5.col-xl-4.ph-pricing.mt-2.mb-2.border.shadow-sm.p-2.bg-light >
div.ph-pricing-discount')
        prodDetail['Discount'] = t.text.strip()
    except:
        prodDetail['Discount'] = " "

def getSuggestion(soup):
    try:
        t = soup.select_one('#pricing-assessment > div > div >
div.col-12.col-sm-12.col-md-6.m-0 > p')
        prodDetail['Suggestion'] = t.text.strip()
    except:
        prodDetail['Suggestion'] = " "

def getStatus(soup):
    try:
        t = soup.select_one('#fix-bottom > div > div > div >
div.col-3.p-0.rounded.text-light > a')
        prodDetail['Status'] = t.text
    except:
        prodDetail['Status'] = " "

def ext(prodUrl):
    try:
        html = getSauce(prodUrl)
        if html == 0:
            print("Unable to retrieve info")
            return 0
        else:
            soup = BeautifulSoup(html, 'html.parser')
            getTitle(soup)
            getLowPrice(soup)
            getLowPriceDate(soup)
            getHighPrice(soup)
            getHighPriceDate(soup)
```

```
            getAvgPrice(soup)
            getAvgPriceDate(soup)
            getCurrentPrice(soup)
            getLabelPrice(soup)
            getDiscount(soup)
            getStatus(soup)
            getSuggestion(soup)
            if prodDetail['Current Price'] != " " and prodDetail['Label Price'] != " ":
                t1 = prodDetail['Current Price'].replace("₹","")
                t1 = t1.replace(",","")
                t2 = prodDetail['Label Price'].replace("₹","")
                t2 = t2.replace(",","")
                prodDetail['Savings'] = int(t2) - int(t1)
                prodDetail['Savings'] = "₹" + str(prodDetail['Savings'])
            else:
                prodDetail['Savings'] = " "
            return prodDetail
    except:
        raise RuntimeError('Unable to get info (General)')

#di =
ext("https://www.amazon.in/Godrej-Convertible-Split-AC-12TINV3R32-GWA/dp/B0BN37ZCF7/ref=sr
_1_1_sspa?_encoding=UTF8&content-id=amzn1.sym.58c90a12-100b-4a2f-8e15-7c06f1abe2be&dib=eyJ
2IjoiMSJ9.LpujZ4uISPUK8sa_6yNGVaJ6gZ1SvJMqMGwoZI950ZumxSLryiO68ly2lNtjsEb1U80Wn8Xhe33o4hK4
pCTyyBhL8ne8M9UXIluv2LUxkGvqAbW9MgYkVpvDw4vooMxRTYsy_KXPAaEkmt5u8sZF0yQFscrCJ0ejsc1wzwQB1Z
yAASrLLxJDK1kDPpDdtI7_lFn_SFV2kiAtumeiOuk9Tc0zlGEzOaQJOPcMJCt0vTwrHq72MToJ1uU4FZFkqhdY9gXM
U-I9l8d4qIFFJNjJTEwgsBhzO4o4ghxLWujsEmg.fHCIcJ7Ojj7HZNzWq2_RzGgOFqKo4jRkxQlGJnRyCu4&dib_ta
g=se&pd_rd_r=7090cfc8-3768-4ef4-a5a9-1ea7bb92d35b&pd_rd_w=1WmFB&pd_rd_wg=xPWBn&pf_rd_p=58c
90a12-100b-4a2f-8e15-7c06f1abe2be&pf_rd_r=KFZ13V5H5K4JFJ0RG5Z6&qid=1712186662&refinements=
p_85%3A10440599031&rps=1&s=kitchen&sr=1-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGZfYnJvd3Nl&th
=1")
#di =
ext("https://www.amazon.in/Neemans-Casual-Trainers-Comfortable-Lightweight/dp/B0BY6GCZVS/r
ef=lp_1983518031_1_1_sspa?keywords=Men%27s+Shoes&pf_rd_p=9e034799-55e2-4ab2-b0d0-eb42f95b2
d05&pf_rd_r=8K2BSATTTRS47BM0B8EB&sp_csd=d2lkZ2V0TmFtZT1zcF9hcGJfZGVza3RvcF9icm93c2VfaW5saW
5lX2F0Zg&psc=1")
#print(di)
```

productDetailAmazon.py

```
import requests
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.support import expected_conditions as EC
```

```python
from .maxFile import get_max_numbered_jpg

prodDetail = {}

def getTitle(soup):
    try:
        t = soup.select_one('#productTitle')
        prodDetail['Title'] = t.text
    except:
        prodDetail['Title'] = " "

def getCurrentPrice(soup):
    try:
        symbol = soup.select_one('#corePriceDisplay_desktop_feature_div > div.a-section.a-spacing-none.aok-align-center.aok-relative > span.a-price.aok-align-center.reinventPricePriceToPayMargin.priceToPay > span:nth-child(2) > span.a-price-symbol').text
        t = soup.select_one('#corePriceDisplay_desktop_feature_div > div.a-section.a-spacing-none.aok-align-center.aok-relative > span.a-price.aok-align-center.reinventPricePriceToPayMargin.priceToPay > span:nth-child(2) > span.a-price-whole')
        prodDetail['Current Price'] = symbol + t.text
    except:
        prodDetail['Current Price'] = " "

def getInfo(soup):
    try:
        list_items = soup.select_one('#feature-bullets > ul')
        list_items_text = [item.get_text(strip=True) for item in list_items]
        filtered_list = [item for item in list_items_text if item]
        prodDetail['About'] = filtered_list
    except:
        prodDetail['About'] = " "

def amazonAI(soup):
    try:
        t = soup.select_one('#product-summary > p.a-spacing-small > span')
        prodDetail['Amazon AI'] = t.text
    except:
        prodDetail['Amazon AI'] = " "

def getImage(soup):
    try:
        image_tag = soup.select_one('#landingImage')
        image_url = image_tag['src']
        dir = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image"
        img_name = int(get_max_numbered_jpg(dir).replace(".jpg", ""))
```

```
            img_name += 1
            img_name = str(img_name)
            response = requests.get(image_url)
            path = f"C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image\\{img_name}.jpg"
            with open(path, 'wb') as f:
                f.write(response.content)
        except Exception as e:
            dir = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image"
            print(f"Error getting image: {e}")
            img_name = int(get_max_numbered_jpg(dir).replace(".jpg", ""))
            img_name += 1
            img_name = str(img_name)
            path = f"C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image\\{img_name}.jpg"
            with open(path, 'wb') as f:
                f.write(b'Dummy content')

    def get_html(productUrl, driver):
        try:
            driver.get(productUrl)
            wait = WebDriverWait(driver, 10)
            wait.until(EC.presence_of_all_elements_located((By.XPATH, '//*')))
            html_content = driver.page_source
            driver.quit()
            return html_content
        except:
            driver.quit()
            return None

    def get_data(productUrl,driver):
        try:
            html = get_html(productUrl,driver)
            soup = BeautifulSoup(html, 'html.parser')
            getTitle(soup)
            getCurrentPrice(soup)
            getInfo(soup)
            amazonAI(soup)
            getImage(soup)
        except:
            raise RuntimeError("Unable to get data")

    def ext(prod):
        try:
            chrome_options = Options()
            chrome_options.add_argument('--headless')
            chrome_options.add_argument('--disable-gpu')
            chrome_options.add_argument("--silent")
            chrome_options.add_argument('--log-level=3')
```

```
        driver = webdriver.Chrome(options=chrome_options)
        get_data(prod,driver)
        return prodDetail
    except:
        raise RuntimeError("Unable to get product info (Amazon)")

