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 focuses on a comprehensive analysis of laptops across multiple e-commerce platforms, specifically Amazon, Flipkart, and BestBuy, to gain a deeper understanding of the factors that influence their pricing and the specifications that significantly impact customer purchasing decisions. The primary objective is to identify key features that not only drive the pricing strategies of laptops but also shape customer preferences, ultimately guiding both sellers and buyers in making well-informed decisions.

In recent years, the laptop market has become highly competitive, with numerous brands and models offering varying specifications at different price points. This has led to an increasing complexity for consumers in making informed purchasing choices. By analyzing extensive data from e-commerce platforms, this project aims to uncover the most critical factors influencing laptop prices, such as processor type, storage capacity, screen size, RAM, and brand reputation. Additionally, the study focuses on identifying the features that customers prioritize when selecting a laptop, such as battery life, display quality, and processing power.

Understanding the relationship between these factors and laptop pricing is crucial for sellers aiming to optimize their product offerings. Retailers can use the insights gained from this analysis to adjust their pricing strategies, ensuring that they remain competitive in the market while meeting the demands of customers. For instance, if certain specifications, such as larger storage or more RAM, are found to significantly increase the value of a laptop in the eyes of customers, sellers can price these laptops accordingly to reflect this perceived value. Moreover, understanding customer preferences helps sellers in developing marketing strategies that highlight the features most likely to attract buyers.

On the other hand, this analysis also benefits consumers by providing them with a clearer understanding of which features matter most within their price range. By offering insights into the laptop features that contribute to higher prices and those that provide the most value for money, buyers can make better purchasing decisions, ultimately ensuring that they get the best product suited to their needs.

Data for this study was sourced from leading e-commerce platforms, including Amazon, Flipkart, and BestBuy, which provide a wide range of laptop models from various manufacturers. This diverse dataset allows for a thorough comparison across different brands, pricing strategies, and customer reviews, providing a holistic view of the laptop market.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Statement

The rapid advancement of technology has resulted in a vast array of laptops available to consumers, each offering unique combinations of specifications, features, and price points. In this competitive market, understanding the factors that influence laptop pricing, as well as the specifications that customers prioritize, has become essential for both sellers and buyers. With so many options to choose from, it can be overwhelming for customers to decide which laptop meets their needs while fitting within their budget. Similarly, sellers must determine the optimal pricing strategies to attract customers and remain competitive in the marketplace.

This project aims to address these challenges by analyzing laptops listed on popular e-commerce platforms such as Amazon, Flipkart, and BestBuy. Through this analysis, we aim to gain a deeper understanding of the key features that significantly impact laptop pricing and customer purchasing decisions. By identifying these factors, we can provide insights that will help both buyers and sellers make more informed decisions. Specifically, the objectives of this analysis are as follows:

- **Identify the key features that influence laptop prices:** By examining various specifications such as processor type, RAM size, storage capacity, screen size, and brand, we aim to understand how each of these features contributes to the overall price of the laptop. This will allow us to uncover the factors that retailers consider when setting prices and how these features vary across different platforms.

- **Understand customer priorities in laptop specifications:** Every consumer has different needs when it comes to choosing a laptop, whether it is for gaming, productivity, or general use. This analysis will highlight the specifications that are most important to customers based on their preferences, reviews, and feedback. Understanding what features attract customers will provide valuable information for manufacturers to design better products.

- **Investigate how features influence customer purchase decisions:** Beyond the technical specifications, the appearance, brand, and overall impression of a laptop also play crucial roles in consumer choices. We will analyze how these factors create a lasting impression on customers, influencing their final purchasing decision.

1.2 Objectives and Significance

The primary goal of this project is to provide valuable insights for both sellers and customers in the laptop market. By leveraging data from multiple e-commerce platforms, we seek to address several key objectives that will benefit both parties:

- **For Sellers:** This project aims to equip sellers with a deeper understanding of customer preferences and the features that attract the most attention. By identifying the key drivers behind purchasing decisions, sellers will be able to refine their pricing

strategies and product offerings to better align with market demand. This knowledge will enable them to create laptops that resonate with customer needs, ultimately improving sales performance and customer satisfaction.

- **For Customers:** On the other hand, customers can benefit from this analysis by making more informed decisions when selecting laptops. By understanding the essential features that fit within their desired price range, customers can optimize their purchasing decisions. The insights provided in this project will help customers compare laptops across platforms, ensuring they get the best value for their money while meeting their specific needs.

This project aims to understand the factors influencing laptop pricing and consumer preferences across different e-commerce platforms. By examining key specifications like RAM, storage, processor, and screen size, we identify features that impact prices and consumer decisions. Analyzing customer reviews will help pinpoint which specifications resonate most with buyers, providing insights for manufacturers to optimize product configurations.

Based on the data, we create a recommendation engine that suggests laptops to buyers according to their preferences (e.g., budget, processor, RAM), incorporating customer ratings and reviews. This engine helps consumers find the best value within their price range.

By comparing laptop prices across platforms, we uncover insights into platform-specific pricing and marketing strategies. This allows brands and retailers to refine offerings, better meet consumer demands, and increase sales.

A personalized recommendation engine enhances the consumer experience by suggesting the best laptop options, reducing decision fatigue, and improving sales for brands. Retailers can also use insights to optimize pricing and promotional strategies. This project provides valuable insights into pricing, consumer behavior, and a practical tool for improving the laptop buying experience for all stakeholders.

This analysis will also have broader implications for the e-commerce industry, as it will help retailers and brands better understand the relationship between laptop features, customer preferences, and pricing. By applying this knowledge, sellers can improve their product designs and adjust pricing strategies to meet the evolving demands of consumers. On the other hand, customers will have access to detailed information that can guide them in selecting laptops that offer the best combination of features, performance, and value within their budget. Thus, this project aims to bridge the gap between consumer expectations and market offerings, benefiting both buyers and sellers in the process.

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 included product details, specifications, and customer reviews from various e-commerce platforms.

Data Structuring

We transformed the unstructured data into a structured format, organizing key attributes such as brand, model, price, screen size, and more into specific columns for easy analysis.

Data Cleaning

The data was cleaned and pre-processed to ensure its quality:

- **Removing Unnecessary Information:** Filters were applied to remove irrelevant or redundant data.
- **Duplicate Removal:** Any duplicate entries in the dataset were identified and eliminated.
- **Missing Data Handling:** We handled missing values by either filling them with appropriate values or removing affected rows.

