LAPTRACK

University at Buffalo, SUNY

December 2024

Team Members

- **Member 1** Name: Shaurya Mathur UB ID: 50611201 E-Mail: `smathur4@buffalo.edu`
- **Member 2** Name: Vaibhav Saran UB ID: 50615031 E-Mail: `vsaran@buffalo.edu`
- **Member 3** Name: Yeswanth Chitturi UB ID: 50591666 E-Mail: `ychittur@buffalo.edu`

Abstract

This project analyzes laptops across Amazon, Flipkart, and BestBuy to understand factors influencing their pricing and specifications that impact customer purchasing decisions. With the competitive laptop market offering a wide range of models and features, it has become increasingly challenging for consumers to make informed choices. By studying extensive e-commerce data, this project identifies critical factors affecting laptop prices, such as processor type, storage capacity, screen size, RAM, and brand reputation. It also highlights customer priorities like battery life, display quality, and processing power.

Understanding these relationships is vital for sellers aiming to optimize product offerings and pricing strategies. Insights from this analysis help retailers adjust prices to reflect features that add value, such as higher RAM or larger storage, ensuring competitiveness while meeting customer needs. Additionally, identifying customer preferences supports the development of targeted marketing strategies emphasizing features most likely to drive purchases.

For consumers, the study offers a clearer perspective on which features matter most within specific price ranges, enabling better purchasing decisions. By revealing the features that significantly increase laptop value and those providing the best value for money, buyers can make more informed choices tailored to their requirements.

The dataset, sourced from Amazon, Flipkart, and BestBuy, encompasses diverse brands, models, pricing strategies, and customer reviews. This comprehensive analysis provides insights into the laptop market, benefiting both sellers and buyers by bridging the gap between pricing strategies and customer expectations.

Contents

1 Introduction

1.1 Problem Statement

The rapid advancement of technology has led to a wide variety of laptops with diverse specifications, features, and price points, making it challenging for consumers to make informed decisions. Similarly, sellers face difficulties in optimizing pricing strategies to remain competitive. Understanding the factors influencing laptop prices and the specifications customers prioritize is crucial for both buyers and sellers.

This project addresses these challenges by analyzing laptops listed on Amazon, Flipkart, and BestBuy to uncover the key features impacting laptop pricing and customer purchasing decisions. Specifically, the objectives are:

- **Identify key features influencing laptop prices:** Analyze specifications such as processor type, RAM size, storage capacity, screen size, and brand to understand their impact on pricing and variations across platforms.
- **Understand customer priorities:** Highlight specifications important to customers based on preferences and reviews, providing valuable insights for manufacturers to design better products.
- **Examine features influencing purchase decisions:** Explore how technical specifications, appearance, brand, and customer impressions shape buying choices.

1.2 Objectives and Significance

This project provides insights for both sellers and customers:

- **For Sellers:** Identify features driving purchasing decisions to refine pricing strategies and product offerings. Leverage insights to design laptops aligned with customer preferences, boosting sales and satisfaction.
- **For Customers:** Help buyers make informed decisions by understanding essential features within their price range. Compare laptops across platforms to maximize value for money.

By analyzing key factors such as RAM, storage, processor, and screen size, this study bridges the gap between consumer expectations and market offerings. A recommendation engine will be developed to suggest laptops tailored to buyer preferences, incorporating customer reviews and ratings. This tool reduces decision fatigue, improves the consumer experience, and supports sellers in optimizing their pricing and marketing strategies. The findings will benefit both buyers and sellers, ensuring a better alignment of product features, pricing, and consumer demand.

2 Data Collection

To ensure a comprehensive analysis, data was gathered from prominent e-commerce platforms, including Amazon, Flipkart, and BestBuy. These platforms were chosen for their extensive laptop offerings and diverse customer reviews, which provide valuable insights into market trends and preferences.

Raw Data

The collected data includes product details, specifications, and customer reviews from e-commerce platforms.

Data Structuring

Unstructured data was organized into a structured format, with attributes such as brand, model, price, screen size, and specifications arranged into columns for analysis.

Data Cleaning

The dataset was pre-processed to ensure quality:

- **Removing Unnecessary Information:** Irrelevant data was filtered out.
- **Duplicate Removal:** Duplicate entries were identified and eliminated.
- **Handling Missing Data:** Missing values were filled with appropriate values or removed.

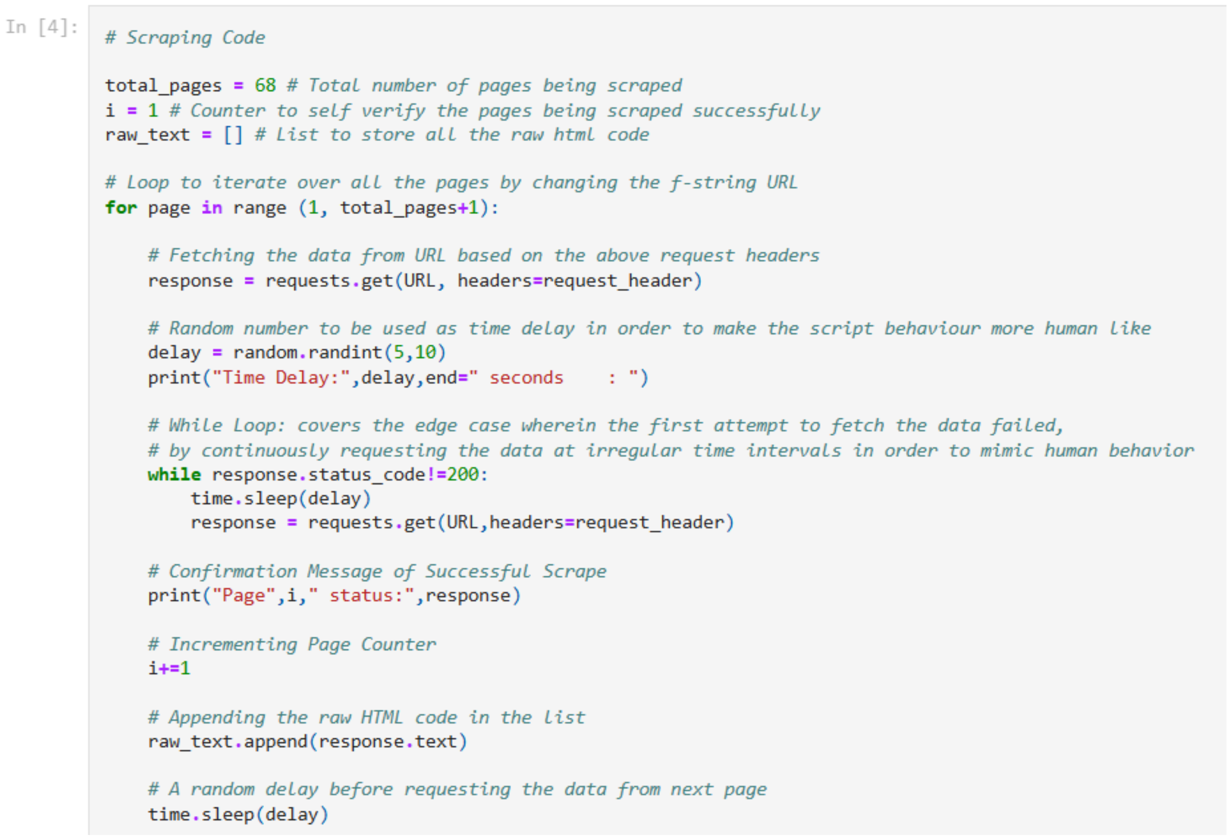
Data Export and Format

Cleaned data was exported as a CSV file. Each row represents a unique laptop product, ensuring granularity without aggregation.

Modules Used

- **requests:** Fetches web page data via HTTP requests.
- **BeautifulSoup:** Parses HTML for extracting product details.
- **re:** Applies regular expressions to filter and organize data.
- **numpy & pandas:** Handles datasets, creating dataframes and performing numerical operations.
- **selenium:** The Selenium module is an open-source framework that automates web browser interactions for testing and web scraping purposes.

2.1 Flipkart Data Collection



```
In [4]:
# Scraping Code

total_pages = 68 # Total number of pages being scraped
i = 1 # Counter to self verify the pages being scraped successfully
raw_text = [] # List to store all the raw html code

# Loop to iterate over all the pages by changing the f-string URL
for page in range (1, total_pages+1):

    # Fetching the data from URL based on the above request headers
    response = requests.get(URL, headers=request_header)

    # Random number to be used as time delay in order to make the script behaviour more human like
    delay = random.randint(5,10)
    print("Time Delay:",delay,end=" seconds    : ")

    # While Loop: covers the edge case wherein the first attempt to fetch the data failed,
    # by continuously requesting the data at irregular time intervals in order to mimic human behavior
    while response.status_code!=200:
        time.sleep(delay)
        response = requests.get(URL,headers=request_header)

    # Confirmation Message of Successful Scrape
    print("Page",i," status:",response)

    # Incrementing Page Counter
    i+=1

    # Appending the raw HTML code in the list
    raw_text.append(response.text)

    # A random delay before requesting the data from next page
    time.sleep(delay)
```

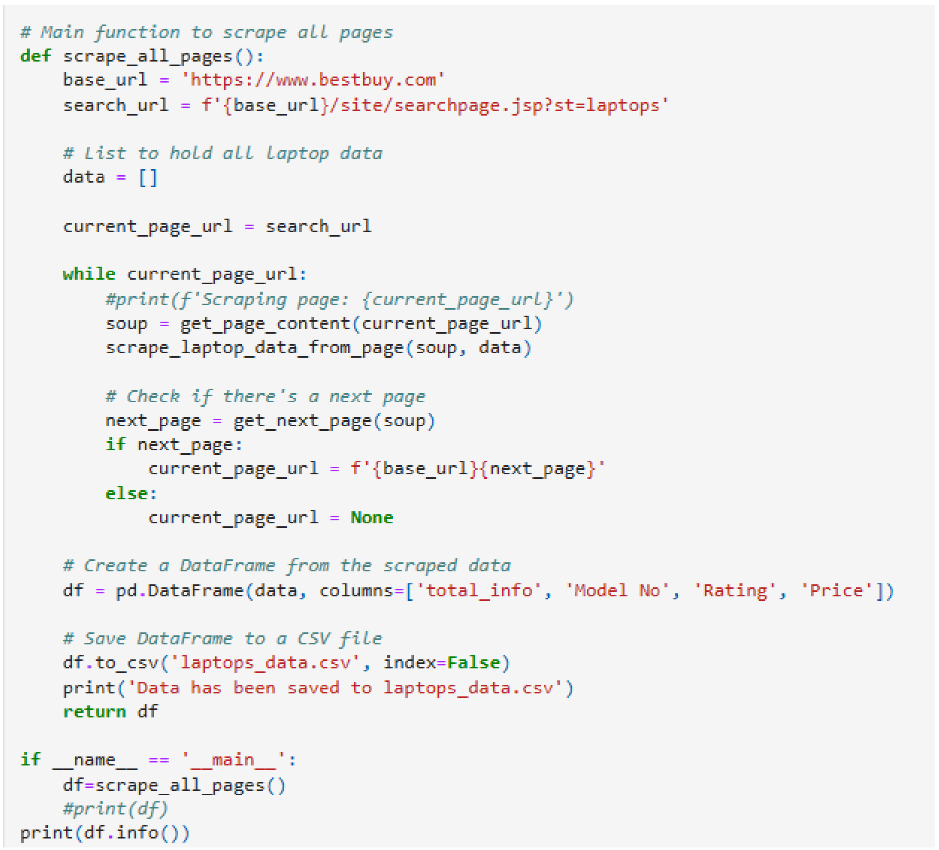
Figure 1: Flipkart Scraping Code

This Python script scrapes data from 68 web pages, storing the raw HTML content for analysis. It initializes a counter and a list to track the scraping process, dynamically generating URLs for each page. To mimic human behavior and avoid anti-scraping detection, the script introduces random delays of 5–10 seconds between requests using random.randint(). HTTP requests are sent with the requests.get() method, using headers to simulate browser behavior.