# test run
#ext("https://www.amazon.in/Xiaomi-inches-Vision-Google-L50M8-A2IN/dp/B0CH31C1BR/ref=sr_1_1_sspa?dib=eyJ2IjoiMSJ9.wJOzhbBKpQgolxe4lGBuoCUrXbxzpMOAxzXJIgwJvEUuXumWdDpTdCD0nXp6FxNolHVI8fbPwH8KlijHTFHE1sV9EAZl27_KskBpE3Llyk8DCp1Bb9_cysi1laJ_x50HmhIIq8N7-UUHb9RPxs-DAL4IySSXaQgfHmlmRLLIuFNCoC2sl-9O0zvUiQ3Z_S-X09n8a2XkzChMKj57FbanKXBVdc2xO7Oy0Q6LqUhpbr8.zwyZfZxZONuaFCZ27DCqxzunhBJS2thPW08dmTcEkTg&dib_tag=se&keywords=tv&qid=1712837408&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&psc=1")
```

productDetailFlipkart.py

```
import requests
from bs4 import BeautifulSoup
from .maxFile import get_max_numbered_jpg

prodDetail = {}

def getTitle(soup):
    try:
        t = soup.find('span',{'class': 'VU-ZEz'})
        prodDetail['Title'] = t.text
    except:
        prodDetail['Title'] = " "

def getCurrentPrice(soup):
    try:
        t = soup.find('div',{'class': 'Nx9bqj CxhGGd'})
        prodDetail['Current Price'] = t.text
    except:
        prodDetail['Current Price'] = " "

def getInfo(soup):
    try:
        t = soup.find_all('li',{'class': '_7eSDEz'})
        t_ = []
        for i in t:
            t_.append(i.text)
        prodDetail['About'] = t_
    except:
        prodDetail['About'] = " "

def getImage(soup):
```

```
    try:
        image_tag = soup.find("div",{"class":"_4WELSP _6lpKCl"})
        image_tag = soup.find_all('img')
        image_url = image_tag[2]['src']
        dir = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image"
        img_name = int(get_max_numbered_jpg(dir).replace(".jpg", ""))
        img_name += 1
        img_name = str(img_name)
        headers = {
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36',
            'Accept': '*/*'
        }
        response = requests.get(image_url, headers=headers)
        path = f"C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image\\{img_name}.jpg"
        with open(path, 'wb') as f:
            f.write(response.content)
    except Exception as e:
        dir = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image"
        print(f"Error getting image: {e}")
        img_name = int(get_max_numbered_jpg(dir).replace(".jpg", ""))
        img_name += 1
        img_name = str(img_name)
        path = f"C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image\\{img_name}.jpg"
        with open(path, 'wb') as f:
            f.write(b'Dummy content')

def get_html(productUrl):
    try:
        resp = requests.get(productUrl)
        html_content = resp.content
        return html_content
    except:
        return None

def get_data(productUrl):
    try:
        html = get_html(productUrl)
        soup = BeautifulSoup(html, 'html.parser')
        getTitle(soup)
        getCurrentPrice(soup)
        getInfo(soup)
        getImage(soup)
    except:
        raise RuntimeError("Unable to get data")

def ext(prod):
```

```
    try:
        get_data(prod)
        return prodDetail
    except:
        raise RuntimeError("Unable to get product info (Flipkart)")

# test run
#print(ext("https://www.flipkart.com/philips-gc1903-1300-w-steam-iron/p/itma2af216e592da?pid=IRND2UFFYBMGSPUN&lid=LSTIRND2UFFYBMGSPUNG4PV8Q&marketplace=FLIPKART&store=j9e%2Fabm%2Fa0u&srno=b_1_2&otracker=nmenu_sub_Appliances_0_Irons&fm=organic&iid=ffd3abce-9258-4262-926f-c8737df61ddc.IRND2UFFYBMGSPUN.SEARCH&ppt=browse&ppn=browse&ssid=afxrdhlakg0000001714546159013"))
```

SentimentClassification and Summarizer Python files

classifier.py

```
import pandas as pd
import numpy as np
import tensorflow as tf
import torch
import re
import string
import pickle
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from keras.layers import LSTM
from keras.initializers import Orthogonal
import os
import sys
from transformers import pipeline, AutoTokenizer, AutoModelForSequenceClassification
import warnings
warnings.filterwarnings('ignore')

pos1 = 0
pos2 = 0
pos3 = 0

neg1 = 0
neg2 = 0
neg3 = 0

# Ensure NLTK stopwords are downloaded
#nltk.download('stopwords')
```

```
#nltk.download('punkt')

#TF_ENABLE_ONEDNN_OPTS=0

def get_max_numbered_csv(directory):
    files = os.listdir(directory)
    max_number = float('-inf')
    max_filename = None
    for file_name in files:
        if file_name.endswith('.csv'):
            try:
                file_number = int(os.path.splitext(file_name)[0])
                if file_number > max_number:
                    max_number = file_number
                    max_filename = file_name
            except ValueError:
                continue
    return max_filename

def normalize_text(text):
    stemmer = SnowballStemmer("english")
    normalized_text = []
    for word in text.split():
        stemmed_word = stemmer.stem(word)
        normalized_text.append(stemmed_word)
    return ' '.join(normalized_text)

def remove_stopwords(texto):
    stop_words = set(stopwords.words('english'))
    tokens = nltk.word_tokenize(texto.lower())
    return " ".join([token for token in tokens if token not in stop_words])

def clean_text(text):
    try:
        text = re.sub(r'[^A-Za-zÀ-ú ]+', '', text)
        text = re.sub('book|one', '', text)
        text = text.lower()
        text = text.translate(str.maketrans('', '', string.punctuation))
        text = re.sub(r'\s+', ' ', text).strip()
        return text
    except:
        return text

def rameez(z):
    try:
        z['Rating'] = z['Rating'].astype(str)
        z['Rating'] = z['Rating'].str.extract(r'(\d+\.\d+)').astype(float)
```

```
        z['Label'] = z['Rating'].apply(lambda x: 1 if x >= 3 else 0)
        z['Combined'] = z['Review Title'].apply(str) + ' ' + z['Review Body']
        return z
    except:
        z['Rating'] = z['Rating'].astype(str)

        # Extract numeric ratings from the 'Rating' column
        z['Rating'] = z['Rating'].str.extract(r'(\d+\.\d+)').astype(float)

        # Apply the label based on the extracted ratings
        z['Label'] = z['Rating'].apply(lambda x: 1 if x >= 3 else 0)

        # Combine 'Review Title' and 'Review Body' into a single column
        z['Combined'] = z['Review Title'].apply(str) + ' ' + z['Review Body']

        return z
    """
    if kind == "amazon":
        z['rating_value'] = z['Rating'].str.extract(r'(\d+\.\d+)').astype(float)
        z['label'] = z['rating_value'].apply(lambda x: 1 if x >= 3 else 0)
        z.drop(columns=["Unnamed: 0","Rating","rating_value",'productTitle'],
inplace=True)
        z.columns = ['title', 'text','label_on_rating']
        z['text'] = z['text'].apply(str) + ' ' + z['title'].apply(str)
        z.drop('title', axis = 1, inplace = True)
        return z
    elif kind == "flipkart":
        z['rating_value'] = z['Rating'].str.extract(r'(\d+\.\d+)').astype(float)
        z['label'] = z['rating_value'].apply(lambda x: 1 if x >= 3 else 0)
        z.drop(columns=["Rating","rating_value"], inplace=True)
        z.columns = ['title', 'text','label_on_rating']
        z['text'] = z['text'].apply(str) + ' ' + z['title'].apply(str)
        z.drop('title', axis = 1, inplace = True)
        return z
    """

def clean2(a):
    a['Combined'] = a['Combined'].apply(clean_text)
    a['Combined'] = a['Combined'].apply(remove_stopwords)
    a['Combined'] = a['Combined'].apply(normalize_text)
    return a

def predict_and_evaluate(data, max_len=100):
    global pos1
    global neg1
    with open(r"C:\Users\ramee\Desktop\AI
Lab\Project\UI\SentimentSummarizer\models\Tokenizer.h5", 'rb') as handle:
```