Data Export

Once the data was cleaned and structured, it was exported as a CSV file for further analysis and modeling.

Data Format

- **Granularity:** Each row represents a unique laptop product.
- **Data Type:** The dataset is granular, with no coarse or aggregated data.

Modules used

- **requests module:** This module is used to send HTTP requests to a URL to fetch the data from web pages. It allows us to interact with websites programmatically and retrieve the HTML content for further processing.

1. Importing Modules

In [1]:

```
import requests
from bs4 import BeautifulSoup
import re
import numpy as np
import pandas as pd
import time
import random
```

Figure 1: Modules Used

- **BeautifulSoup:** BeautifulSoup is a Python class used to parse and extract data from HTML documents. By creating a BeautifulSoup object, we can navigate the HTML structure and extract useful data such as product names, prices, specifications, etc.
- **re module:** The `re` module is used to apply regular expression patterns for filtering and extracting specific pieces of information from the raw HTML data. We use this module to match patterns in text (like phone numbers, prices, etc.) and organize the data into a structured format.
- **numpy and pandas modules:** The `numpy` and `pandas` libraries are essential for handling and manipulating large datasets. `pandas` is used for creating dataframes, cleaning, and transforming data, while `numpy` is used for performing numerical operations and handling arrays efficiently.
- **time module:** The `time` module is used to create time delays during the scraping process. This is important to avoid overwhelming the web server with too many requests in a short period and to comply with the website's usage policies.
- **random module:** The `random` module is used to generate random numbers that can be used to introduce random time delays during the scraping process. These delays make the scraping more natural and reduce the likelihood of getting blocked by the website.

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 2: Flipkart Scraping Code

This Python script is designed to scrape data from a series of web pages, specifically 68 pages in this case. The core functionality begins with initializing the number of pages to scrape (total pages = 68), a counter (i = 1) to track the current page being scraped, and an empty list (raw text) to store the raw HTML content from each page. The script loops through each page in the specified range using a for loop, dynamically generating the URL for each page request. To ensure the scraping process mimics human behavior, a random time delay between 5 and 10 seconds is introduced using random.randint(5, 10). This helps in avoiding detection by anti-scraping measures and reduces the risk of overwhelming the server with rapid requests. The HTTP request is sent using the requests.get() method, where the URL and a set of headers (request header) are passed to simulate a legitimate browser request.

In case of a failed request, the script uses a while loop to repeatedly attempt to fetch the page at random time intervals until a successful response (status code 200) is received. Upon success, a confirmation message is printed, and the raw HTML is added to the rawtext list for later processing. A random delay is introduced between requests to prevent server overload and avoid triggering anti-bot mechanisms. This ensures the script mimics human behavior with natural pauses, and by the end, all the raw HTML data from 68 pages is stored for further analysis.

In [14]:

```
processor = []
processor_company = []
for page in pages:
    soup = BeautifulSoup(page)
    for i in soup.find_all("div",class_="KzDlHZ"):

        # Regex to find the processor company of the laptop
        regex1 = re.findall("Intel|intel|AMD|M1|M2|M3|Chromebook|Snapdragon",str(i.text))
        if regex1:
            processor_company.append(regex1[0])
        else:
            processor_company.append(np.nan)

        # Regex to find the exact processor in the laptop
        regex2 = re.findall("(?:Intel|intel|AMD|M1|M2|M3|Chromebook|Snapdragon)\s(.+) - ",str(i.text))
        if regex2:
            processor.append(regex2[0])
        else:
            processor.append(np.nan)
```

Figure 3: Converting unstructured data to structured data using regex.

In [40]:

```
# Removing extra characters from the price column
laptop_df["Price"] = laptop_df["Price"].str.replace(',','').str.replace('₹', '')
laptop_df.head()
```

Figure 4: Data cleaning.

In [61]:

```
# Taking a final look at info and description of data
laptop_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1624 entries, 0 to 1623
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Laptop_Brand      1624 non-null   object
 1   Laptop_Name       1624 non-null   object
 2   Processor_Company 1624 non-null   object
 3   Processor         1624 non-null   object
 4   Operating_System  1624 non-null   object
 5   RAM               1624 non-null   int64
 6   Storage           1624 non-null   int64
 7   Storage_Type      1624 non-null   object
 8   Screen_Size       1624 non-null   float64
 9   Rating            1624 non-null   float64
 10  Number_of_Reviews 1624 non-null   float64
 11  Price             1624 non-null   float64
dtypes: float64(4), int64(2), object(6)
memory usage: 152.4+ KB
```

Figure 5: Extracted data columns.

The raw, unstructured data was processed and transformed into a structured format using regular expressions (regex). Initially, the data contained various elements like product details, specifications, and customer reviews in a chaotic form. By applying regex patterns, specific data attributes such as brand, model, price, screen size, and other key features were extracted from the raw HTML and structured into columns. This structured data was then cleaned by removing unnecessary or duplicate information, ensuring accuracy and consistency. The result is a well-organized dataset, ready for further analysis and manipulation.

This process significantly enhanced data quality, making it easier to work with, as the unstructured raw data was cleaned and organized into distinct, usable columns for future tasks like analysis or building models.

2.2 Best Buy Data Collection

```python
# 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 6: Sample Best buy Data Scraping code

The above provided script defines a function scrapeallpages() that is responsible for scraping laptop data from the BestBuy website, extracting relevant information, and storing the collected data in a CSV file. The function begins by defining the baseurl for BestBuy and constructing the searchurl that specifically targets the laptops section of the website. It initializes an empty list data to store the scraped laptop information and sets the currentpageurl to the searchurl to start the scraping process from the first page.

The core of the function is a while loop that continues scraping as long as there is a valid page URL to fetch. It begins by calling getpagecontent(currentpageurl), which retrieves the HTML content of the current page and returns a BeautifulSoup object (soup). The scrapelaptopdatafrompage(soup, data) function is then called to extract specific laptop details from the HTML content and append it to the data list. This function presumably collects information such as model number, price, rating, and other key specifications of the laptops.

After scraping the data from the current page, the script checks for the presence of a "next" page using the getnextpage(soup) function. If a next page is found, it updates the currentpageurl to point to the next page and the loop continues. If no next page is found, the loop terminates by setting currentpageurl to None. This approach ensures that the script can scrape all available pages in the series, moving from one to the next until all pages are processed.

Once all pages are scraped, the collected data is structured into a pandas DataFrame with columns for totalinfo, Model No, Rating, and Price. The DataFrame is then saved to a CSV file called laptopsdata.csv. Finally, the function returns the DataFrame, and the script prints out information about the DataFrame's structure using df.info(), giving an overview of the data, including the number of records and the types of columns. This method effectively automates the process of scraping and storing laptop data from the BestBuy website for further analysis.