In the event of a failed request, the script retries fetching the page with a while loop until a successful response (status code 200) is received. Confirmation messages and random delays ensure the process avoids overwhelming the server. Successfully retrieved HTML content is added to the list for later processing.

Raw HTML data is processed and structured using regular expressions (regex). Initially unstructured, the data—comprising product details, specifications, and reviews—is transformed into a clean dataset with distinct columns like brand, model, price, and screen size. Duplicate and irrelevant entries are removed, ensuring high-quality, usable data for analysis and model building.

2.2 Best Buy Data Collection



```
# Main function to scrape all pages
def scrape_all_pages():
    base_url = 'https://www.bestbuy.com'
    search_url = f'{base_url}/site/searchpage.jsp?st=laptops'

    # List to hold all laptop data
    data = []

    current_page_url = search_url

    while current_page_url:
        #print(f'Scraping page: {current_page_url}')
        soup = get_page_content(current_page_url)
        scrape_laptop_data_from_page(soup, data)

        # Check if there's a next page
        next_page = get_next_page(soup)
        if next_page:
            current_page_url = f'{base_url}{next_page}'
        else:
            current_page_url = None

    # Create a DataFrame from the scraped data
    df = pd.DataFrame(data, columns=['total_info', 'Model No', 'Rating', 'Price'])

    # Save DataFrame to a CSV file
    df.to_csv('laptops_data.csv', index=False)
    print('Data has been saved to laptops_data.csv')
    return df

if __name__ == '__main__':
    df=scrape_all_pages()
    #print(df)
print(df.info())
```

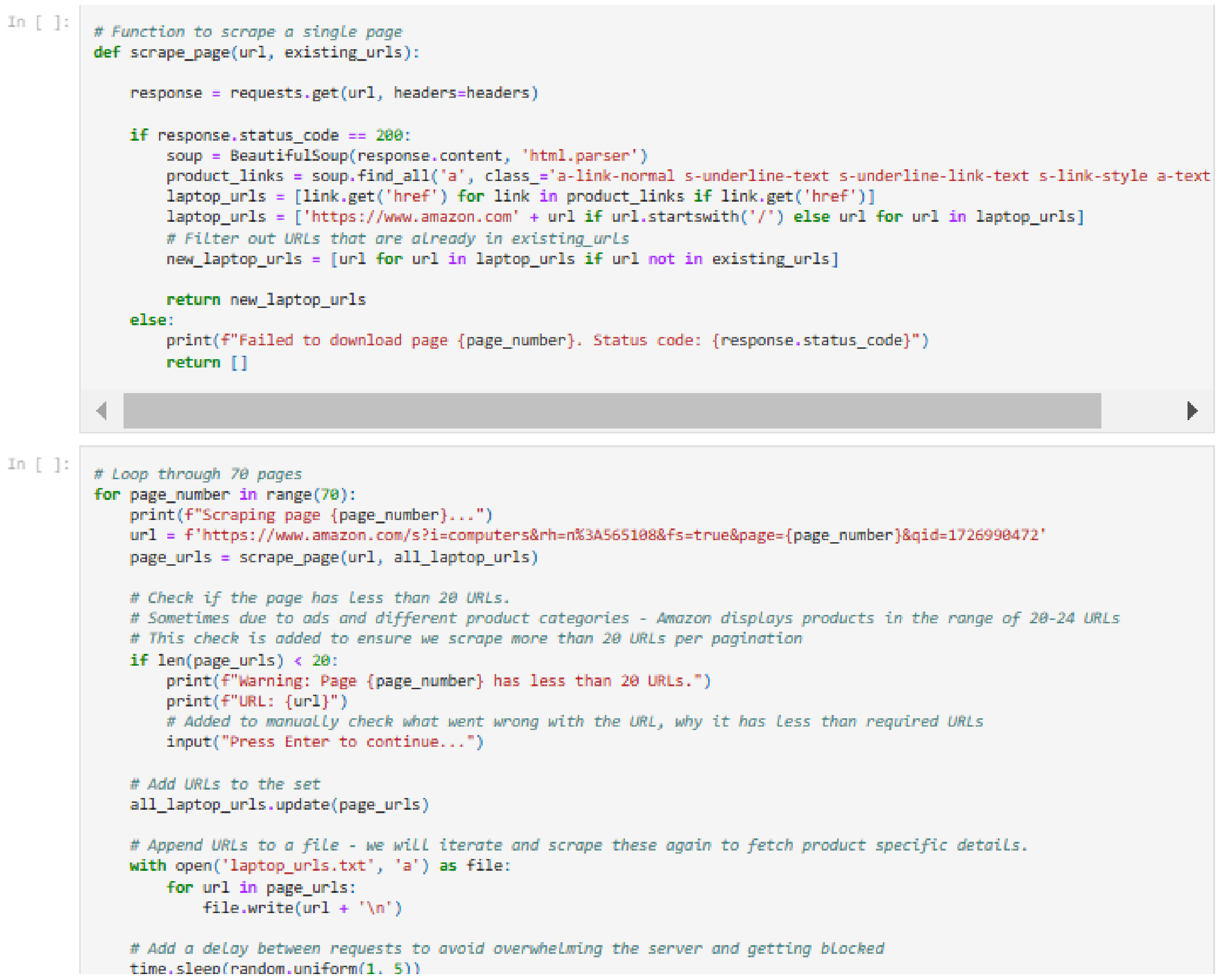
Figure 2: Sample Best buy Data Scraping code

The scrapeallpages() function automates scraping laptop data from BestBuy, extracting relevant details, and saving the data in a CSV file. The function starts by defining the base URL for laptops on BestBuy and initializing an empty list, data, to store scraped information. Using a while loop, it iteratively fetches pages, beginning with the first, using getpagecontent(currentpageurl) to retrieve the HTML and scrapelaptopdatafrompage(soup, data) to extract details like model, price, and ratings.

The loop checks for additional pages with getnextpage(soup) and continues until no "next" page is found, ensuring all data is collected. Once scraping is complete, the data is structured into a pandas DataFrame with columns like totalinfo, Model No, Rating, and Price, and saved to a CSV file, laptopsdata.csv. The function returns the DataFrame, and its structure is summarized with df.info().

Raw data was processed using regular expressions to extract attributes such as brand, model, and specifications, transforming unstructured content into organized columns. Data cleaning ensured consistency, removing duplicates and unnecessary details, yielding a high-quality dataset ready for analysis.

2.3 Amazon Data Collection



```
In [ ]:

# Function to scrape a single page
def scrape_page(url, existing_urls):

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')
        product_links = soup.find_all('a', class_='a-link-normal s-underline-text s-underline-link-text s-link-style a-text
        laptop_urls = [link.get('href') for link in product_links if link.get('href')]
        laptop_urls = ['https://www.amazon.com' + url if url.startswith('/') else url for url in laptop_urls]
        # Filter out URLs that are already in existing_urls
        new_laptop_urls = [url for url in laptop_urls if url not in existing_urls]

        return new_laptop_urls
    else:
        print(f"Failed to download page {page_number}. Status code: {response.status_code}")
        return []
```

```
In [ ]:

# Loop through 70 pages
for page_number in range(70):
    print(f"Scraping page {page_number}...")
    url = f'https://www.amazon.com/s?i=computers&rh=n%3A565108&fs=true&page={page_number}&qid=1726990472'
    page_urls = scrape_page(url, all_laptop_urls)

    # Check if the page has less than 20 URLs.
    # Sometimes due to ads and different product categories - Amazon displays products in the range of 20-24 URLs
    # This check is added to ensure we scrape more than 20 URLs per pagination
    if len(page_urls) < 20:
        print(f"Warning: Page {page_number} has less than 20 URLs.")
        print(f"URL: {url}")
        # Added to manually check what went wrong with the URL, why it has less than required URLs
        input("Press Enter to continue...")

    # Add URLs to the set
    all_laptop_urls.update(page_urls)

    # Append URLs to a file - we will iterate and scrape these again to fetch product specific details.
    with open('laptop_urls.txt', 'a') as file:
        for url in page_urls:
            file.write(url + '\n')

    # Add a delay between requests to avoid overwhelming the server and getting blocked
    time.sleep(random.uniform(1, 5))
```

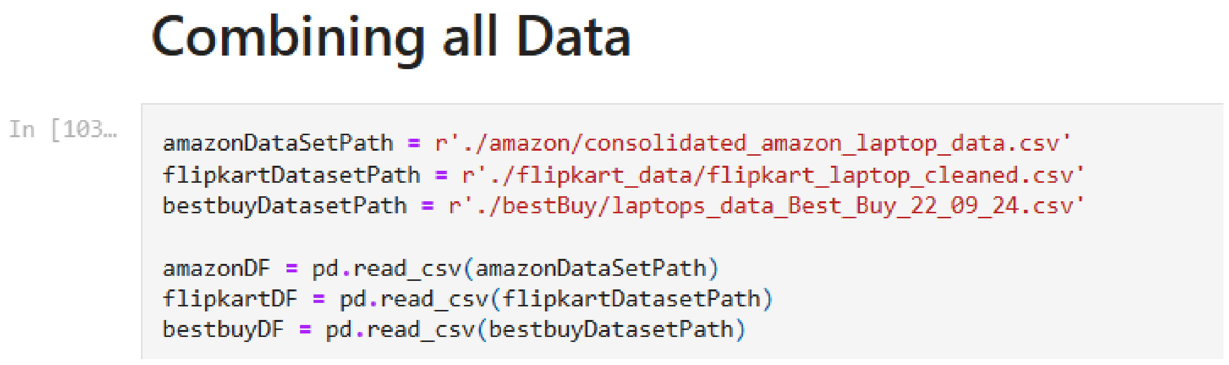
Figure 3: Sample Amazon Scraping Code

This Amazon scraping script collects URLs of laptop products from 70 pages of search results. The scrapepage function sends requests to Amazon, retrieves HTML content, and uses BeautifulSoup to parse it. It identifies product links from ¡a¿ tags with relevant class attributes, processes them into complete URLs, filters duplicates using the existingurls set, and returns the links as a list. If a request fails, an error message is logged for visibility.

In the main loop, URLs are dynamically generated for each page, and scrapepage extracts product links. The script flags pages with fewer than 20 links for investigation. Unique URLs are added to a set and saved to laptopurls.txt for persistence. To avoid detection, random delays between 1 and 5 seconds are introduced between requests. At the end, the script outputs the total number of unique URLs collected, ready for further detailed scraping.

The raw data is processed using regular expressions to extract key attributes like brand, model, price, and specifications. After cleaning to remove duplicates and inconsistencies, the data is loaded into an SQL database, prepared for analysis and insights.

2.4 Final Data

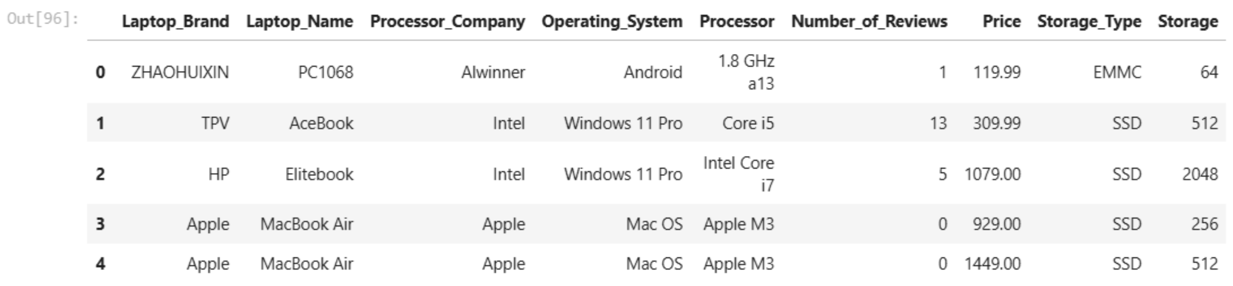


Combining all Data

```
In [103…
amazonDataSetPath = r'./amazon/consolidated_amazon_laptop_data.csv'
flipkartDatasetPath = r'./flipkart_data/flipkart_laptop_cleaned.csv'
bestbuyDatasetPath = r'./bestBuy/laptops_data_Best_Buy_22_09_24.csv'

amazonDF = pd.read_csv(amazonDataSetPath)
flipkartDF = pd.read_csv(flipkartDatasetPath)
bestbuyDF = pd.read_csv(bestbuyDatasetPath)
```

Figure 4: Combining Data

The raw data scraped from Amazon, BestBuy, and Flipkart was processed and structured using regular expressions to extract key attributes like brand, model, price, screen size, RAM, and ratings. Following this, the data was cleaned to remove duplicates and irrelevant information, ensuring consistency and accuracy across all platforms. The final cleaned datasets from all three websites were combined into a single, well-organized dataset. This dataset is now ready for analysis, providing valuable insights into laptop trends, pricing strategies, and customer preferences across the three platforms.