```
        tokenizer = pickle.load(handle)

    tokenizer.fit_on_texts(data['Combined'])
    sequences = tokenizer.texts_to_sequences(data['Combined'])
    word_index = tokenizer.word_index

    sequences_padded = pad_sequences(sequences, maxlen=max_len)

    model = load_model(r"C:\Users\ramee\Desktop\AI
Lab\Project\UI\SentimentSummarizer\models\CNN.h5")
    predictions = model.predict(sequences_padded)

    """model_name = "LYTinn/lstm-finetuning-sentiment-model-3000-samples"
    tokenizer2 = AutoTokenizer.from_pretrained(model_name)
    model2 = AutoModelForSequenceClassification.from_pretrained(model_name)
    sentiment_pipeline = pipeline("text-classification", model=model2,
tokenizer=tokenizer2)"""

    """predictions2 = sentiment_pipeline(data['Combined'].tolist())
    predicted_labels2 = [pred['label'] for pred in predictions2]
    #print(predictions2)
    pos2 = predicted_labels2.count('positive')
    neg2 = predicted_labels2.count('negative')"""



    #model2 = load_model(r"C:\Users\ramee\Desktop\AI Lab\Project\UI\Sentiment Classifier
and Summarizer\models\LSTM.h5", custom_objects=custom_objects)
    #predictions2 = model2.predict(sequences_padded)

    def get_label(prediction):
        return 'positive' if prediction >= 0.5 else 'negative'

    predicted_labels = [get_label(prediction) for prediction in predictions]

    #predicted_labels2 = [get_label(prediction) for prediction in predictions2]

    data['predicted_label'] = predicted_labels
    #data['predicted_label2'] = predicted_labels2

    positive_count = sum(1 for label in predicted_labels if label == "positive")
    negative_count = sum(1 for label in predicted_labels if label == "negative")

    #positive_count2 = sum(1 for label in predicted_labels2 if label == "positive")
    #negative_count2 = sum(1 for label in predicted_labels2 if label == "negative")

    #print("The Model's Predicted Positive count: ",positive_count)
```

```
    #print("The Model's Predicted Negative count: ",negative_count)

    #correct_count = 0
    #wrong_count = 0

    #for org_label, pred_label in zip(data['label_on_rating'], data['predicted_label']):
    #    if (org_label == 1 and pred_label == 'positive') or (org_label == 0 and
pred_label == 'negative'):
    #        correct_count += 1
    #    else:
    #        wrong_count += 1

    pos1 = positive_count
    neg1 = negative_count

"""def trans(data):
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    print(data.columns)
    encoded_texts = [tokenizer.encode_plus(text, max_length=100, padding='max_length',
truncation=True, return_tensors='tf') for text in data['Combined']]

    # Extract input IDs from the encoded texts and remove the extra dimension
    sequences_padded = np.array([encoded_text['input_ids'].numpy().squeeze() for
encoded_text in encoded_texts])

    # Load the trained model
    model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased')

    # Make predictions on the padded sequences
    outputs = model.predict(sequences_padded)
    predictions = tf.nn.softmax(outputs.logits, axis=-1).numpy()
    print(predictions)
    # Determine the predicted labels
    predicted_labels = ['positive' if prediction[1] >= 0.5 else 'negative' for prediction
in predictions]
    #data['predicted_labels']=predicted_labels
    # Evaluate the model's performance
    positive_count = sum(1 for label in predicted_labels if label == "positive")
    negative_count = sum(1 for label in predicted_labels if label == "negative")
    print("The Model's Predicted Positive count:", positive_count)
    print("The Model's Predicted Negative count:", negative_count)

    correct_count = 0
    wrong_count = 0
    for org_label, pred_label in zip(data['label_on_rating'], predicted_labels):
        if (org_label == 1 and pred_label == 'positive') or (org_label == 0 and pred_label
== 'negative'):
```

```
            correct_count += 1
        else:
            wrong_count += 1

    #accuracy = correct_count / len(data)
    #print("Accuracy:", accuracy)
    """

def trans2(data):
    global pos3
    global neg3
    global pos2
    global neg2
    # Load BERT tokenizer and model
    tokenizer = AutoTokenizer.from_pretrained("MarieAngeA13/Sentiment-Analysis-BERT")
    model =
AutoModelForSequenceClassification.from_pretrained("MarieAngeA13/Sentiment-Analysis-BERT")

    tokenizer2 = AutoTokenizer.from_pretrained("Dmyadav2001/Sentimental-Analysis")
    model2 =
AutoModelForSequenceClassification.from_pretrained("Dmyadav2001/Sentimental-Analysis")

    sentiment_analysis2 = pipeline("text-classification", model=model2,
tokenizer=tokenizer2)


    # Initialize sentiment analysis pipeline
    sentiment_analysis = pipeline("text-classification", model=model, tokenizer=tokenizer)

    # Convert Pandas Series to list
    combined_texts = data['Combined'].tolist()

    predictions2 = sentiment_analysis2(combined_texts)

    # Predict sentiment and count positive/negative labels
    predictions = sentiment_analysis(combined_texts)
    #print(predictions)
    predicted_labels = [pred['label'] for pred in predictions]
    temp = []
    for pred in predictions:
        if pred['label'] == 'positive':
            temp.append("positive")
        else:
            temp.append("negative")
    data['Sentiment'] = temp
    pos3 = predicted_labels.count('positive')
    neg3 = predicted_labels.count('negative')
```

```
    pos2 = sum(1 for pred in predictions2 if  pred['label'] == 'LABEL_1')
    neg21 = sum(1 for pred in predictions2 if pred['label'] == 'LABEL_0')
    neg22 = sum(1 for pred in predictions2 if pred['label'] == 'LABEL_2')
    neg2=neg21+neg22


def perform_classification():
    dir = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data"
    file = get_max_numbered_csv(dir)
    dir = dir + "\\" + file
    df = pd.read_csv(dir)
    a=rameez(df)
    a=clean2(a)
    trans2(a)
    df.to_csv("C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data
Classified\\"+file)
    predict_and_evaluate(a)

#a.to_csv("C:\\Users\\praga\\Desktop\\aiproject\\Predicted_CSV\\Modified1030.csv",index=Fa
lse)
#modified_df =
pd.read_csv("C:\\Users\\praga\\Desktop\\aiproject\\Predicted_CSV\\Modified1030.csv")
#predicted_labels = modified_df['predicted_label']
#original_df = pd.read_csv(dir)
#original_df['predicted_label'] = predicted_labels
#original_df.to_csv(dir, index=False)

##############################
#perform_classification()
#print(pos3)
#print(neg3)
#print(pos2)
#print(neg2)
#print(pos1)
#print(neg1)
##############################
```

summarizer.py

```
from transformers import BartForConditionalGeneration, BartTokenizer
import pandas as pd
import os
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
```