```
#Storage
storage_pattern = r'(\d+\s*(TB|GB|G)\s*(SSD|HDD|Solid State Drive|Flash Storage|Hard Drive|eMMC|UFS|SDD|PCIe|NVMe|Storage)|'
df1['storage'] = df1['total_info'].str.extract(storage_pattern, expand=False)[0]
#Processor
processor_pattern = r'(Intel\s+\w+\s*\w*|\bM[12]\s+(?:Pro|Max|chip)\s*Built\s*for\s*Apple\s*-\w+|M[12]\s+(?:Pro|Max|chip)\b|
extracted_processors = df1['total_info'].str.extract(processor_pattern)
df1['processor'] = extracted_processors[0].fillna(value=pd.NA)
#Display size
display_pattern=r'(\d{2}"|\d{2}.\d"|\d{2}-inch|\d{2}-Inch|\d{2}.\d-inch|\d{2}.\d-Inch)'
extracted_displays = df1['total_info'].str.extract(display_pattern)
df1['display'] = extracted_displays[0].fillna(value=pd.NA)
#Laptop Name
model_pattern = r'(Geek Squad Certified Refurbished MacBook Air|Envy|XPS|OmniBook|ProBook|Flex|LOQ|Katana|Blade|Aero|Vector|
df1['model'] = df1['total_info'].str.extract(model_pattern)
#Graphics
graphics_pattern = r'(NVIDIA GeForce RTX \d+|Ryzen \d+U|Intel Iris Xe Graphics|NVIDIA Quadro P1000|Intel UHD Graphics 620|NV
df1['graphics'] = df1['total_info'].str.extract(graphics_pattern)
#Storage Type
storagetype_pattern = r'(FlashStorage|HardDrive|SSD|HDD|SDD|eMMC|Flash Storage|Hard Drive|NVMe|PCIe|UFS|Solid State Drive)'
df1['storage type'] = df1['storage'].str.extract(storagetype_pattern)
```

Figure 7: Sample code Coverting unstructured data to raw data using regex.

```
In [4]:
df_BB['graphics'] = df_BB['graphics'].fillna('No Graphics')

df_BB['no_reviews'] = df_BB['no_reviews'].fillna('0')

df_BB['Rating_5'] = df_BB['Rating_5'].fillna('0 Reviews')

df_BB['storage_type'] = df_BB['storage_type'].fillna('SSD')

df_BB['processor'] = df_BB['processor'].fillna('No Info')
```

Figure 8: Sample Data cleaning.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1285 entries, 0 to 1284
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Price              1285 non-null   object
 1   Brand              1285 non-null   object
 2   Colour             1285 non-null   object
 3   ram                1285 non-null   object
 4   storage            1285 non-null   object
 5   display            1285 non-null   object
 6   model              1285 non-null   object
 7   graphics           1285 non-null   object
 8   storage_type       1285 non-null   object
 9   no_reviews         1285 non-null   object
 10  Rating_5           1285 non-null   object
 11  processor_company  1285 non-null   object
 12  os                 1285 non-null   object
 13  processor_model    1285 non-null   object
dtypes: object(14)
memory usage: 140.7+ KB
```

Figure 9: Extracted data Columns

The raw, unstructured data scraped from BestBuy was processed and transformed into a structured format using regular expressions (regex). Initially, the data included elements such as product descriptions, specifications, pricing, and customer ratings presented in an unorganized form. By leveraging regex patterns, key attributes like brand, model, price, rating, and specifications were extracted from the raw HTML content and organized into well-defined columns.

This structured dataset was further refined through a data cleaning process, where unnecessary or duplicate information was removed to ensure consistency and accuracy. The resulting dataset is now clean, organized, and ready for in-depth analysis, enabling tasks such as identifying trends, building models, or generating customer insights.

2.3 Amazon Data Collection

In [ ]:
```python
# 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 [ ]:
```python
# 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 10: Amazon Scraping Code

This Amazon scraping script is designed to collect URLs of laptop products from 70 pages of search results. It uses the scrapepage function to send requests to Amazon, retrieve HTML content, and parse it using BeautifulSoup. The function identifies product links by searching for specific ¡a¿ tags with relevant class attributes, ensuring only valid links are captured. These links are processed to form complete URLs, filtered to remove duplicates using the existingurls set, and returned as a list. If the HTTP request fails, the script prints an error message, ensuring visibility into any issues.

In the main loop, URLs are generated dynamically for each page, and the scrapepage function retrieves the product links. The script checks if fewer than 20 links are found, which could indicate issues with the page structure, and alerts the user to investigate further. Newly scraped URLs are added to a set to ensure uniqueness and appended to a file, laptopurls.txt, for persistence. To mimic human behavior and avoid detection, random delays between 1 and 5 seconds are introduced between requests. The script outputs the total number of unique URLs collected and saves them for subsequent detailed scraping, providing a reliable foundation for further data extraction.

```python
# Extract product details
details = {}
detail_bullets = soup.find('table', {'class': 'a-normal a-spacing-micro'})
if detail_bullets:
    for tr in detail_bullets.find_all('tr'):
        key = tr.find('td', {'class': 'a-span3'}).text.strip().replace(':', '').replace('&lrm;','')
        value = tr.find('td', {'class': 'a-span9'}).text.strip().replace('\u200e', '').replace(':', '')
        details[key] = value

# Extract technical details
tech_details = {}
tech_table = soup.find('table', {'id': 'productDetails_techSpec_section_1'})
if tech_table:
    for row in tech_table.find_all('tr'):
        key = row.find('th').text.strip().replace('&lrm;','')
        value = row.find('td').text.strip().replace('\u200e', '').replace(':', '')
        tech_details[key] = value

# Extract other technical details
other_tech_details = {}
other_tech_table = soup.find('table', {'id': 'productDetails_techSpec_section_2'})
if other_tech_table:
    for row in other_tech_table.find_all('tr'):
        key = row.find('th').text.strip().replace('&lrm;','')
        value = row.find('td').text.strip().replace('\u200e', '').replace(':', '')
        other_tech_details[key] = value
```

Figure 11: Sample Coverting unstructured data to raw data.