Out[96]:

	Laptop_Brand	Laptop_Name	Processor_Company	Operating_System	Processor	Number_of_Reviews	Price	Storage_Type	Storage
0	ZHAOHUIXIN	PC1068	Alwinner	Android	1.8 GHz a13	1	119.99	EMMC	64
1	TPV	AceBook	Intel	Windows 11 Pro	Core i5	13	309.99	SSD	512
2	HP	Elitebook	Intel	Windows 11 Pro	Intel Core i7	5	1079.00	SSD	2048
3	Apple	MacBook Air	Apple	Mac OS	Apple M3	0	929.00	SSD	256
4	Apple	MacBook Air	Apple	Mac OS	Apple M3	0	1449.00	SSD	512

Figure 5: Sample Data

3 EDA (Exploratory Data Analysis)

This section presents the EDA performed on the dataset to understand patterns and trends.

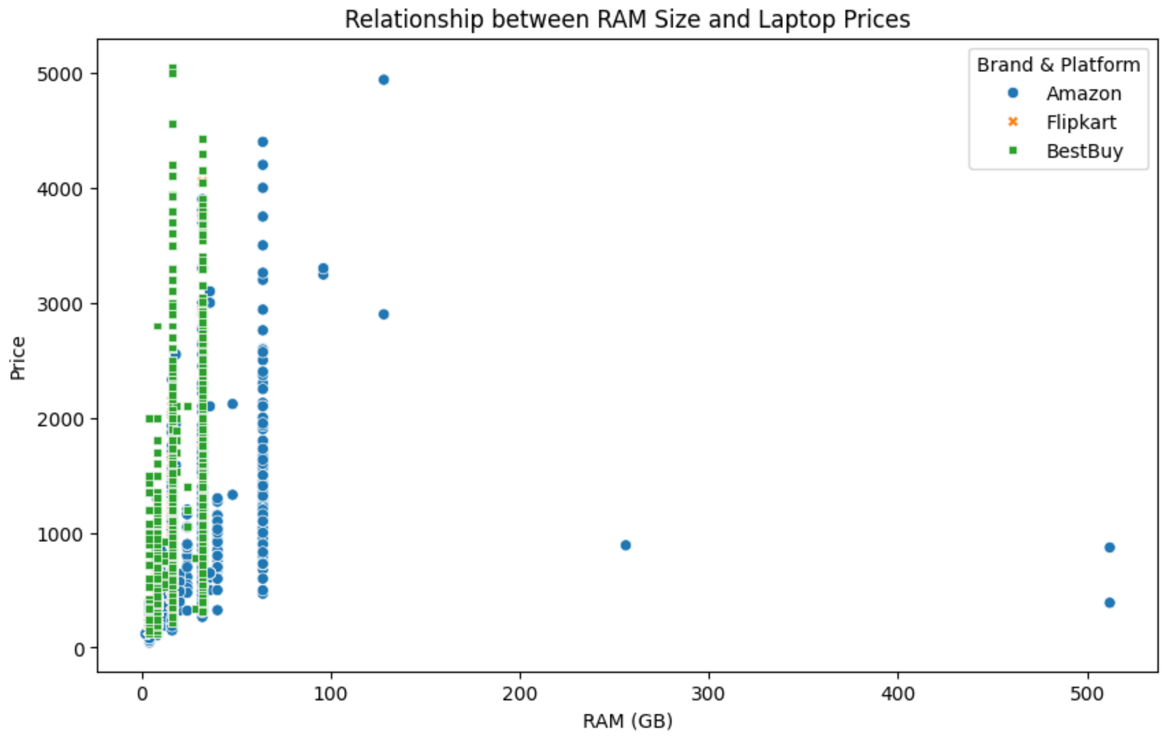
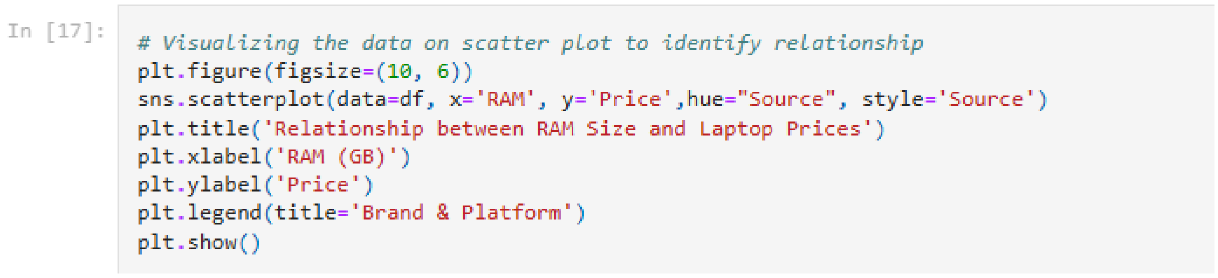


Figure 6: Relationship between RAM size and laptop prices

The plot indicates that for a given RAM size, the price of laptops can vary significantly, showing high variance. Notably, there are some outliers, such as a laptop with 250GB RAM priced at 1000 USD, which is from Amazon. While these outliers are present, they are not majorly impactful at this stage and can be addressed during model development if they cause issues. This observed variance is consistent across all the sources, as demonstrated by the plot above.



```
In [17]:
# Visualizing the data on scatter plot to identify relationship
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='RAM', y='Price',hue="Source", style='Source')
plt.title('Relationship between RAM Size and Laptop Prices')
plt.xlabel('RAM (GB)')
plt.ylabel('Price')
plt.legend(title='Brand & Platform')
plt.show()
```

Figure 7: Sample Code

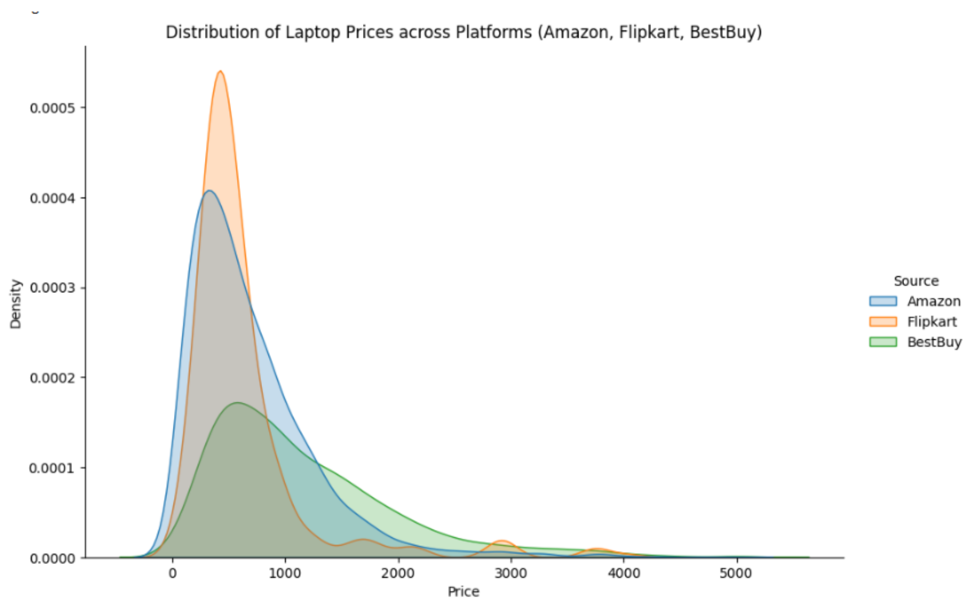


Figure 8: Across Brands

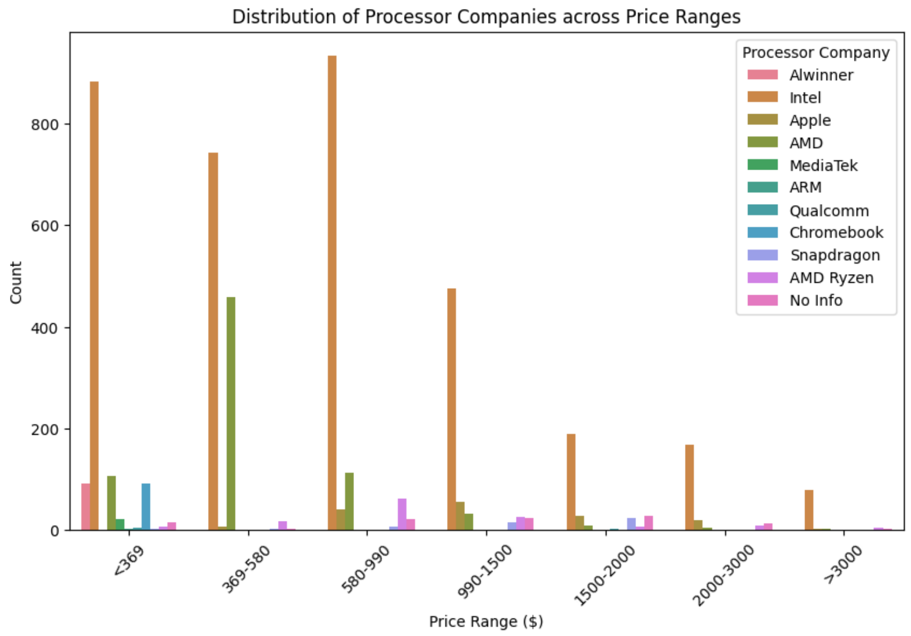


Figure 9: Distribution of processor companies across price ranges.

The plot highlights Intel's dominance across price ranges, followed by Apple in high-end segments. ARM-based processors like MediaTek and Snapdragon target low-end laptops, while Apple is absent in lower price brackets (369 USD). Other brands lack the volume to rival Intel and Apple.