```
from nltk.probability import FreqDist
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.tokenize import word_tokenize
from nltk.cluster.util import cosine_distance
import numpy as np
import google.generativeai as genai

genai_api = "AIzaSyA44RXOJCpwf1OTmxF_PDQOc0OCffcJlX4"

genai.configure(api_key=genai_api)


positive_summary = None
negative_summary = None
e_positive_summary = None
e_negative_summary = None

bart_model = BartForConditionalGeneration.from_pretrained('facebook/bart-large-cnn')
bart_tokenizer = BartTokenizer.from_pretrained('facebook/bart-large-cnn')

def preprocess_text(sentence):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(sentence.lower())
    return [word for word in words if word.isalnum() and word not in stop_words]

def sentence_similarity(sent1, sent2):
    words1 = preprocess_text(sent1)
    words2 = preprocess_text(sent2)
    all_words = list(set(words1 + words2))
    vector1 = [1 if word in words1 else 0 for word in all_words]
    vector2 = [1 if word in words2 else 0 for word in all_words]
    return 1 - cosine_distance(vector1, vector2)

def build_similarity_matrix(sentences):
    similarity_matrix = np.zeros((len(sentences), len(sentences)))
    for i in range(len(sentences)):
        for j in range(len(sentences)):
            if i != j:
                similarity_matrix[i][j] = sentence_similarity(sentences[i], sentences[j])
    return similarity_matrix

def generate_summary(text, num_sentences):
    sentences = sent_tokenize(text)
    sentence_similarity_matrix = build_similarity_matrix(sentences)
    scores = np.sum(sentence_similarity_matrix, axis=1)
    ranked_sentences = sorted(((scores[i], sentence) for i, sentence in
enumerate(sentences)), reverse=True)
```

```
    summary = ' '.join([sentence for score, sentence in ranked_sentences[:num_sentences]])
    return summary



def get_max_numbered_csv(directory):
    files = os.listdir(directory)
    max_number = float('-inf')
    max_filename = None
    for file_name in files:
        if file_name.endswith('.csv'):
            try:
                file_number = int(os.path.splitext(file_name)[0])
                if file_number > max_number:
                    max_number = file_number
                    max_filename = file_name
            except ValueError:
                continue
    return max_filename

def summarize_text(text, model, tokenizer, max_length=200, min_length=40):
    inputs = tokenizer([text], max_length=1024, truncation=True, return_tensors='pt')
    summary_ids = model.generate(inputs['input_ids'], max_length=max_length,
min_length=min_length, length_penalty=2.0, num_beams=4, early_stopping=True)
    summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    return summary

def perform_summarize_on_df():
    global positive_summary
    global negative_summary
    global e_positive_summary
    global e_negative_summary

    dir = get_max_numbered_csv("C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data
Classified")
    df = pd.read_csv("C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data
Classified\\"+ dir)

    positive_reviews = " ".join(df[df['Sentiment'] == 'positive']['Review Body'])
    negative_reviews = " ".join(df[df['Sentiment'] == 'negative']['Review Body'])

    model = genai.GenerativeModel('gemini-1.5-flash')

    positive_summary = summarize_text(positive_reviews, bart_model, bart_tokenizer)
    #print("\nPositive Reviews Summary:\n", positive_summary)

    # Summarize negative reviews
```

```
    negative_summary = summarize_text(negative_reviews, bart_model, bart_tokenizer)
    #print("\nNegative Reviews Summary:\n", negative_summary)

    response1 = model.generate_content("Here is an abstractive summary of a product's
review now modify it so that it reads out like third person view"+positive_summary)
    positive_summary = response1.text

    response2 = model.generate_content("Here is an abstractive summary of a product's
review now modify it so that it reads out like third person view"+negative_summary)
    negative_summary = response2.text

    e_negative_summary = generate_summary(negative_reviews, 2)
    e_positive_summary = generate_summary(positive_reviews, 2)


############################
#perform_summarize_on_df()
#print(positive_summary)
#print(negative_summary)
#print(e_negative_summary)
#print(e_positive_summary)
############################
```

Other Python Files

wordCloud.py

```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import os

d1 = None
d2 = None
d3 = None

def get_max_numbered_csv(directory):
    files = os.listdir(directory)
    max_number = float('-inf')
    max_filename = None
    for file_name in files:
        if file_name.endswith('.csv'):
            try:
                file_number = int(os.path.splitext(file_name)[0])
                if file_number > max_number:
                    max_number = file_number
```

```
                    max_filename = file_name
            except ValueError:
                continue
    return max_filename

def get_max_numbered_jpg(directory):
    files = os.listdir(directory)
    max_number = float('-inf')
    max_filename = None
    for file_name in files:
        if file_name.endswith('.jpg'):
            try:
                file_number = int(os.path.splitext(file_name)[0])
                if file_number > max_number:
                    max_number = file_number
                    max_filename = file_name
            except ValueError:
                continue
    return max_number

def create_wordcloud():
    global d1
    global d2
    global d3

    dir = get_max_numbered_csv("C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data
Classified")
    df = pd.read_csv("C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Scraped Data
Classified\\"+ dir)

    dir_img = get_max_numbered_jpg("C:\\Users\\ramee\\Desktop\\AI
Lab\\Project\\UI\\static\\Images\\Wordcloud1")

    d1 = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\UI\\static\\Images\\Wordcloud1\\"
    d2 = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\UI\static\\Images\\Wordcloud2\\"
    d3 = "C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\UI\\static\\Images\\Wordcloud3\\"

    text = ' '.join(df['Review Body'].dropna())
    text1 = ' '.join(df[df['Sentiment'] == 'negative']['Review Body'].dropna())
    text2 = ' '.join(df[df['Sentiment'] == 'positive']['Review Body'].dropna())

    if len(text1) == 0:
        text1 = "nothing nothing"
    if len(text2) == 0:
        text2 = "nothing nothing"
    if len(text) == 0:
        text = "nothin nothin"
```

```
    wordcloud = WordCloud(width=800, height=400, background_color='rgba(255, 255, 255, 0)').generate(text)
    wordcloud2 = WordCloud(width=800, height=400, background_color='rgba(255, 255, 255, 0)').generate(text1)
    wordcloud3 = WordCloud(width=800, height=400, background_color='rgba(255, 255, 255, 0)').generate(text2)

    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title("Overall Body Word Cloud")
    d1 = d1 + str(dir_img+1) + ".jpg"
    plt.savefig(d1)  # Save wordcloud image to d1 directory

    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud2, interpolation='bilinear')
    plt.axis('off')
    plt.title("Negative Body Word Cloud")
    d2 = d2 + str(dir_img+1) + ".jpg"
    plt.savefig(d2)  # Save wordcloud1 image to d2 directory

    plt.figure(figsize=(10, 5))
    plt.imshow(wordcloud3, interpolation='bilinear')
    plt.axis('off')
    plt.title("Positive Body Word Cloud")
    d3 = d3 + str(dir_img+1) + ".jpg"
    plt.savefig(d3)  # Save wordcloud2 image to d3 directory

#create_wordcloud()
```

backgroundRemover.py

```
import cv2
import os

def get_max_numbered_jpg(directory):
    files = os.listdir(directory)
    max_number = float('-inf')
    max_filename = None
    for file_name in files:
        if file_name.endswith('.jpg'):
            try:
                file_number = int(os.path.splitext(file_name)[0])
                if file_number > max_number:
                    max_number = file_number
```