In [16]:
```python
processorBrandColumns = ['Other Technical Details_Processor Brand','New Product Details_Processor Brand']
df = df.copy()
df['Processor_Brand'] = df[processorBrandColumns].bfill(axis=1).iloc[:,0]
```

In [17]:
```python
# For Processor Model
processorModelColumns = ['New Product Details_CPU Model Number','Product Details_CPU Model','Technical Details_Processor','
df = df.copy()
df['Processor_Model'] = df[processorModelColumns].bfill(axis=1).iloc[:,0]
df.head(2)
```

Figure 12: Sample Data cleaning.

Out[52]:

	count	unique	top	freq
Laptop_Brand	1930	71	HP	430
Laptop_Name	1930	767	Latitude	152
Processor_Company	1930	7	Intel	1535
Operating_System	1930	45	Windows 11 Pro	748
Processor	1930	157	Core i5	190
Storage_Type	1930	3	SSD	1723
Source	1930	1	Amazon	1930

Figure 13: Extracted data Columns

The raw, unstructured data scraped from Amazon was processed and converted into a structured format using regular expressions (regex). Initially, the data contained various elements, including product descriptions, specifications, pricing, and customer ratings, presented in an unorganized manner. By applying regex patterns, key attributes such as brand, model, price, rating, and specifications were extracted from the raw HTML content and systematically organized into clearly defined columns.

This structured dataset was then refined through a thorough data cleaning process, removing unnecessary or duplicate information to ensure its accuracy and consistency.The final dataset has been successfully loaded into the SQL database. The cleaned and organized dataset is now ready for in-depth analysis, supporting tasks like trend identification, model building, and generating valuable customer 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 14: 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 15: Sample Data