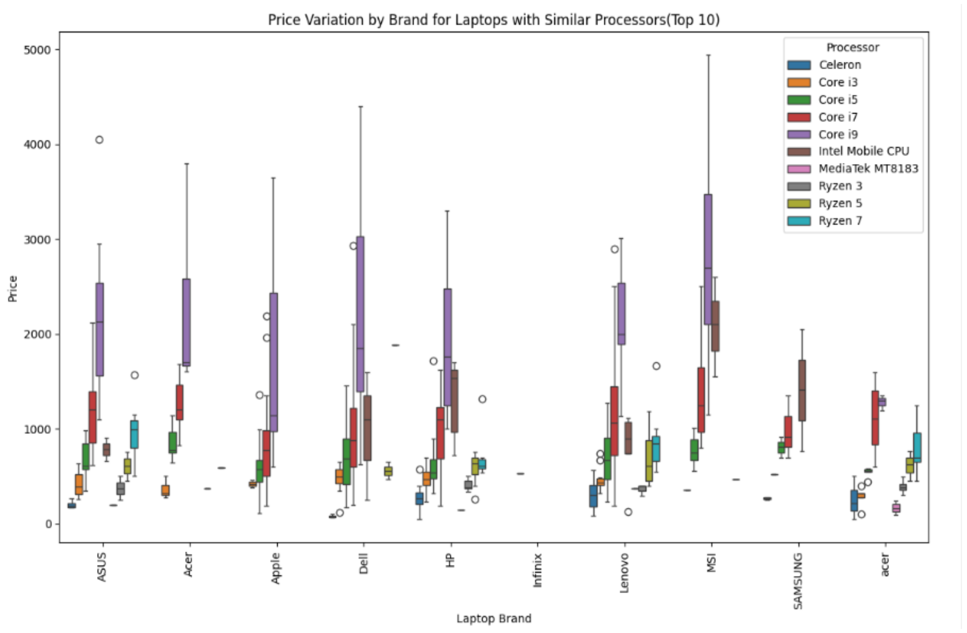


Figure 10: Price variation by brand for laptops with similar processors

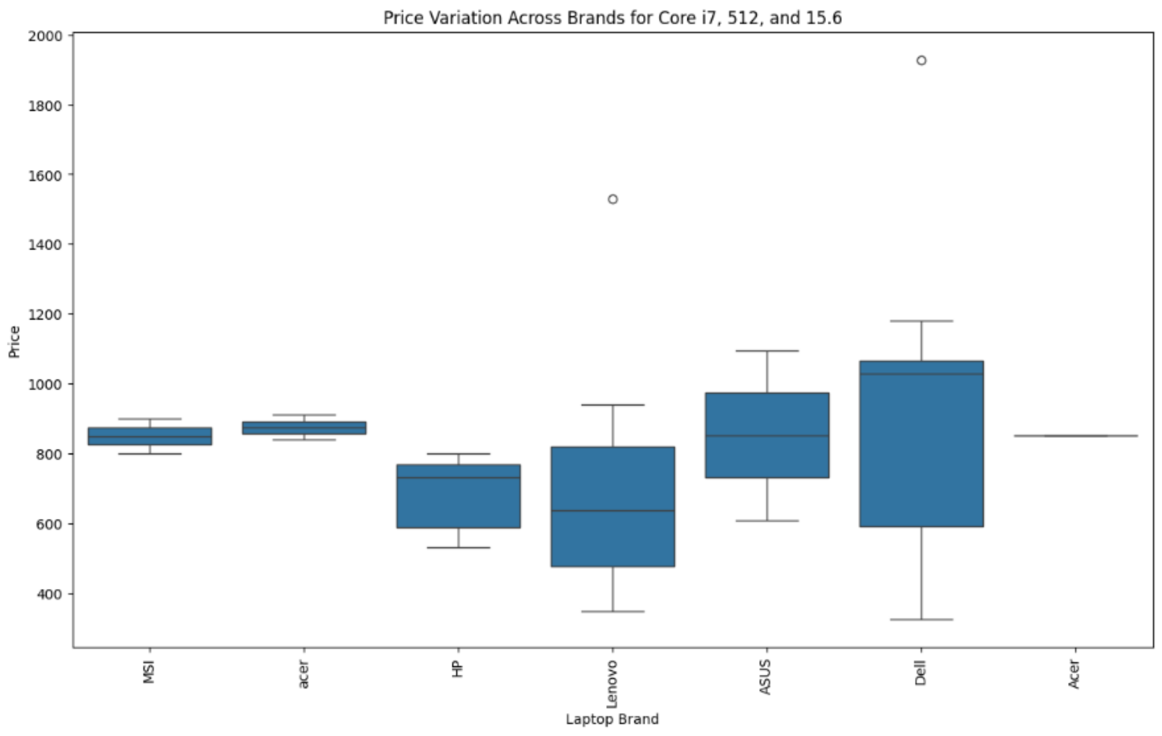


Figure 11: Price variation by brand for core i7,512,and 15.6

Exploratory Data Analysis (EDA) examined features like RAM, price, screen size, and processor types to uncover trends and correlations influencing laptop prices. Visualizations highlighted patterns, outliers, and significant factors, laying the foundation for predictive modeling and insights for decision-making.

4 Model Building

This section outlines the machine learning models used, the rationale for their selection, and the data preprocessing approach.

4.1 Libraries Used

- **numpy**: Handles numerical operations and array manipulations.
- **pandas**: Manages structured data using DataFrames for manipulation and analysis.
- **sqlite3**: Interfaces with SQLite databases for querying and managing data.
- **matplotlib.pyplot**: Creates static visualizations such as line, scatter, and bar plots.
- **seaborn**: Builds statistical data visualizations with attractive plots.
- **sklearn**: Provides tools for preprocessing, training, and evaluating models.

4.2 Preprocessing Pipeline

The preprocessing pipeline handles both categorical and numerical columns. It begins by defining:

- **Categorical Columns**: Includes attributes like Brand, ProcessorBrand, OperatingSystem, and StorageType.
- **Numerical Columns**: Includes ExtractedRating, StorageCapacity(GB), RAM(GB), and Price.

These lists combine into `decidingColumns`, which also includes `Stock`.

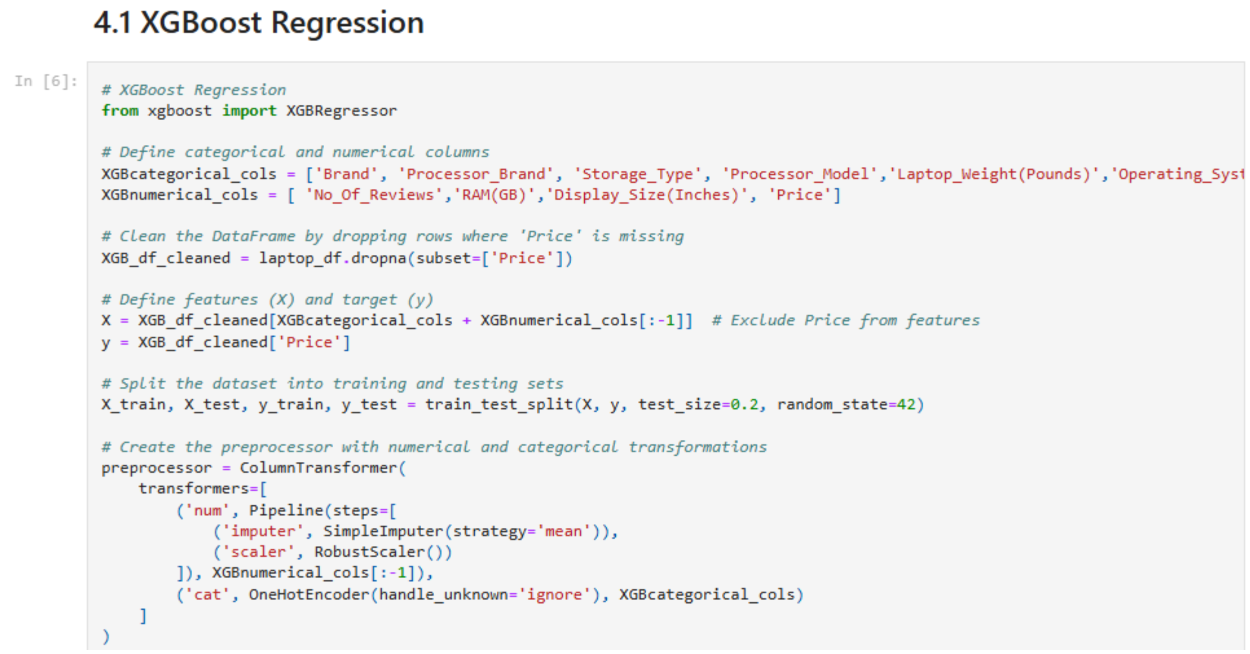
For numerical data:

- **SimpleImputer**: Fills missing values with the column mean.
- **RobustScaler**: Scales the data using median, ensuring robustness to outliers.

For categorical data:

- **OneHotEncoder**: Converts categorical variables into binary columns.

The `ColumnTransformer` applies these transformations, preparing the dataset for model training.



4.1 XGBoost Regression

```
In [6]:
# XGBoost Regression
from xgboost import XGBRegressor

# Define categorical and numerical columns
XGBcategorical_cols = ['Brand', 'Processor_Brand', 'Storage_Type', 'Processor_Model','Laptop_Weight(Pounds)','Operating_Syst
XGBnumerical_cols = [ 'No_Of_Reviews','RAM(GB)','Display_Size(Inches)', 'Price']

# Clean the DataFrame by dropping rows where 'Price' is missing
XGB_df_cleaned = laptop_df.dropna(subset=['Price'])

# Define features (X) and target (y)
X = XGB_df_cleaned[XGBcategorical_cols + XGBnumerical_cols[:-1]]  # Exclude Price from features
y = XGB_df_cleaned['Price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the preprocessor with numerical and categorical transformations
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),
            ('scaler', RobustScaler())
        ]), XGBnumerical_cols[:-1]),
        ('cat', OneHotEncoder(handle_unknown='ignore'), XGBcategorical_cols)
    ]
)
```

Figure 12: Xg Boost Regression sample code

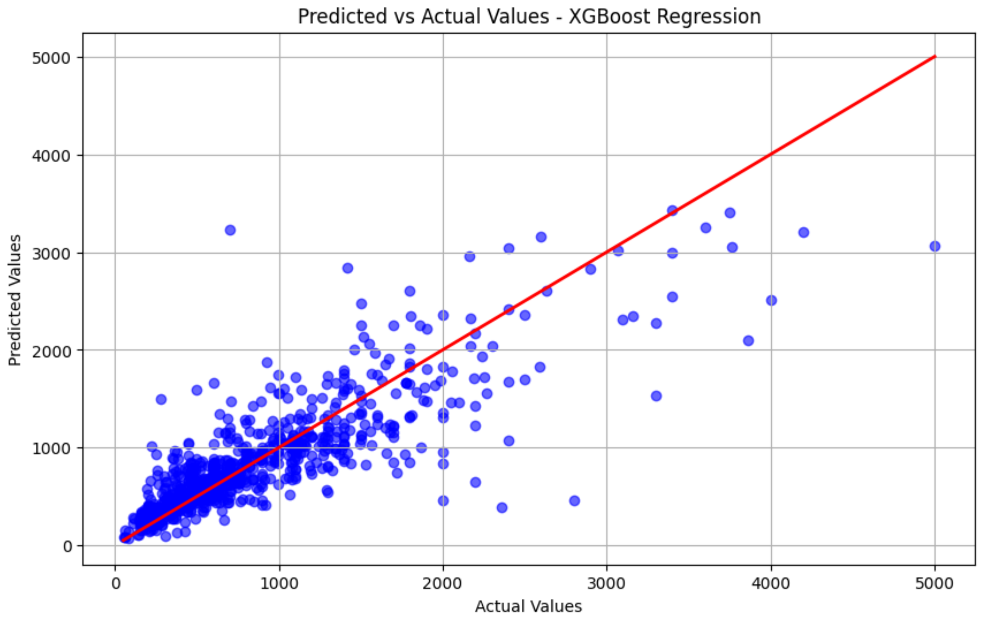
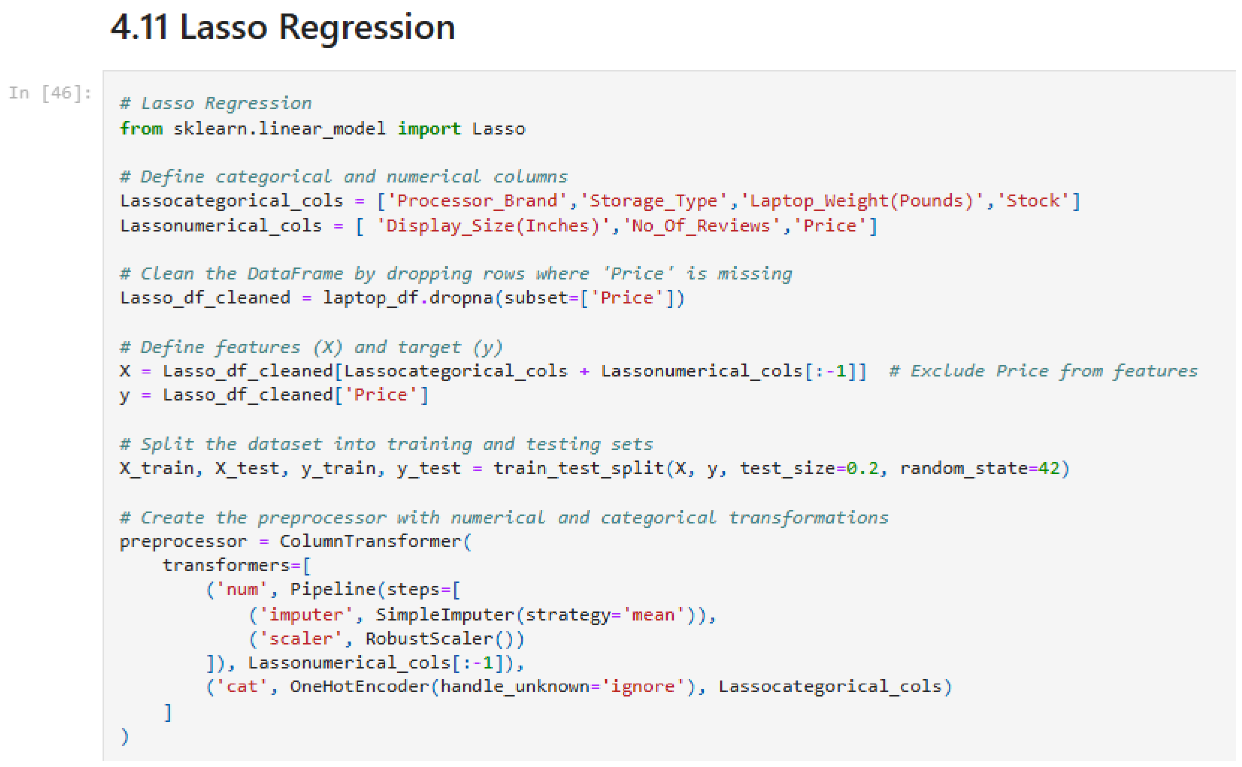