```
                    max_filename = file_name
            except ValueError:
                continue
    return max_filename

def remove_background():
    dir = 'C:\\Users\\ramee\\Desktop\\AI Lab\\Project\\Image\\'
    fileName = get_max_numbered_jpg(dir)
    image = cv2.imread(dir+"\\"+fileName)

    # Convert the image to BGRA
    image_bgra = cv2.cvtColor(image, cv2.COLOR_BGR2BGRA)

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    _, alpha = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY)

    # Invert the mask
    alpha = cv2.bitwise_not(alpha)

    # Add the new alpha channel to the image
    image_bgra[:, :, 3] = alpha

    # Save the result
    cv2.imwrite(dir+"\\"+fileName.replace('.jpg', '.png'), image_bgra)

#remove_background()
```

maxFile.py

```
import os

def get_max_numbered_csv(directory):
    files = os.listdir(directory)
    max_number = float('-inf')
    max_filename = None
    for file_name in files:
        if file_name.endswith('.csv'):
            try:
                file_number = int(os.path.splitext(file_name)[0])
                if file_number > max_number:
                    max_number = file_number
                    max_filename = file_name
            except ValueError:
                continue
    return max_filename
```

```python
def get_max_numbered_jpg(directory):
    files = os.listdir(directory)
    max_number = float('-inf')
    max_filename = None
    for file_name in files:
        if file_name.endswith('.jpg'):
            try:
                file_number = int(os.path.splitext(file_name)[0])
                if file_number > max_number:
                    max_number = file_number
                    max_filename = file_name
            except ValueError:
                continue
    return max_filename

def get_max_numbered_png(directory):
    files = os.listdir(directory)
    max_number = float('-inf')
    max_filename = None
    for file_name in files:
        if file_name.endswith('.png'):
            try:
                file_number = int(os.path.splitext(file_name)[0])
                if file_number > max_number:
                    max_number = file_number
                    max_filename = file_name
            except ValueError:
                continue
    return max_filename
```

MySQL Commands

ProductDetailsTwo.sql

```sql
CREATE TABLE Products (
    `Product ID` INT AUTO_INCREMENT PRIMARY KEY,
    `Name` VARCHAR(40),
    `Date` DATE,
    `ImagePath` VARCHAR(255),
    `ImagePath1` VARCHAR(255),
    `ImagePath2` VARCHAR(255),
    `ImagePath3` VARCHAR(255),
    `origin` VARCHAR(50)
);
```

```
CREATE TABLE PricingDetails (
  `Product ID` INT NOT NULL,
  `Title` VARCHAR(255) NOT NULL,
  `Highest Price` VARCHAR(20) NOT NULL,
  `Date of Highest Price` VARCHAR(100) NOT NULL,
  `Lowest Price` VARCHAR(20) NOT NULL,
  `Date of Lowest Price` VARCHAR(100) NOT NULL,
  `Average Price` VARCHAR(20) NOT NULL,
  `Date of Average Price` VARCHAR(100) NOT NULL,
  `Current Price` VARCHAR(20) NOT NULL,
  `Label Price` VARCHAR(20) NOT NULL,
  `Savings` VARCHAR(20) NOT NULL,
  `Discount` VARCHAR(30) NOT NULL,
  `Suggestion` VARCHAR(255) NOT NULL,
  `Status` VARCHAR(30) NOT NULL,
  PRIMARY KEY (`Product ID`),
  FOREIGN KEY (`Product ID`) REFERENCES `Products` (`Product ID`) ON DELETE CASCADE
);

CREATE TABLE ReviewClassifier (
  `Product ID` INT NOT NULL,
  `CNN Positive` INT NOT NULL,
  `CNN Negative` INT NOT NULL,
  `LSTM Positive` INT NOT NULL,
  `LSTM Negative` INT NOT NULL,
  `Transformer Positive` INT NOT NULL,
  `Transformer Negative` INT NOT NULL,
  `Total Number of Reviews` INT NOT NULL,
  `Total number of Reviews Present in Web` INT NOT NULL,
  PRIMARY KEY (`Product ID`),
  FOREIGN KEY (`Product ID`) REFERENCES `Products` (`Product ID`) ON DELETE CASCADE
);

CREATE TABLE ReviewSummary (
  `Product ID` INT NOT NULL,
  `Extractive Positive Summary` TEXT NOT NULL,
  `Extractive Negative Summary` TEXT NOT NULL,
  `Abstractive Positive Summary` TEXT NOT NULL,
  `Abstractive Negative Summary` TEXT NOT NULL,
  `Product detail` TEXT NOT NULL,
  PRIMARY KEY (`Product ID`),
  FOREIGN KEY (`Product ID`) REFERENCES `Products` (`Product ID`) ON DELETE CASCADE
);

CREATE TABLE AmazonSummary (
  `Product ID` INT NOT NULL,
  `Amazon Summary` TEXT NOT NULL,
```

```
  PRIMARY KEY (`Product ID`),
  FOREIGN KEY (`Product ID`) REFERENCES `Products` (`Product ID`) ON DELETE CASCADE
);

CREATE TABLE imageDirs (
  `Product ID` INT NOT NULL,
  `imagePath1` VARCHAR(255),
  `imagePath2` VARCHAR(255),
  `imagePath3` VARCHAR(255),
  `imagePath4` VARCHAR(255),
  PRIMARY KEY (`Product ID`),
  FOREIGN KEY (`Product ID`) REFERENCES `Products` (`Product ID`) ON DELETE CASCADE
);
```

Python file to manage the backend

app.py

```
from flask import Flask, render_template, request, flash, redirect, url_for, send_file
from WebScraper import productDetailAmazon
from WebScraper import amazonProductReviewScraper
from WebScraper import flipkartProductReviewPartial
from WebScraper import productDetailFlipkart
from WebScraper import productDetailGeneral
from WebScraper import getAmazonpgs
from WebScraper import getFlipkartpgs
from WebScraper import maxFile
from WebScraper import backgroundRemover
from SentimentSummarizer import classifier
from SentimentSummarizer import wordCloud
from SentimentSummarizer import summarizer
from datetime import date
import asyncio
import secrets
import threading
import ast
import mysql.connector
import atexit
import signal

app = Flask(__name__)
app.secret_key = secrets.token_hex(16)

searchURL = None
```

```
amazonProdInfo = {}
flipkartProdInfo = {}

genProdInfo = {}

ai = []

n = 0

img_lst = []

content1 = []
content2 = {}

content3 = None

image_dir = None

image_dir1 = None
image_dir2 = None
image_dir3 = None

numRev = 0
numPgs = 0
number_pgs = 0

seller = None

thread1 = None

db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="rameez",
    database="product_detail_two"
)

def finalContentAmazon():
    global image_dir1
    global image_dir2
    global image_dir3

    thread1.join()
    today = date.today()

    genProdInfo = productDetailGeneral.prodDetail
    if amazonProdInfo['Title'] == " ":
```