3 EDA (Exploratory Data Analysis)

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

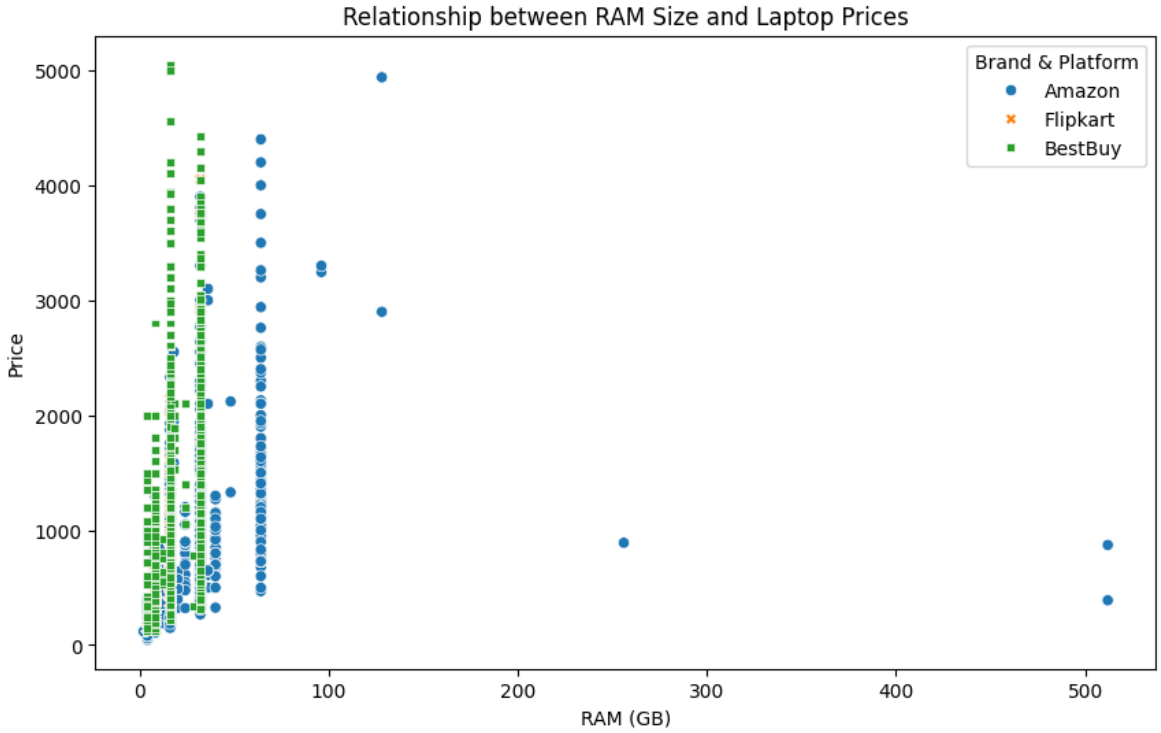

Figure 16: Relationship between RAM size and laptop prices

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 17: Sample Code

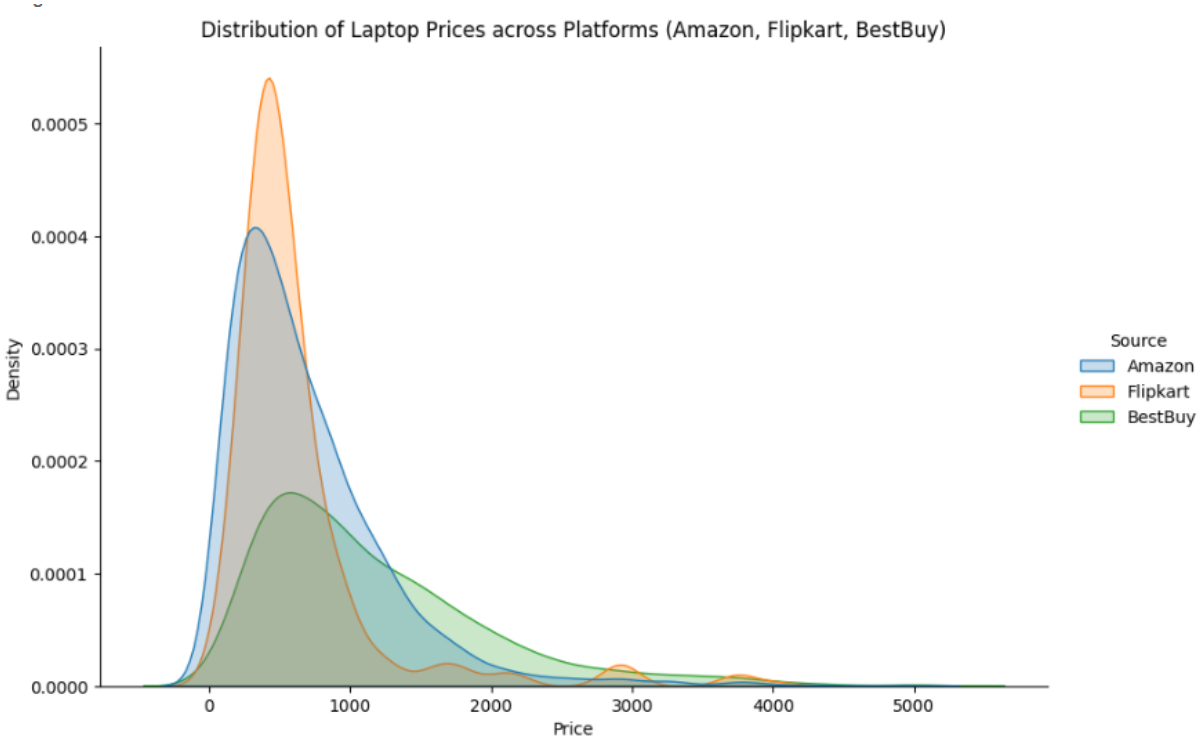

Figure 18: Across Brands

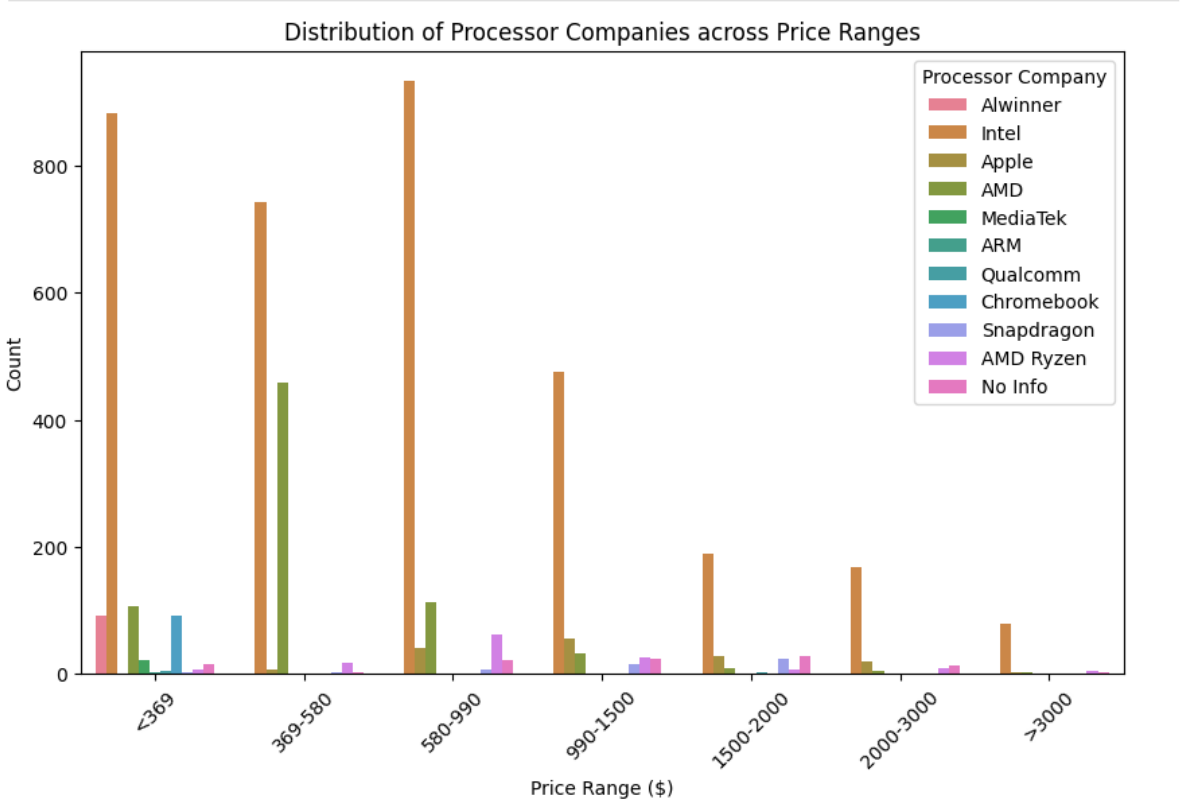

Figure 19: Distribution of processor companies across price ranges.

The plot reveals that Intel dominates the market across various price ranges, followed by Apple. Despite Apple being known for its high-end, expensive products, the number of Intel laptops in the same price range far exceeds that of Apple. Other processor companies are present in different price segments, but they lack the volume to compete with the market leaders, Intel and Apple. An interesting trend is the entry of ARM-based processor companies, such as MediaTek, Snapdragon, and Qualcomm, which are focusing on low-end laptops. Additionally, Apple is noticeably absent in the lower price bracket of around 369 USD.