Figure 13: Model Performance

This performance indicates the model has captured meaningful relationships between the input features (RAM, processor company, brand, storage type, OS, display size, weight, stock availability, and reviews) and laptop prices



4.11 Lasso Regression

```
In [46]:
# Lasso Regression
from sklearn.linear_model import Lasso

# Define categorical and numerical columns
Lassocategorical_cols = ['Processor_Brand','Storage_Type','Laptop_Weight(Pounds)','Stock']
Lassonumerical_cols = [ 'Display_Size(Inches)','No_Of_Reviews','Price']

# Clean the DataFrame by dropping rows where 'Price' is missing
Lasso_df_cleaned = laptop_df.dropna(subset=['Price'])

# Define features (X) and target (y)
X = Lasso_df_cleaned[Lassocategorical_cols + Lassonumerical_cols[:-1]]  # Exclude Price from features
y = Lasso_df_cleaned['Price']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the preprocessor with numerical and categorical transformations
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),
            ('scaler', RobustScaler())
        ]), Lassonumerical_cols[:-1]),
        ('cat', OneHotEncoder(handle_unknown='ignore'), Lassocategorical_cols)
    ]
)
```

Figure 14: Lasso Regression sample code

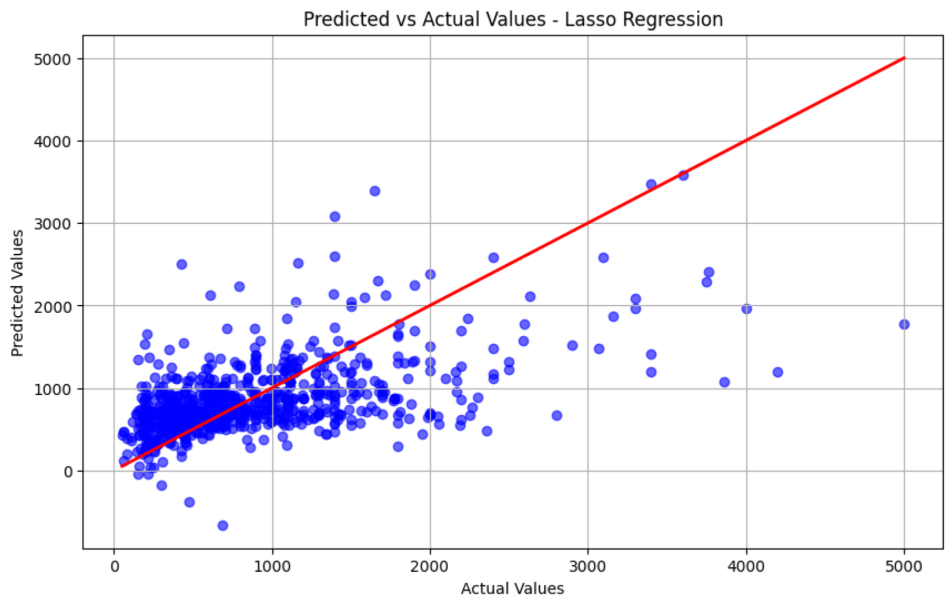


Figure 15: Model Performance

Lasso Regression showed a relatively lower R^2 score, indicating that it struggled to capture the variance in laptop prices effectively.

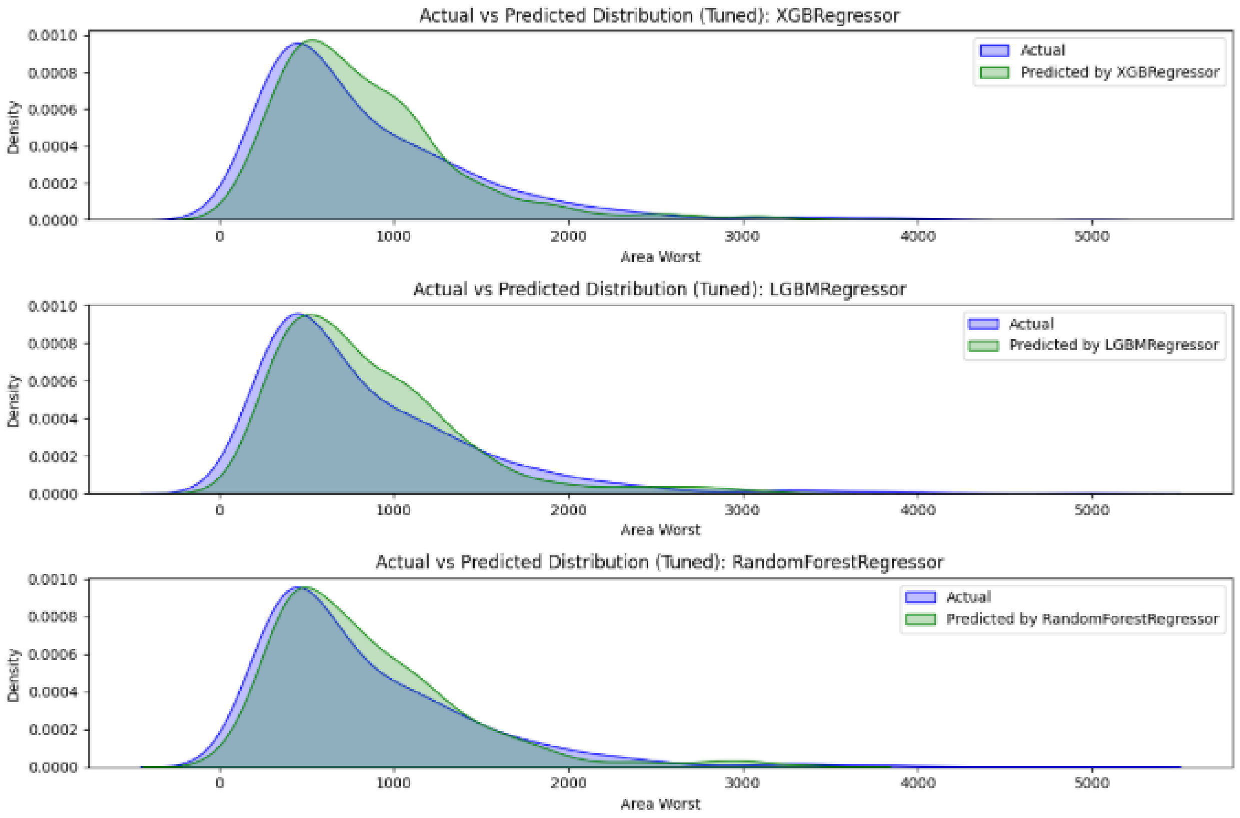


Figure 16: Comparision of model performances among all the models trained.

Final Model Performance Analysis

After hyperparameter tuning, three ensemble models—XGBoost Regressor, LightGBM Regressor, and Random Forest Regressor—were evaluated. XGBoost outperformed the others, followed by LightGBM and Random Forest. The performance comparison is as follows:

- **XGBoost Regressor:** MSE: 109,798.53, R^2: 0.738, Adjusted R^2: 0.735
- **LightGBM Regressor:** MSE: 113,916.90, R^2: 0.728, Adjusted R^2: 0.725
- **Random Forest Regressor:** MSE: 120,103.55, R^2: 0.714, Adjusted R^2: 0.710

Conclusion

XGBoost Regressor was the top performer with an R^2 score of 0.738, indicating strong predictive accuracy. LightGBM followed closely with an R^2 score of 0.728, while Random Forest had a slightly lower performance. All models demonstrated strong generalization, with minimal differences in R^2 and adjusted R^2 scores, ensuring good predictive power without overfitting. XGBoost is the most reliable model for laptop price prediction.

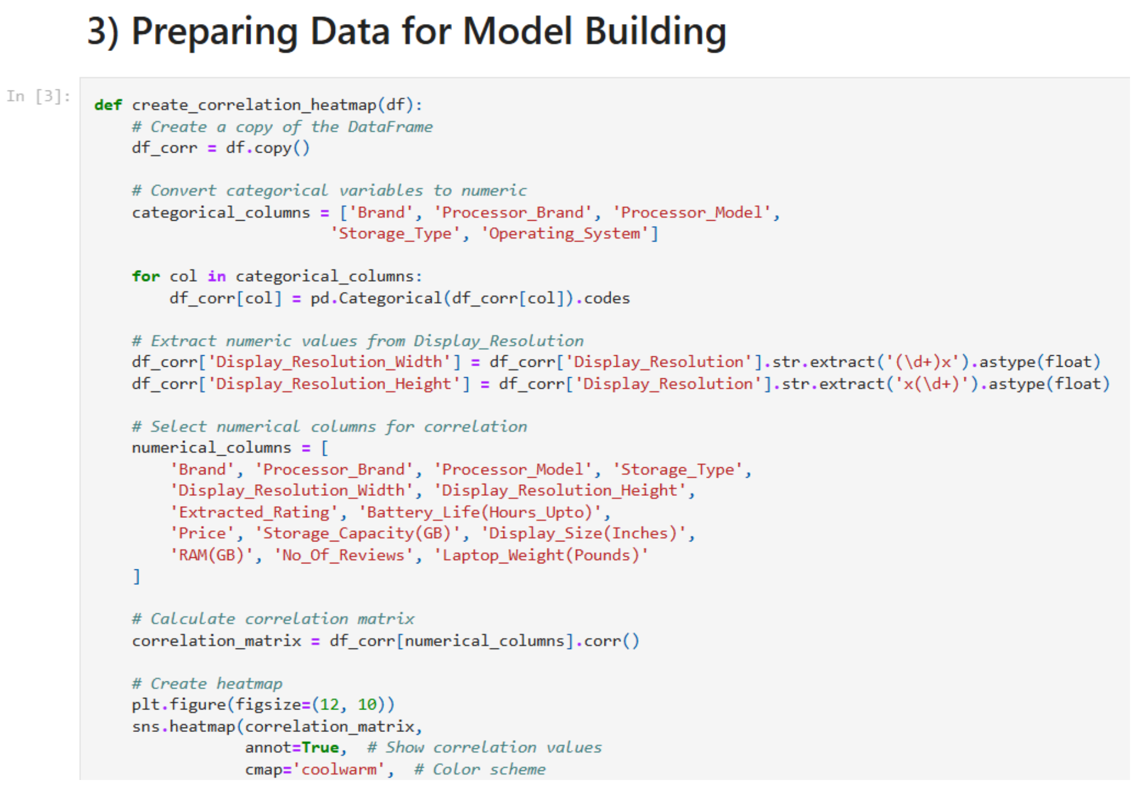
5 Final Models for prediction and recommendation

This section explains the recommendation system developed for providing insights to customers and retailers.

5.1 Final ML Model

5.1.1 Preparing the Data for model Building

Importing Required Modules and Loading Data
We begin by importing essential Python libraries, such as pandas for data manipulation and sqlite3 for database connectivity. After establishing a connection to the SQL database, the required dataset is loaded into a pandas DataFrame for further processing and analysis. This step sets up the foundation for subsequent data analysis tasks.