```
    if genProdInfo['Title'] == " ":
        content2['Title'] = "-"
    else:
        content2['Title'] = genProdInfo['Title']
else:
    content2['Title'] = amazonProdInfo['Title']

if genProdInfo['Highest Price'] == " ":
    content2['HighestPrice'] = "-"
else:
    content2['HighestPrice'] = genProdInfo['Highest Price']

if genProdInfo['Date of highest price'] == " ":
    content2['DateofHighestPrice'] = "-"
else:
    content2['DateofHighestPrice'] = genProdInfo['Date of highest price']

if genProdInfo['Lowest Price'] == " ":
    content2['LowestPrice'] = "-"
else:
    content2['LowestPrice'] = genProdInfo['Lowest Price']

if genProdInfo['Date of lowest price'] == " ":
    content2['DateofLowestPrice'] = "-"
else:
    content2['DateofLowestPrice'] = genProdInfo['Date of lowest price']

if genProdInfo['Average Price'] == " ":
    content2['AveragePrice'] = "-"
else:
    content2['AveragePrice'] = genProdInfo['Average Price']

if genProdInfo['Average Price as of'] == " ":
    content2['DateofAveragePrice'] = "-"
else:
    content2['DateofAveragePrice'] = genProdInfo['Average Price as of']

if amazonProdInfo['Current Price'] == " ":
    if genProdInfo['Current Price'] == " ":
        content2['CurrentPrice'] = "-"
    else:
        content2['CurrentPrice'] = genProdInfo['Current Price']
else:
    content2['CurrentPrice'] = amazonProdInfo['Current Price']

if genProdInfo['Label Price'] == " ":
    content2['LabelPrice'] = "-"
```

```
else:
    content2['LabelPrice'] = genProdInfo['Label Price']

if genProdInfo['Savings'] == " ":
    content2['Savings'] = "-"
else:
    content2['Savings'] = genProdInfo['Savings']

if genProdInfo['Discount'] == " ":
    content2['Discount'] = "-"
else:
    content2['Discount'] = genProdInfo['Discount']

if genProdInfo['Suggestion'] == " ":
    content2['Suggestion'] = "-"
else:
    content2['Suggestion'] = genProdInfo['Suggestion']

if genProdInfo['Status'] == " ":
    content2['Status'] = "-"
else:
    content2['Status'] = genProdInfo['Status']

if amazonProdInfo['Amazon AI'] == " ":
    content2['AmazonAI'] = "-"
else:
    content2['AmazonAI'] = amazonProdInfo['Amazon AI']

content2['NumRev'] = numRev
content2['UsrNumRev'] = int(number_pgs)*10

classifier.perform_classification()
wordCloud.create_wordcloud()

image_dir1 = wordCloud.d1
image_dir2 = wordCloud.d2
image_dir3 = wordCloud.d3

content2['pos1'] = int(classifier.pos1)
content2['pos2'] = int(classifier.pos2)
content2['pos3'] = int(classifier.pos3)
content2['neg1'] = int(classifier.neg1)
content2['neg2'] = int(classifier.neg2)
content2['neg3'] = int(classifier.neg3)

summarizer.perform_summarize_on_df()
```

```
    content2['posSum'] = summarizer.positive_summary
    content2['negSum'] = summarizer.negative_summary
    content2['EposSum'] = summarizer.e_positive_summary
    content2['EnegSum'] = summarizer.e_negative_summary
    content2['About'] = amazonProdInfo['About']

    mycursor = db.cursor()
    name = content2['Title']
    name = " ".join(name.split()[:4])
    mycursor.execute("INSERT INTO Products (Name, Date, ImagePath, ImagePath1, ImagePath2,
ImagePath3, origin) VALUES (%s, %s, %s, %s, %s, %s, %s)",(name,today,image_dir,
image_dir1, image_dir2, image_dir3, 'amazon'))
    last_inserted_id = mycursor.lastrowid
    mycursor.execute("INSERT INTO PricingDetails (`Product ID`, Title, `Highest Price`,
`Date of Highest Price`, `Lowest Price`, `Date of Lowest Price`, `Average Price`, `Date of
Average Price`, `Current Price`, `Label Price`, Savings, Discount, Suggestion, Status)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s)",(last_inserted_id,content2['Title'],content2['HighestPrice'],content2['DateofHighestP
rice'],content2['LowestPrice'],content2['DateofLowestPrice'],content2['AveragePrice'],cont
ent2['DateofAveragePrice'],content2['CurrentPrice'],content2['LabelPrice'],content2['Savin
gs'],content2['Discount'],content2['Suggestion'],content2['Status']))
    mycursor.execute("INSERT INTO ReviewClassifier (`Product ID`, `CNN Positive`, `CNN
Negative`, `LSTM Positive`, `LSTM Negative`, `Transformer Positive`, `Transformer
Negative`, `Total Number of Reviews`, `Total number of Reviews Present in Web`) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s,
%s)",(last_inserted_id,content2['pos1'],content2['neg1'],content2['pos2'],content2['neg2']
,content2['pos3'],content2['neg3'],content2['UsrNumRev'],content2['NumRev']))
    mycursor.execute("INSERT INTO ReviewSummary (`Product ID`, `Extractive Positive
Summary`, `Extractive Negative Summary`, `Abstractive Positive Summary`, `Abstractive
Negative Summary`, `Product detail`) VALUES (%s, %s, %s, %s, %s,
%s)",(last_inserted_id,content2['EposSum'],content2['EnegSum'],content2['posSum'],content2
['negSum'],str(content2['About'])))
    mycursor.execute("INSERT INTO AmazonSummary (`Product ID`, `Amazon Summary`) VALUES
(%s, %s)",(last_inserted_id,content2['AmazonAI']))
    db.commit()
    mycursor.close()
    #db.close()
    print("Product Details of amazon are inserted successfully in the table")

def finalContentFlipkart():
    global image_dir1
    global image_dir2
    global image_dir3

    thread1.join()
    today = date.today()
    genProdInfo = productDetailGeneral.prodDetail
```

```
if flipkartProdInfo['Title'] == " ":
    if genProdInfo['Title'] == " ":
        content2['Title'] = "-"
    else:
        content2['Title'] = genProdInfo['Title']
else:
    content2['Title'] = flipkartProdInfo['Title']
if genProdInfo['Highest Price'] == " ":
    content2['HighestPrice'] = "-"
else:
    content2['HighestPrice'] = genProdInfo['Highest Price']
if genProdInfo['Date of highest price'] == " ":
    content2['DateofHighestPrice'] = "-"
else:
    content2['DateofHighestPrice'] = genProdInfo['Date of highest price']
if genProdInfo['Lowest Price'] == " ":
    content2['LowestPrice'] = "-"
else:
    content2['LowestPrice'] = genProdInfo['Lowest Price']
if genProdInfo['Date of lowest price'] == " ":
    content2['DateofLowestPrice'] = "-"
else:
    content2['DateofLowestPrice'] = genProdInfo['Date of lowest price']
if genProdInfo['Average Price'] == " ":
    content2['AveragePrice'] = "-"
else:
    content2['AveragePrice'] = genProdInfo['Average Price']
if genProdInfo['Average Price as of'] == " ":
    content2['DateofAveragePrice'] = "-"
else:
    content2['DateofAveragePrice'] = genProdInfo['Average Price as of']
if flipkartProdInfo['Current Price'] == " ":
    if genProdInfo['Current Price'] == " ":
        content2['CurrentPrice'] = "-"
    else:
        content2['CurrentPrice'] = genProdInfo['Current Price']
else:
    content2['CurrentPrice'] = flipkartProdInfo['Current Price']
if genProdInfo['Label Price'] == " ":
    content2['LabelPrice'] = "-"
else:
    content2['LabelPrice'] = genProdInfo['Label Price']
if genProdInfo['Savings'] == " ":
    content2['Savings'] = "-"
else:
    content2['Savings'] = genProdInfo['Savings']
if genProdInfo['Discount'] == " ":
```