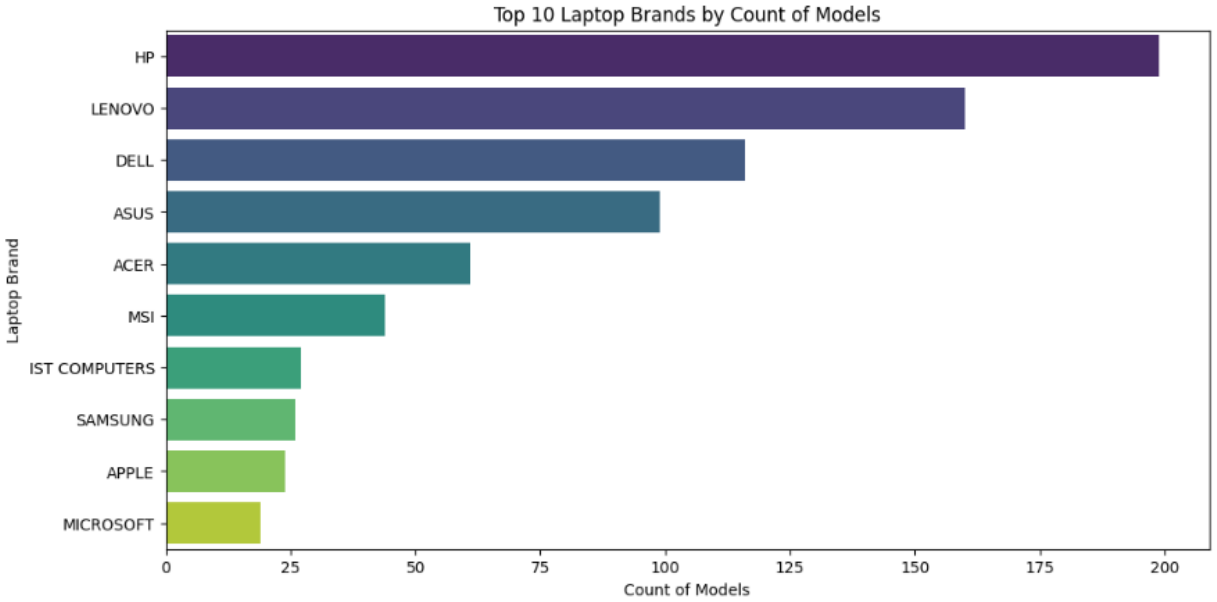

Figure 20: Top 10 brand with more no of models

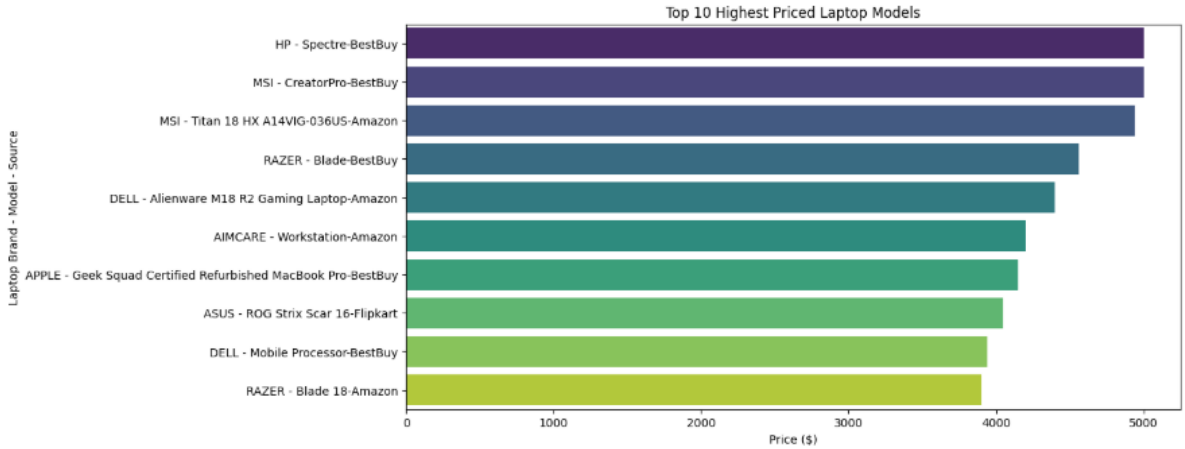

Figure 21: Top 10 Highest priced laptop brands

Hp spectre Bestbuy, MSI creator pro BestBuy , MSI 18 Titan Hx Amazon are the top three costliest models.

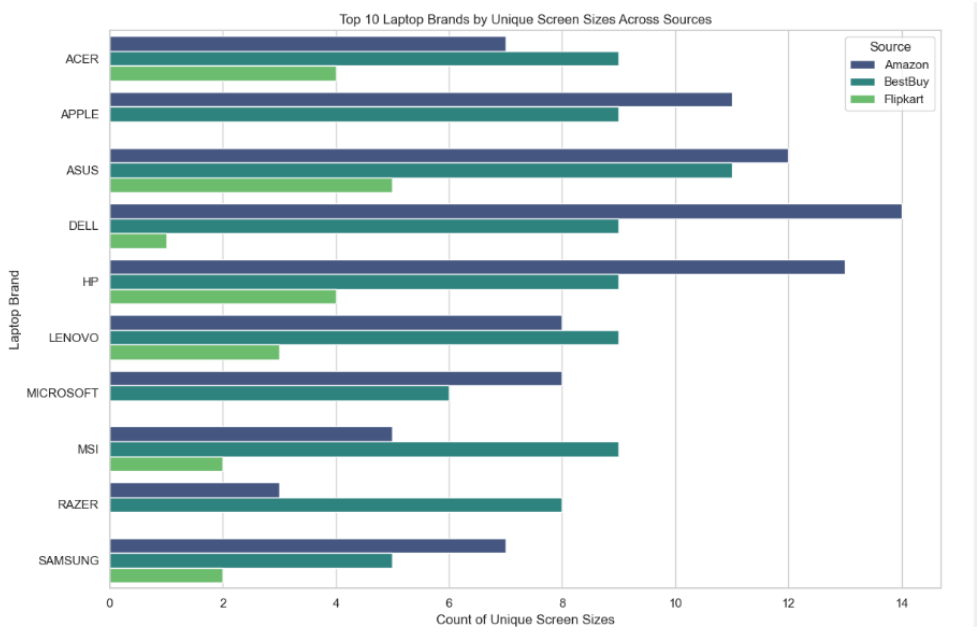

Figure 22: Top 10 Brands by unique screen sizes across brands

On Amazon, Dell, HP, and Asus have the highest number of screen sizes available. In BestBuy, Asus, Acer, and Apple lead in the variety of screen sizes offered. Meanwhile, on Flipkart, Asus, Acer, and HP top the list for the greatest number of screen size options.