3) Preparing Data for Model Building

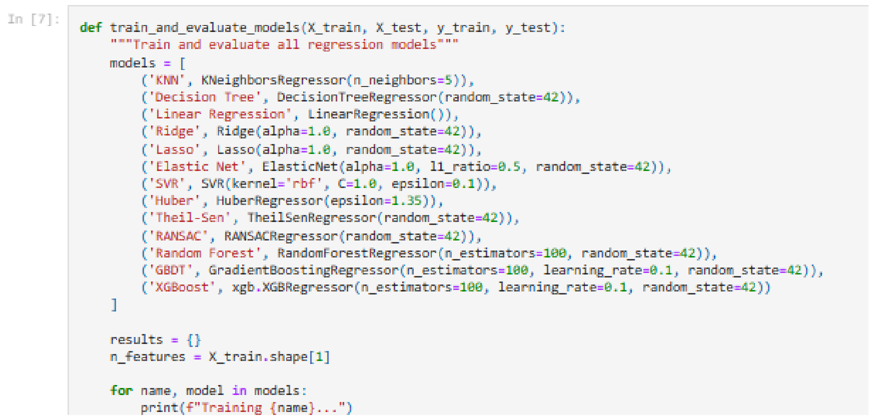
```
In [3]:
def create_correlation_heatmap(df):
    # Create a copy of the DataFrame
    df_corr = df.copy()

    # Convert categorical variables to numeric
    categorical_columns = ['Brand', 'Processor_Brand', 'Processor_Model',
                          'Storage_Type', 'Operating_System']

    for col in categorical_columns:
        df_corr[col] = pd.Categorical(df_corr[col]).codes

    # Extract numeric values from Display_Resolution
    df_corr['Display_Resolution_Width'] = df_corr['Display_Resolution'].str.extract('(\d+)x').astype(float)
    df_corr['Display_Resolution_Height'] = df_corr['Display_Resolution'].str.extract('x(\d+)').astype(float)

    # Select numerical columns for correlation
    numerical_columns = [
        'Brand', 'Processor_Brand', 'Processor_Model', 'Storage_Type',
        'Display_Resolution_Width', 'Display_Resolution_Height',
        'Extracted_Rating', 'Battery_Life(Hours_Upto)',
        'Price', 'Storage_Capacity(GB)', 'Display_Size(Inches)',
        'RAM(GB)', 'No_Of_Reviews', 'Laptop_Weight(Pounds)'
    ]

    # Calculate correlation matrix
    correlation_matrix = df_corr[numerical_columns].corr()

    # Create heatmap
    plt.figure(figsize=(12, 10))
    sns.heatmap(correlation_matrix,
                annot=True,  # Show correlation values
                cmap='coolwarm',  # Color scheme
```

Figure 17: Sample code for Preparing the data for model building

.

This code prepares data for model building by creating a correlation heatmap and identifying features correlated with the target variable, `Price`. Categorical columns are converted to numeric codes, and `Display_Resolution` is split into width and height. The correlation matrix for features like RAM, `Price`, and Battery Life is computed, with correlations printed to highlight significant predictors.

5.1.2 Training and Evaluating Models



```
In [7]:
def train_and_evaluate_models(X_train, X_test, y_train, y_test):
    """Train and evaluate all regression models"""
    models = [
        ('KNN', KNeighborsRegressor(n_neighbors=5)),
        ('Decision Tree', DecisionTreeRegressor(random_state=42)),
        ('Linear Regression', LinearRegression()),
        ('Ridge', Ridge(alpha=1.0, random_state=42)),
        ('Lasso', Lasso(alpha=1.0, random_state=42)),
        ('Elastic Net', ElasticNet(alpha=1.0, l1_ratio=0.5, random_state=42)),
        ('SVR', SVR(kernel='rbf', C=1.0, epsilon=0.1)),
        ('Huber', HuberRegressor(epsilon=1.35)),
        ('Theil-Sen', TheilSenRegressor(random_state=42)),
        ('RANSAC', RANSACRegressor(random_state=42)),
        ('Random Forest', RandomForestRegressor(n_estimators=100, random_state=42)),
        ('GBDT', GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)),
        ('XGBoost', xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42))
    ]

    results = {}
    n_features = X_train.shape[1]

    for name, model in models:
        print(f"Training {name}...")
```

Figure 18: Sample model training code.

The `train_and_evaluate_models` function trains and evaluates multiple regression models (e.g., KNN, Decision Tree, Linear Regression). It fits each model to training data, makes predictions on the test data, and evaluates them using predefined metrics. Results are stored in a DataFrame for easy comparison.

Model	**MSE**	**RMSE**	**R2**	**Adjusted R2**
XGBoost	**1.180690e+05**	**343.611718**	**0.718419**	**0.713987**
Random Forest	**1.239727e+05**	**352.097569**	**0.704339**	**0.699686**
GBDT	**1.447133e+05**	**380.412035**	**0.654875**	**0.649444**
KNN	2.310375e+05	480.663600	0.449002	0.440330
Decision Tree	2.539057e+05	503.890601	0.394464	0.384934
Elastic Net	3.350358e+05	578.822769	0.200978	0.188403

Table 1: Model Evaluation Results

5.1.3 Hyperparameter Tuning on Top 3 Models

Hyperparameter tuning was applied to the top three performing models: XGBoost, Random Forest, and GBDT. These models showed the best R^2 and MSE values among the initial evaluations.

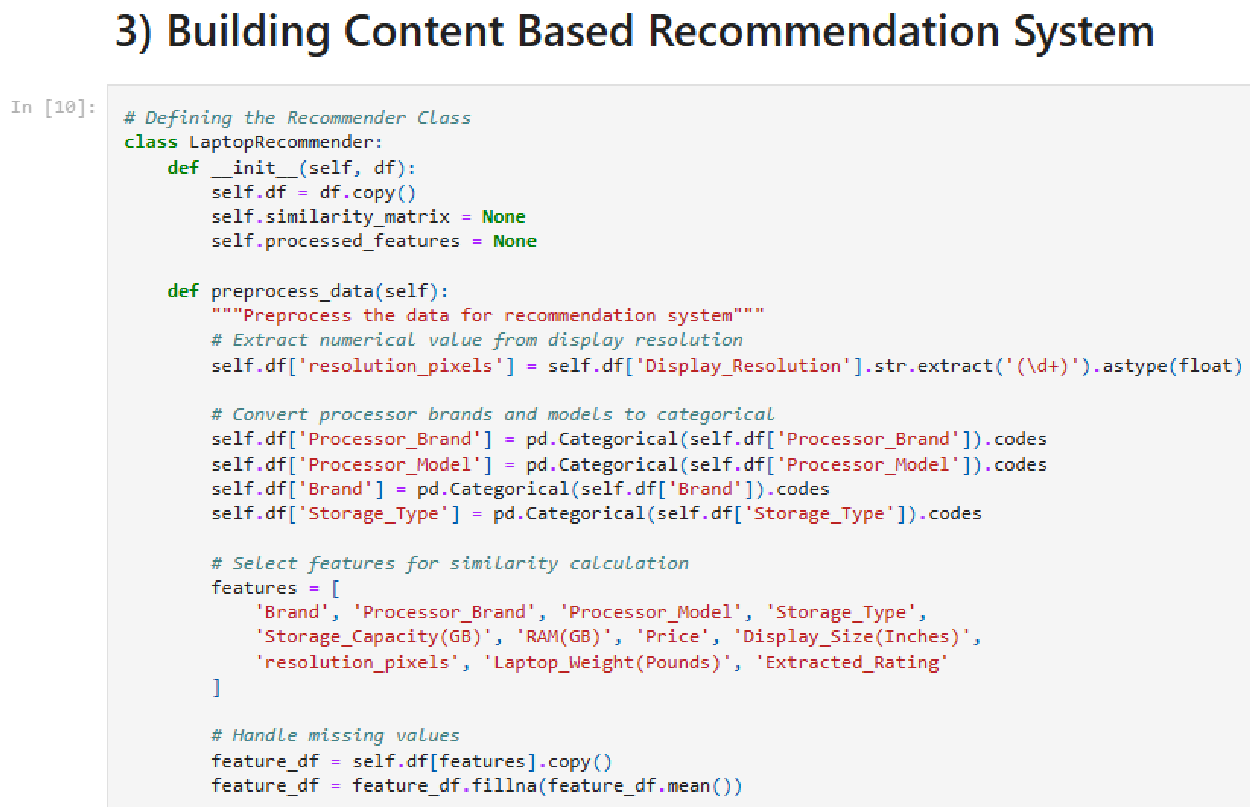
Model	**MSE**	**RMSE**	**R2**	**Adjusted R2**
XGBoost	100303.9623	316.7080	0.7608	0.7570
Random Forest	113862.8698	337.4357	0.7285	0.7242
GBDT	101470.4289	318.5442	0.7580	0.7542

Table 2: Tuned Model Evaluation Results

XGBoost outperforms the other models with the lowest MSE and RMSE, and the highest R^2 and Adjusted R^2, indicating superior prediction accuracy and model fit.

5.2 Recommendation System

Updated data columns and data collection were revisited to enhance the dataset with new attributes and improve the analysis process.



3) Building Content Based Recommendation System

```
In [10]:
# Defining the Recommender Class
class LaptopRecommender:
    def __init__(self, df):
        self.df = df.copy()
        self.similarity_matrix = None
        self.processed_features = None

    def preprocess_data(self):
        """Preprocess the data for recommendation system"""
        # Extract numerical value from display resolution
        self.df['resolution_pixels'] = self.df['Display_Resolution'].str.extract('(\d+)').astype(float)

        # Convert processor brands and models to categorical
        self.df['Processor_Brand'] = pd.Categorical(self.df['Processor_Brand']).codes
        self.df['Processor_Model'] = pd.Categorical(self.df['Processor_Model']).codes
        self.df['Brand'] = pd.Categorical(self.df['Brand']).codes
        self.df['Storage_Type'] = pd.Categorical(self.df['Storage_Type']).codes

        # Select features for similarity calculation
        features = [
            'Brand', 'Processor_Brand', 'Processor_Model', 'Storage_Type',
            'Storage_Capacity(GB)', 'RAM(GB)', 'Price', 'Display_Size(Inches)',
            'resolution_pixels', 'Laptop_Weight(Pounds)', 'Extracted_Rating'
        ]

        # Handle missing values
        feature_df = self.df[features].copy()
        feature_df = feature_df.fillna(feature_df.mean())
```

Figure 19: Sample Building Recommendation code

.

The `LaptopRecommender` class provides personalized laptop recommendations based on similarity to other laptops or user-defined features. Initialized with a laptop dataset, it includes preprocessing methods such as extracting numerical values from `Display_Resolution`, converting categorical features into numerical codes, and scaling features using `MinMaxScaler`. The `get_recommendations` method calculates cosine similarity between laptops and returns the most similar ones, while `get_recommendations_by_features` allows users to input preferences like `RAM` or `Storage_Capacity` for targeted recommendations.

Using content-based filtering, the system suggests laptops based on features like brand, processor, RAM, storage, display resolution, and price. Cosine similarity helps identify similar laptops or recommend options based on specific preferences.

To enhance recommendation accuracy, the system preprocesses data by encoding categorical variables, handling missing values, and scaling features, ensuring uniformity across attributes. It also provides similarity scores, showing how closely the recommended laptops match user preferences. These techniques ensure effective and personalized laptop recommendations, making the system a valuable tool for laptop shoppers.

6 App Building

Here, we describe the web application developed to present the analysis results to users and admins.

The key features :

- **Buying Options:** Users can see different buying options for the selected product ie across different soruces (Amazon,BestBuy and Flipkart).
- **Similar Products Recommendations:** Alongside product browsing, the app shows products with similar features to help users make informed decisions.
- **Admin Panel:** Admins have control over the laptop database, enabling them to update features, prices, and specifications. Admins can also get price analysis graphs and get price predictions.