```
        content2['Discount'] = "-"
    else:
        content2['Discount'] = genProdInfo['Discount']
    if genProdInfo['Suggestion'] == " ":
        content2['Suggestion'] = "-"
    else:
        content2['Suggestion'] = genProdInfo['Suggestion']
    if genProdInfo['Status'] == " ":
        content2['Status'] = "-"
    else:
        content2['Status'] = genProdInfo['Status']
    content2['NumRev'] = numRev
    content2['UsrNumRev'] = int(number_pgs)*10

    classifier.perform_classification()
    wordCloud.create_wordcloud()

    image_dir1 = wordCloud.d1
    image_dir2 = wordCloud.d2
    image_dir3 = wordCloud.d3

    content2['pos1'] = int(classifier.pos1)
    content2['pos2'] = int(classifier.pos2)
    content2['pos3'] = int(classifier.pos3)
    content2['neg1'] = int(classifier.neg1)
    content2['neg2'] = int(classifier.neg2)
    content2['neg3'] = int(classifier.neg3)

    summarizer.perform_summarize_on_df()

    content2['posSum'] = summarizer.positive_summary
    content2['negSum'] = summarizer.negative_summary
    content2['EposSum'] = summarizer.e_positive_summary
    content2['EnegSum'] = summarizer.e_negative_summary

    content2['About'] = flipkartProdInfo['About']

    mycursor = db.cursor()
    name = content2['Title']
    name = " ".join(name.split()[:4])
    mycursor.execute("INSERT INTO Products (Name, Date, ImagePath, ImagePath1, ImagePath2,
ImagePath3, origin) VALUES (%s, %s, %s, %s, %s, %s, %s)",(name,today,image_dir,
image_dir1, image_dir2, image_dir3, 'flipkart'))
    last_inserted_id = mycursor.lastrowid
    mycursor.execute("INSERT INTO PricingDetails (`Product ID`, Title, `Highest Price`,
`Date of Highest Price`, `Lowest Price`, `Date of Lowest Price`, `Average Price`, `Date of
Average Price`, `Current Price`, `Label Price`, Savings, Discount, Suggestion, Status)
```

```
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s)",(last_inserted_id,content2['Title'],content2['HighestPrice'],content2['DateofHighestP
rice'],content2['LowestPrice'],content2['DateofLowestPrice'],content2['AveragePrice'],cont
ent2['DateofAveragePrice'],content2['CurrentPrice'],content2['LabelPrice'],content2['Savin
gs'],content2['Discount'],content2['Suggestion'],content2['Status']))
    mycursor.execute("INSERT INTO ReviewClassifier (`Product ID`, `CNN Positive`, `CNN
Negative`, `LSTM Positive`, `LSTM Negative`, `Transformer Positive`, `Transformer
Negative`, `Total Number of Reviews`, `Total number of Reviews Present in Web`) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s,
%s)",(last_inserted_id,content2['pos1'],content2['neg1'],content2['pos2'],content2['neg2']
,content2['pos3'],content2['neg3'],content2['UsrNumRev'],content2['NumRev']))
    mycursor.execute("INSERT INTO ReviewSummary (`Product ID`, `Extractive Positive
Summary`, `Extractive Negative Summary`, `Abstractive Positive Summary`, `Abstractive
Negative Summary`, `Product detail`) VALUES (%s, %s, %s, %s, %s,
%s)",(last_inserted_id,content2['EposSum'],content2['EnegSum'],content2['posSum'],content2
['negSum'],str(content2['About'])))
    db.commit()
    mycursor.close()
    #db.close()
    print("Product Details of flipkart are inserted successfully in the table")

def ContentHistoryAmazon(prod_id):
    global image_dir
    global image_dir1
    global image_dir2
    global image_dir3

    mycursor = db.cursor()
    product_id = prod_id

    prod = "SELECT * FROM Products WHERE `Product ID` = %s"
    mycursor.execute(prod, (product_id,))
    prod_meta_data = mycursor.fetchall()

    image_dir = prod_meta_data[0][3]
    image_dir1 = prod_meta_data[0][4]
    image_dir2 = prod_meta_data[0][5]
    image_dir3 = prod_meta_data[0][6]

    # Fetching data from AmazonSummary
    amazon_query = "SELECT * FROM AmazonSummary WHERE `Product ID` = %s"
    mycursor.execute(amazon_query, (product_id,))
    amazon_data = mycursor.fetchall()

    # Fetching data from PricingDetails
    pricing_query = "SELECT * FROM PricingDetails WHERE `Product ID` = %s"
    mycursor.execute(pricing_query, (product_id,))
```

```
    pricing_data = mycursor.fetchall()

    # Fetching data from ReviewClassifier
    classifier_query = "SELECT * FROM ReviewClassifier WHERE `Product ID` = %s"
    mycursor.execute(classifier_query, (product_id,))
    classifier_data = mycursor.fetchall()

    # Fetching data from ReviewSummary
    summary_query = "SELECT * FROM ReviewSummary WHERE `Product ID` = %s"
    mycursor.execute(summary_query, (product_id,))
    summary_data = mycursor.fetchall()

    mycursor.close()

    content2['Title'] = str(pricing_data[0][1])
    content2['HighestPrice'] = str(pricing_data[0][2])
    content2['DateofHighestPrice'] = str(pricing_data[0][3])
    content2["LowestPrice"] = str(pricing_data[0][4])
    content2['DateofLowestPrice'] = str(pricing_data[0][5])
    content2["AveragePrice"] = str(pricing_data[0][6])
    content2['DateofAveragePrice'] = str(pricing_data[0][7])
    content2['LabelPrice'] = str(pricing_data[0][8])
    content2['Savings'] = str(pricing_data[0][9])
    content2['Discount'] = str(pricing_data[0][10])
    content2['Suggestion'] = str(pricing_data[0][11])
    content2['Status'] = str(pricing_data[0][12])

    content2['pos1'] = str(classifier_data[0][1])
    content2['neg1'] = str(classifier_data[0][2])
    content2['pos2'] = str(classifier_data[0][3])
    content2['neg2'] = str(classifier_data[0][4])
    content2['pos3'] = str(classifier_data[0][5])
    content2['neg3'] = str(classifier_data[0][6])
    content2['NumRev'] = str(classifier_data[0][7])
    content2['UsrNumRev'] = str(classifier_data[0][8])

    content2['EposSum'] = str(summary_data[0][1])
    content2['EnegSum'] = str(summary_data[0][2])
    content2['posSum'] = str(summary_data[0][3])
    content2['negSum'] = str(summary_data[0][4])
    list_s = ast.literal_eval(summary_data[0][5])
    content2['About'] = list_s
    content2['AmazonAI'] = str(amazon_data[0][1])

def ContentHistoryFlipkart(prod_id):
    global image_dir
    global image_dir1
```

```
global image_dir2
global image_dir3

mycursor = db.cursor()
product_id = prod_id

prod = "SELECT * FROM Products WHERE `Product ID` = %s"
mycursor.execute(prod, (product_id,))
prod_meta_data = mycursor.fetchall()

image_dir = prod_meta_data[0][3]
image_dir1 = prod_meta_data[0][4]
image_dir2 = prod_meta_data[0][5]
image_dir3 = prod_meta_data[0][6]

# Fetching data from PricingDetails
pricing_query = "SELECT * FROM PricingDetails WHERE `Product ID` = %s"
mycursor.execute(pricing_query, (product_id,))
pricing_data = mycursor.fetchall()

# Fetching data from ReviewClassifier
classifier_query = "SELECT * FROM ReviewClassifier WHERE `Product ID` = %s"
mycursor.execute(classifier_query, (product_id,))
classifier_data = mycursor.fetchall()