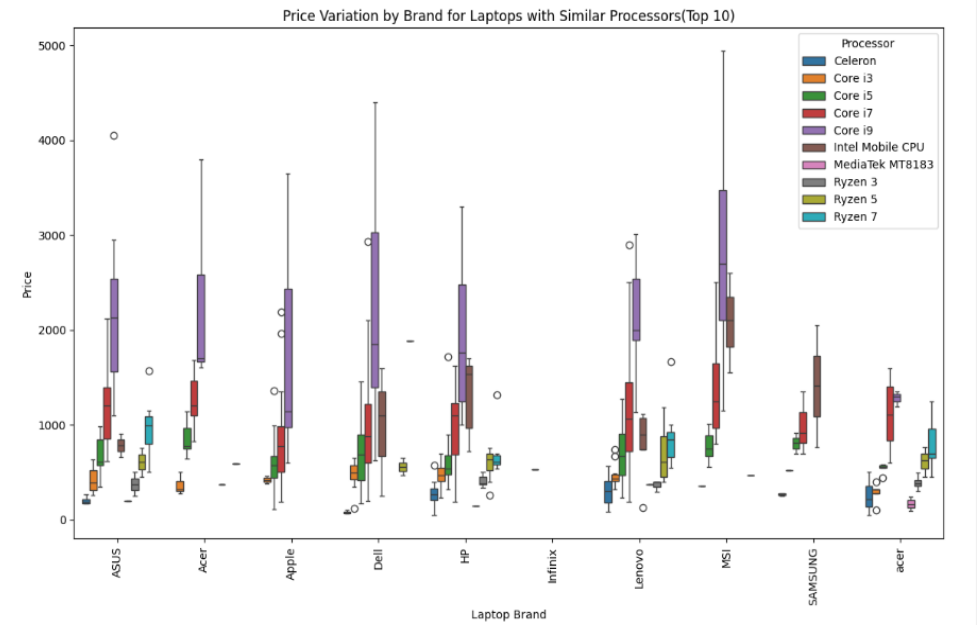

Figure 23: Price variation by brand for laptops with similar processors

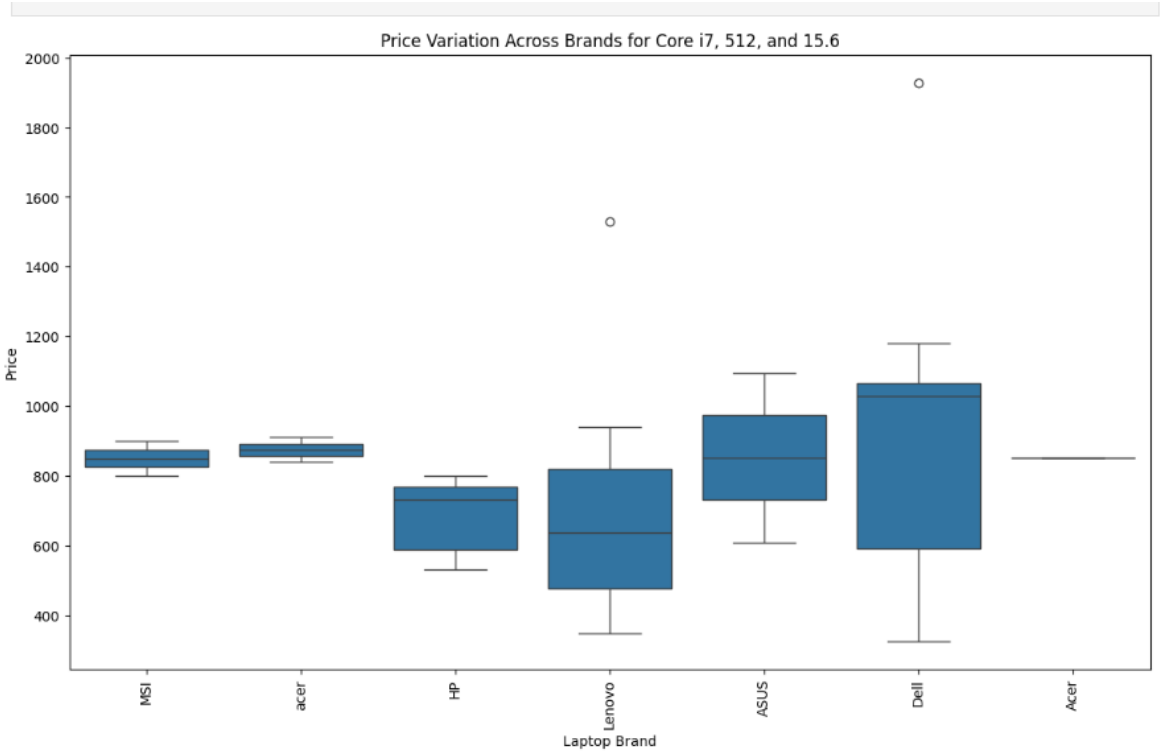

Figure 24: Price variation by brand for core i7,512,and 15.6

Various exploratory data analysis (EDA) operations and analyses were conducted to derive meaningful insights from the data, which are crucial for progressing with the project. Initially, we examined the distribution of key features such as RAM, price, screen size, and processor types to understand their impact on the laptops' pricing and popularity. Correlation analysis helped identify relationships between different features, such as the influence of RAM size or processor brand on the price of the laptops. We also assessed the variance in prices across different brands, processors, and screen sizes, revealing market trends and highlighting outliers that could affect predictive modeling.

Data visualizations, including scatter plots and histograms, were used to illustrate these relationships, helping to uncover patterns in the data that might not be immediately obvious. By identifying these trends, we were able to pinpoint the most significant factors influencing laptop prices and features that appeal to consumers. This analysis serves as a foundation for building models that will predict laptop prices, assess feature importance, and generate further insights for decision-making in the next stages of the project

4 Model Building

In this section, we describe some of the machine learning models used, their selection, and the approach for training and testing.

1. Importing Modules

In [1]:

```
import numpy as np
import pandas as pd
import sqlite3
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, RobustScaler
from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
warnings.filterwarnings("ignore")
```

Figure 25: Importing modules

- **numpy**: A library used for numerical operations and handling arrays and matrices.
- **pandas**: A data manipulation library for working with structured data, specifically DataFrames.
- **sqlite3**: A library that allows interaction with SQLite databases for querying and manipulating data.
- **matplotlib.pyplot**: A plotting library for creating static visualizations such as line, scatter, and bar plots.
- **seaborn**: A statistical data visualization library built on top of Matplotlib, providing an easier interface for attractive plots.
- **warnings**: A module used to manage warning messages, including suppressing them with `filterwarnings`.
- **sklearn**: A machine learning library offering tools for data preprocessing, model training, evaluation, and pipelines. It includes modules for preprocessing, imputation, scaling, and model selection.

2. Loading The Dataset

In [2]:
```
cursor = sqlite3.connect(r'../database/laptrack.db')

laptop_df = pd.read_sql_query("SELECT * FROM Laptop_Phase_2_2", cursor)

cursor.close()

laptop_df.head()
```

Figure 26: Loading the data from sql

3. Preparing the dataset for Model Building

In [3]:
```
# Define columns for each type
categorical_cols = ['Brand', 'Processor_Brand', 'Operating_System', 'Storage_Type', 'Processor_Model']
numerical_cols = ['Extracted_Rating', 'Storage_Capacity(GB)', 'Display_Size(Inches)', 'RAM(GB)', 'No_Of_Reviews', 'Laptop_We

decidingColumns = categorical_cols + numerical_cols
decidingColumns.append('Stock')
```

In [4]:
```
numerical_imputer = SimpleImputer(strategy='mean')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', numerical_imputer),
            ('scaler', RobustScaler())
        ]), numerical_cols),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ]
)
```

Figure 27: Dataset for model building.