6.1 Technology Stack and Workflow

1. **Backend and ETL:**
    - **PySpark:** Used for scraping, cleaning, and combining large datasets of laptop features.
    - **PostgreSQL:** A relational database to store the processed and structured data for laptops and recommendations.
2. **ETL Pipeline:**
    - **Scraping:** Extract laptop data from various sources.
    - **Data Cleaning and Combining:** Use PySpark to preprocess and integrate the data.
    - **Storage:** Load the cleaned data into a PostgreSQL database.
3. **Backend API:**
    - **Flask:** Backend framework to create APIs for serving data to the frontend.
    - APIs include:
        - Fetching laptop recommendations based on user input.
        - Displaying similar products for users.
        - Admin functionalities for updating the database.
4. **Frontend:**
    - **ReactJS:** Frontend framework for building a responsive and user-friendly interface.
        - Allows users to input preferences and view recommendations.
        - Displays similar laptops and admin-controlled updates.

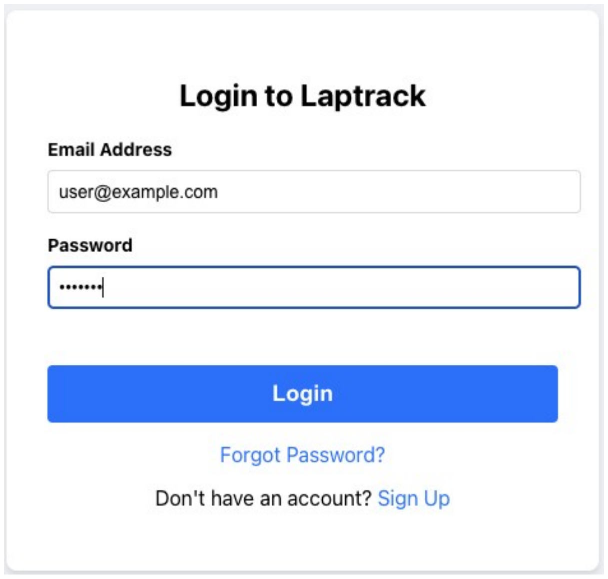


Figure 20: User Login

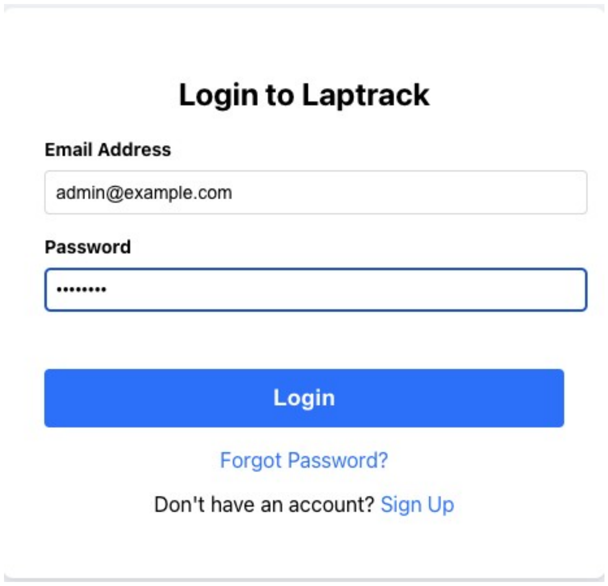


Figure 21: Admin Login

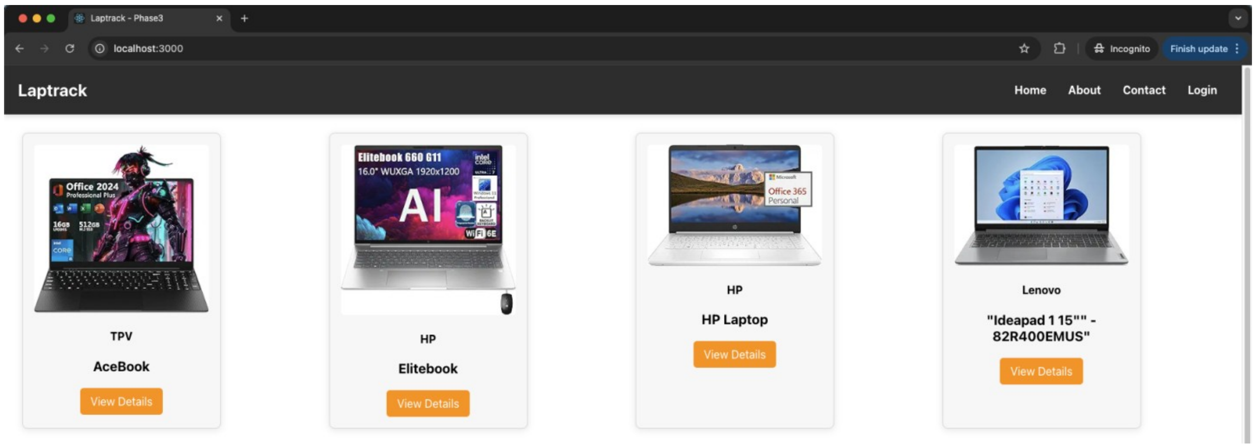
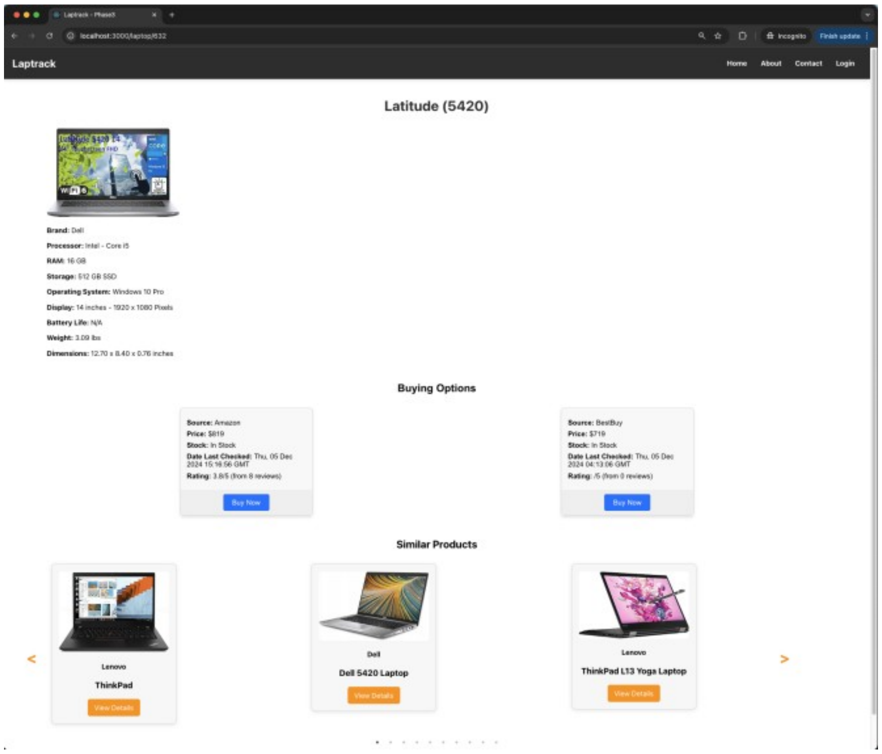


Figure 22: User home page showing laptops

.



Latitude (5420)

Brand: Dell

Processor: Intel - Core i5

RAM: 16 GB

Storage: 512 GB SSD

Operating System: Windows 10 Pro

Display: 14 inches - 1920 x 1080 Pixels

Battery Life: N/A

Weight: 3.09 lbs

Dimensions: 12.70 x 8.40 x 0.76 inches

Buying Options

Source: Amazon
Price: $819
Stock: In Stock
Date Last Checked: Thu, 05 Dec 2024 15:16:56 GMT
Rating: 3.8/5 (from 8 reviews)

Source: BestBuy
Price: $719
Stock: In Stock
Date Last Checked: Thu, 05 Dec 2024 04:13:06 GMT
Rating: /5 (from 0 reviews)

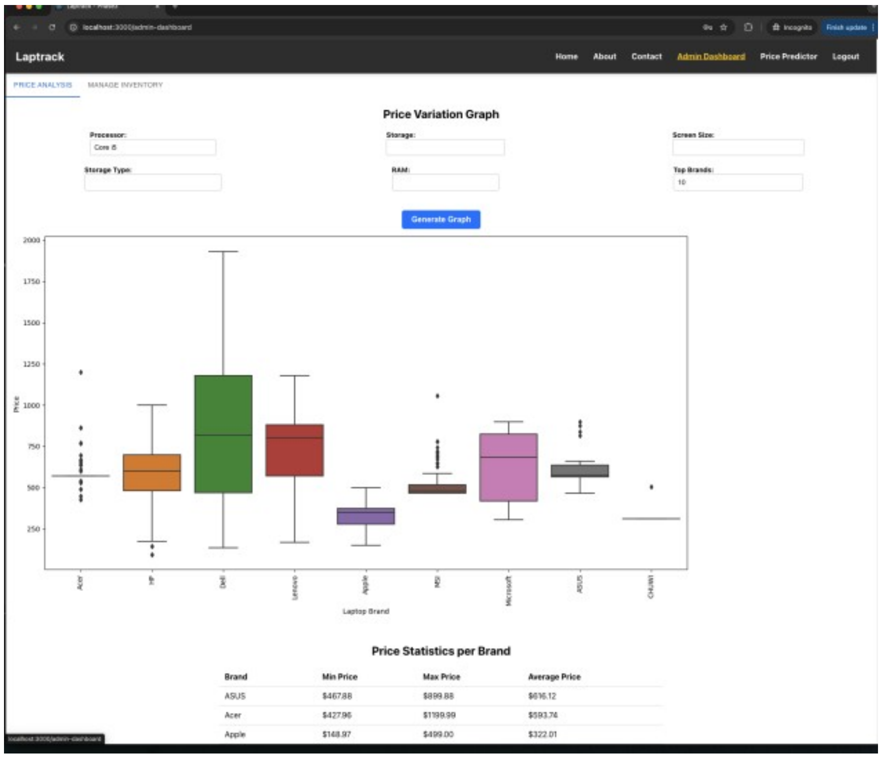
Similar Products

Lenovo
ThinkPad

Dell
Dell 5420 Laptop

Lenovo
ThinkPad L13 Yoga Laptop

Figure 23: Product page showing buying options, similar products.



Price Variation Graph

Processor: Core i5

Storage:

Screen Size:

Storage Type:

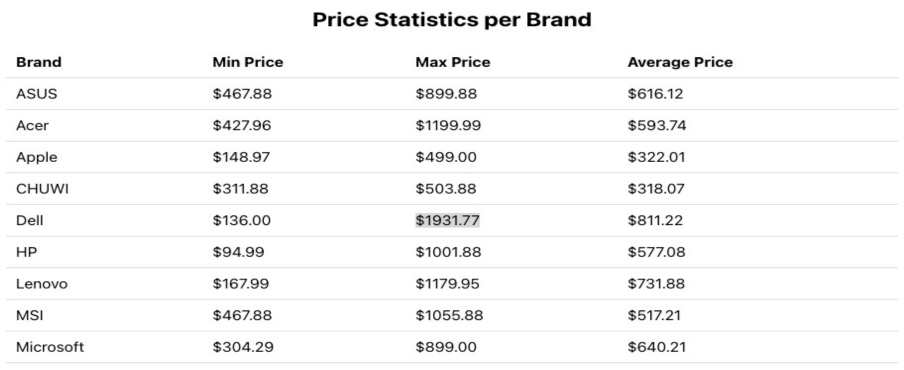
RAM:

Top Brands: 10

Price Statistics per Brand

Brand	Min Price	Max Price	Average Price
ASUS	$467.88	$899.88	$616.12
Acer	$427.96	$1199.99	$593.74
Apple	$148.97	$499.00	$322.01

Figure 24: Admin Home page.