# Fetching data from ReviewSummary
summary_query = "SELECT * FROM ReviewSummary WHERE `Product ID` = %s"
mycursor.execute(summary_query, (product_id,))
summary_data = mycursor.fetchall()

mycursor.close()

content2['Title'] = str(pricing_data[0][1])
content2['HighestPrice'] = str(pricing_data[0][2])
content2['DateofHighestPrice'] = str(pricing_data[0][3])
content2["LowestPrice"] = str(pricing_data[0][4])
content2['DateofLowestPrice'] = str(pricing_data[0][5])
content2["AveragePrice"] = str(pricing_data[0][6])
content2['DateofAveragePrice'] = str(pricing_data[0][7])
content2['LabelPrice'] = str(pricing_data[0][8])
content2['Savings'] = str(pricing_data[0][9])
content2['Discount'] = str(pricing_data[0][10])
content2['Suggestion'] = str(pricing_data[0][11])
content2['Status'] = str(pricing_data[0][12])

content2['pos1'] = str(classifier_data[0][1])
content2['neg1'] = str(classifier_data[0][2])
```

```
    content2['pos2'] = str(classifier_data[0][3])
    content2['neg2'] = str(classifier_data[0][4])
    content2['pos3'] = str(classifier_data[0][5])
    content2['neg3'] = str(classifier_data[0][6])
    content2['NumRev'] = str(classifier_data[0][7])
    content2['UsrNumRev'] = str(classifier_data[0][8])

    content2['EposSum'] = str(summary_data[0][1])
    content2['EnegSum'] = str(summary_data[0][2])
    content2['posSum'] = str(summary_data[0][3])
    content2['negSum'] = str(summary_data[0][4])
    list_s = ast.literal_eval(summary_data[0][5])
    content2['About'] = list_s


@app.route('/')
def index():
    return render_template('home.html')

@app.route('/image')
def serve_image():
    global image_dir
    return send_file(image_dir, mimetype='image/png')

@app.route('/image1')
def serve_image_1():
    global image_dir1
    return send_file(image_dir1, mimetype='image/png')

@app.route('/image22')
def serve_image_22():
    global image_dir2
    return send_file(image_dir2, mimetype='image/png')

@app.route('/image3')
def serve_image_3():
    global image_dir3
    return send_file(image_dir3, mimetype='image/png')

@app.route('/image2')
def serve_image_2():
    global n
    img = img_lst[n]
    n+=1
    #print(img)
    return send_file(img, mimetype='image/jpg')
```

```
@app.route('/submitURL', methods=['POST', 'GET'])
async def submit_URL():
    global numRev
    global numPgs
    global searchURL
    global amazonProdInfo
    global flipkartProdInfo
    global thread1
    global image_dir
    searchURL = request.form['searchURL']
    if 'amazon' in searchURL:
        try:
            loop = asyncio.get_event_loop()
            amazonProdInfo = await loop.run_in_executor(None, lambda:
productDetailAmazon.ext(searchURL))
            thread1 = threading.Thread(target=productDetailGeneral.ext, args=(searchURL,))
            thread1.start()
            numRev = await loop.run_in_executor(None, lambda:
getAmazonpgs.totalReview(searchURL))
            await loop.run_in_executor(None, lambda:
backgroundRemover.remove_background())
            image_dir = "C:\\users\\ramee\\Desktop\\AI
Lab\\Project\\Image"+"\\"+maxFile.get_max_numbered_png("C:\\users\\ramee\\Desktop\\AI
Lab\\Project\\Image")
            numPgs = int(int(numRev) / 10) + 1
            content1.append(amazonProdInfo['Title'])
            content1.append(numRev)
            content1.append(numPgs)
            return redirect(url_for('get_number_pages'))
        except Exception as e:
            print(e)
            return redirect('/')
    elif 'flipkart' in searchURL:
        try:
            loop = asyncio.get_event_loop()
            flipkartProdInfo = await loop.run_in_executor(None, lambda:
productDetailFlipkart.ext(searchURL))
            thread1 = threading.Thread(target=productDetailGeneral.ext, args=(searchURL,))
            thread1.start()
            await loop.run_in_executor(None, lambda:
backgroundRemover.remove_background())
            numRev = await loop.run_in_executor(None, lambda:
getFlipkartpgs.totalReviews(searchURL))
            image_dir = "C:\\users\\ramee\\Desktop\\AI
Lab\\Project\\Image"+"\\"+maxFile.get_max_numbered_png("C:\\users\\ramee\\Desktop\\AI
Lab\\Project\\Image")
            numPgs = int(int(numRev) / 10)
```

```
            content1.append(flipkartProdInfo['Title'])
            content1.append(numRev)
            content1.append(numPgs)
            return redirect(url_for('get_number_pages'))
        except Exception as e:
            print(e)
            return redirect('/')
    else:
        flash('Invalid URL. Please enter a valid URL.', 'error')
        return redirect('/')

@app.route('/pageNumbers', methods=['GET', 'POST'])
async def get_number_pages():
    global thread2
    global genProdInfo
    global number_pgs
    if request.method == "GET":
        return render_template('page1.html', content=content1)
    elif request.method == "POST":
        number_pgs = request.form['pgNum']
        if int(number_pgs) > numPgs or int(number_pgs) < 0:
            flash('Invalid number of pages. Please enter a valid number', 'error')
            return render_template('page1.html', content=content1)
        else:
            if "amazon" in searchURL:
                loop = asyncio.get_event_loop()
                thread2 = threading.Thread(target=amazonProductReviewScraper.get_review,
args=(searchURL,int(number_pgs)))
                thread2.start()
                await loop.run_in_executor(None, lambda: finalContentAmazon())
                return render_template('page2Amazon.html', content=content2)
            elif "flipkart" in searchURL:
                loop = asyncio.get_event_loop()
                thread2 = threading.Thread(target=flipkartProductReviewPartial.get_review,
args=(searchURL,number_pgs))
                thread2.start()
                await loop.run_in_executor(None, lambda: finalContentFlipkart())
                return render_template('page2Flipkart.html', content=content2)

@app.route('/history', methods=['POST', 'GET'])
async def history():
    global content3
    global seller
    if request.method == "GET":
        mycursor = db.cursor()
        mycursor.execute("SELECT * FROM Products")
        content3 = mycursor.fetchall()
```

```
        mycursor.close()
        for i in content3:
            img_lst.append(i[3])
        #print(img_lst)
        return render_template('history.html', content=content3)
    elif request.method == "POST":
        loop = asyncio.get_event_loop()
        product_id = request.form['productID']
        product_id = int(product_id)
        mycursor = db.cursor()
        mycursor.execute("SELECT * FROM Products WHERE `Product ID` = %s",(product_id,))
        origin = mycursor.fetchall()
        seller = str(origin[0][-1])
        if seller == "amazon":
            await loop.run_in_executor(None, lambda: ContentHistoryAmazon(product_id))
            return redirect('/historyInfo')
        else:
            await loop.run_in_executor(None, lambda: ContentHistoryFlipkart(product_id))
            return redirect('/historyInfo')

@app.route('/historyInfo')
def historyInfo():
    if seller == "amazon":
        return render_template('page2Amazon.html',content=content2)
    else:
        return render_template('page2Flipkart.html',content=content2)


def close_db_connection():
    """Function to close the database connection."""
    if db.is_connected():
        db.close()
        print("Database connection closed.")

atexit.register(close_db_connection)

def signal_handler(signal, frame):
    """Signal handler for SIGINT to close the database connection."""
    close_db_connection()
    exit(0)

signal.signal(signal.SIGINT, signal_handler)

if __name__ == '__main__':
    app.run(debug=True)
```

Python Files for Training the Models

CNN.py

LSTM.py