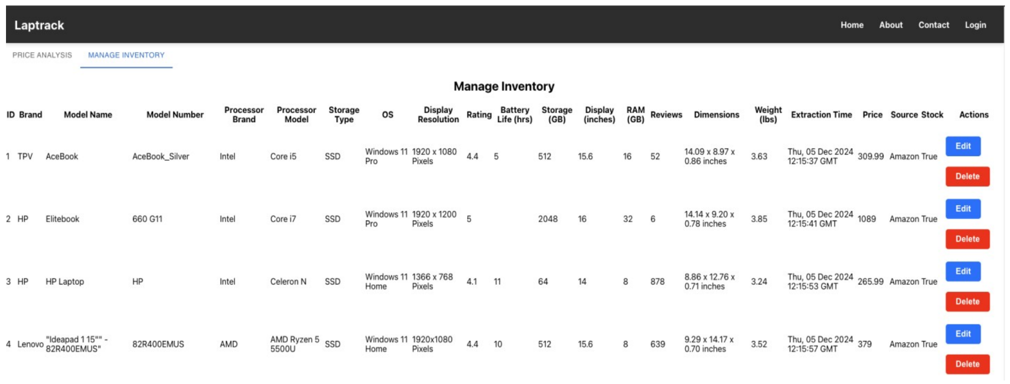


Price Statistics per Brand

Brand	Min Price	Max Price	Average Price
ASUS	$467.88	$899.88	$616.12
Acer	$427.96	$1199.99	$593.74
Apple	$148.97	$499.00	$322.01
CHUWI	$311.88	$503.88	$318.07
Dell	$136.00	$1931.77	$811.22
HP	$94.99	$1001.88	$577.08
Lenovo	$167.99	$1179.95	$731.88
MSI	$467.88	$1055.88	$517.21
Microsoft	$304.29	$899.00	$640.21

Figure 25: Price variation table

.

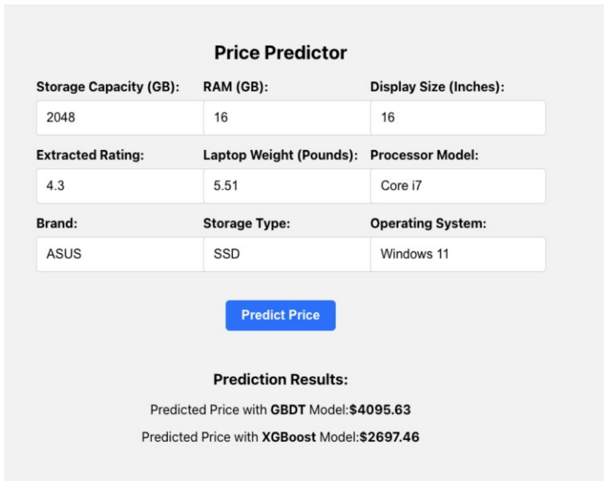


Manage Inventory

ID	Brand	Model Name	Model Number	Processor Brand	Processor Model	Storage Type	OS	Display Resolution	Rating	Battery Life (hrs)	Storage (GB)	Display (inches)	RAM (GB)	Reviews	Dimensions	Weight (lbs)	Extraction Time	Price	Source	Stock	Actions
1	TPV	AceBook	AceBook_Silver	Intel	Core i5	SSD	Windows 11 Pro	1920 x 1080 Pixels	4.4	5	512	15.6	16	52	14.09 x 8.97 x 0.86 inches	3.63	Thu, 05 Dec 2024 12:15:37 GMT	309.99	Amazon	True	Edit Delete
2	HP	Elitebook	660 G11	Intel	Core i7	SSD	Windows 11 Pro	1920 x 1200 Pixels	5		2048	16	32	6	14.14 x 9.20 x 0.78 inches	3.85	Thu, 05 Dec 2024 12:15:41 GMT	1089	Amazon	True	Edit Delete
3	HP	HP Laptop	HP	Intel	Celeron N	SSD	Windows 11 Home	1366 x 768 Pixels	4.1	11	64	14	8	878	8.86 x 12.76 x 0.71 inches	3.24	Thu, 05 Dec 2024 12:15:53 GMT	265.99	Amazon	True	Edit Delete
4	Lenovo	"Ideapad 1 15"" - 82R400EMUS"	82R400EMUS	AMD	AMD Ryzen 5 5500U	SSD	Windows 11 Home	1920x1080 Pixels	4.4	10	512	15.6	8	639	9.29 x 14.17 x 0.70 inches	3.52	Thu, 05 Dec 2024 12:15:57 GMT	379	Amazon	True	Edit Delete

Figure 26: Database edit page for admin

.



Price Predictor

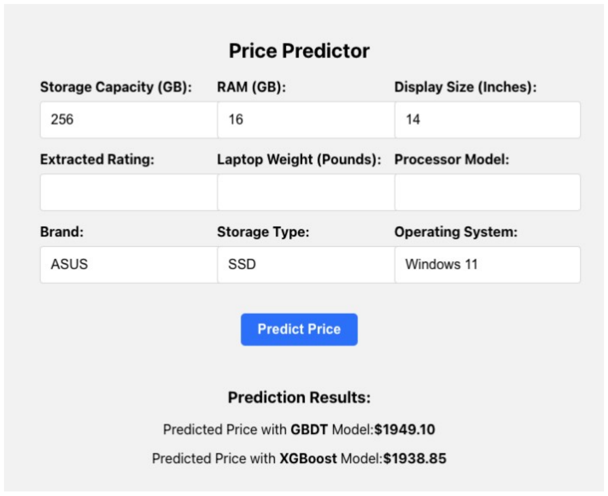
Storage Capacity (GB):	RAM (GB):	Display Size (Inches):
2048	16	16
Extracted Rating:	Laptop Weight (Pounds):	Processor Model:
4.3	5.51	Core i7
Brand:	Storage Type:	Operating System:
ASUS	SSD	Windows 11

Prediction Results:

Predicted Price with **GBDT** Model:**$4095.63**

Predicted Price with **XGBoost** Model:**$2697.46**

Figure 27: Price Prediction Example 1



Price Predictor

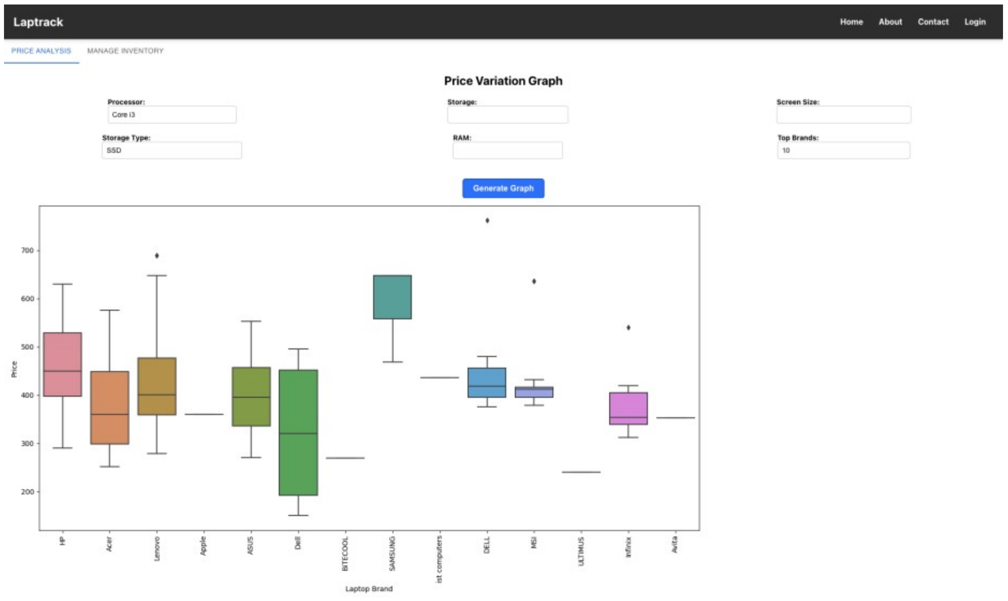
Storage Capacity (GB):	RAM (GB):	Display Size (Inches):
256	16	14
Extracted Rating:	Laptop Weight (Pounds):	Processor Model:
Brand:	Storage Type:	Operating System:
ASUS	SSD	Windows 11

Prediction Results:

Predicted Price with **GBDT** Model:**$1949.10**

Predicted Price with **XGBoost** Model:**$1938.85**

Figure 28: Price Prediction Example 2



Processor: Core i3

Storage:

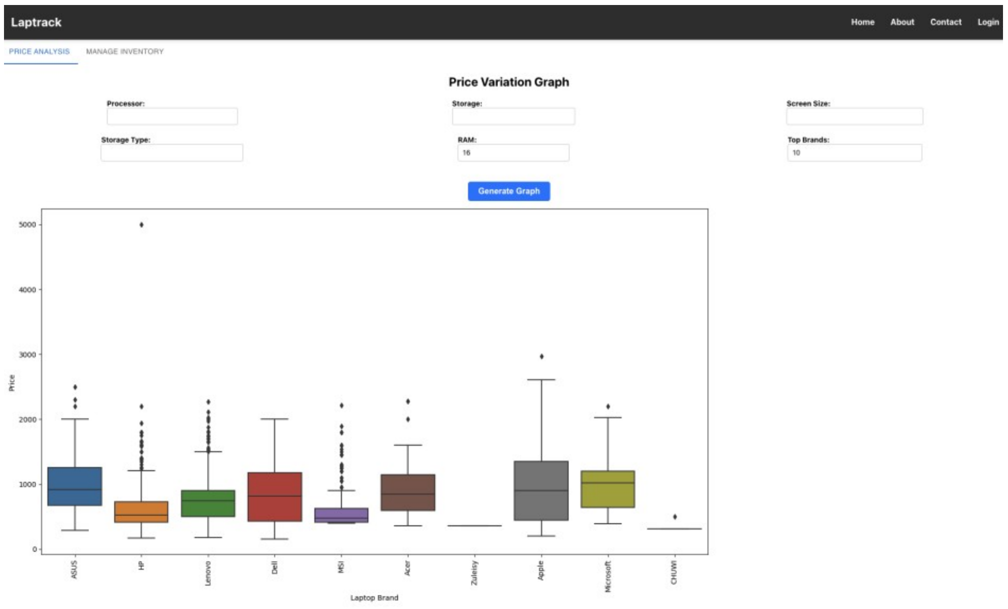
Screen Size:

Storage Type: SSD

RAM:

Top Brands: 10

Figure 29: Plot for price analysis across brands for Core i3 and SSD Storage type



Processor:

Storage:

Screen Size:

Storage Type:

RAM: 16

Top Brands: 10

Figure 30: Plot for price analysis across brands for 16 GB RAM

7 Conclusion

This application analyzes laptop data scraped from Amazon, Flipkart, and BestBuy, focusing on key features like processor type, RAM size, storage capacity, screen size, and brand reputation that influence pricing. The data was processed using web scraping techniques to gather pricing, specifications, customer ratings, and reviews.

Various machine learning regression models, including decision trees, random forests, and linear regression, were applied to predict laptop prices. The best-performing model was selected to build a robust predictive system that estimates laptop prices based on real-time data.

The analysis aims to understand how laptop features impact pricing across different retail platforms and identify factors contributing to price variations. This helps consumers make informed decisions based on budget and specifications, while retailers can optimize pricing strategies and tailor product offerings.

A web application was developed to display the dataset for admin users and provide real-time price predictions to customers, powered by a PySpark backend for efficient data processing. In conclusion, this application assists both consumers in choosing laptops and retailers in adjusting pricing strategies.

8 References

References

@miscreact, author = Meta Platforms, Inc., title = React Documentation, howpublished = `https://react.dev/`, note = Accessed: 2024-12-07

@miscpyspark, author = Apache Software Foundation, title = PySpark Documentation, howpublished = `https://spark.apache.org/docs/latest/api/python/index.html`, note = Accessed: 2024-12-07

@miscpostgresql, author = PostgreSQL Global Development Group, title = PostgreSQL Documentation, howpublished = `https://www.postgresql.org/docs/`, note = Accessed: 2024-12-07

@miscflask, author = Pallets Projects, title = Flask Documentation - Quickstart, howpublished = `https://flask.palletsprojects.com/en/stable/quickstart/#routing`, note = Accessed: 2024-12-07

@miscselenium, author = Selenium Community, title = Selenium Documentation, howpublished = `https://www.selenium.dev/documentation/`, note = Accessed: 2024-12-07

@miscxgboost, author = XGBoost Developers, title = XGBoost Documentation, howpublished = `https://xgboost.readthedocs.io/en/stable/`, note = Accessed: 2024-12-07