The code sets up a preprocessing pipeline to handle both categorical and numerical columns for machine learning tasks. It first defines two lists: categoricalcols for attributes like Brand, ProcessorBrand, OperatingSystem, and StorageType, and numericalcols for columns such as ExtractedRating, StorageCapacity(GB), RAM(GB), and Price. It then combines these lists into decidingColumns, which also includes the Stock column. For numerical data, the pipeline uses a SimpleImputer to fill missing values with the column's mean and applies a RobustScaler to scale the data while being robust to outliers. For categorical data, the OneHotEncoder is used to convert categorical variables into binary columns, with the handleunknown='ignore' parameter ensuring that unseen categories are ignored. Finally, the ColumnTransformer applies the appropriate transformations to the numerical and categorical columns, preparing the dataset for further analysis and model training. This setup ensures that the data is clean, properly scaled, and encoded for

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 28: Xg Boost Regression sample code

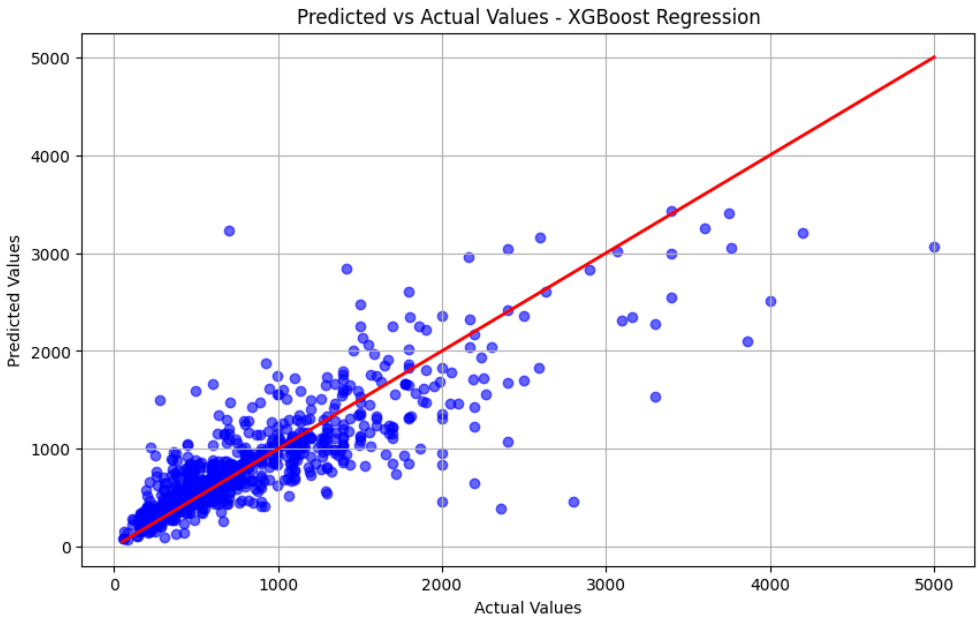

Figure 29: Model Performance

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

4.3 GBDT Regression

In [14]:

```
# GBDT Regression
from sklearn.ensemble import GradientBoostingRegressor

GBDTcategorical_cols = ['Processor_Brand','Operating_System',]
GBDTnumerical_cols = [ 'No_Of_Reviews','RAM(GB)','Laptop_Weight(Pounds)','Price']

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

# Define features (X) and target (y)
X = GBDT_df_cleaned[GBDTcategorical_cols + GBDTnumerical_cols[:-1]]  # Exclude Price from features
y = GBDT_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())
        ]), GBDTnumerical_cols[:-1]),
        ('cat', OneHotEncoder(handle_unknown='ignore'), GBDTcategorical_cols)
    ]
)
```

Figure 30: GBDT Regression sample code

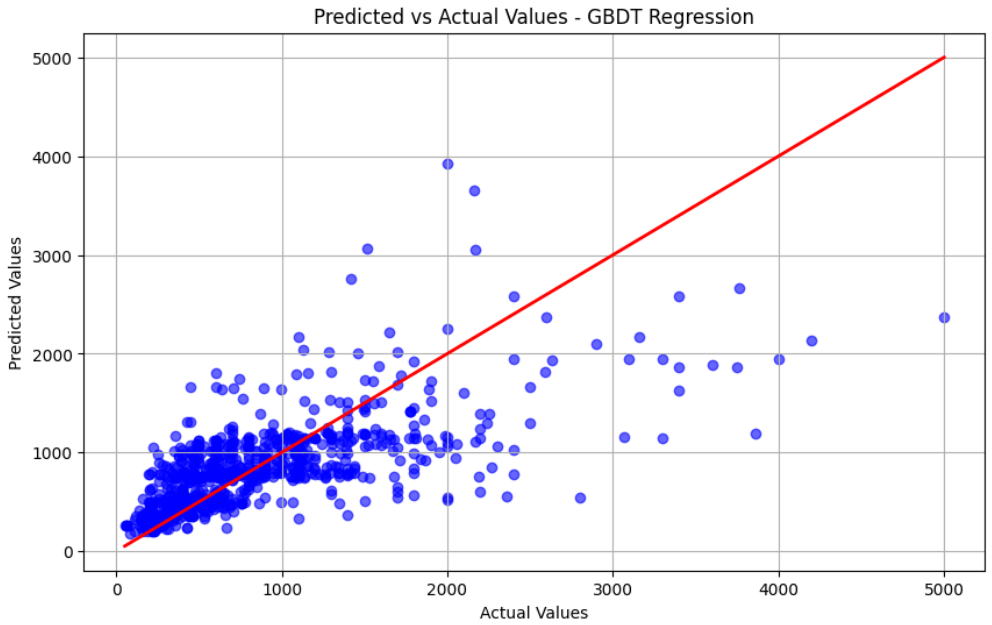

Figure 31: Model Performance

The R^2 score of 0.46 indicates that the model explains 46% of the variance in laptop prices

4.5 Decision Tree Regression

In [22]:

```
# Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor

# Define categorical and numerical columns
DTcategorical_cols = ['Processor_Model','Operating_System']
DTnumerical_cols = [ 'Storage_Capacity(GB)','RAM(GB)','Laptop_Weight(Pounds)','No_Of_Reviews', 'Price']

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

# Define features (X) and target (y)
X = DT_df_cleaned[DTcategorical_cols + DTnumerical_cols[:-1]]  # Exclude Price from features
y = DT_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())
        ]), DTnumerical_cols[:-1]),
        ('cat', OneHotEncoder(handle_unknown='ignore'), DTcategorical_cols)
    ]
)
```

Figure 32: Decision Tree Regression sample code

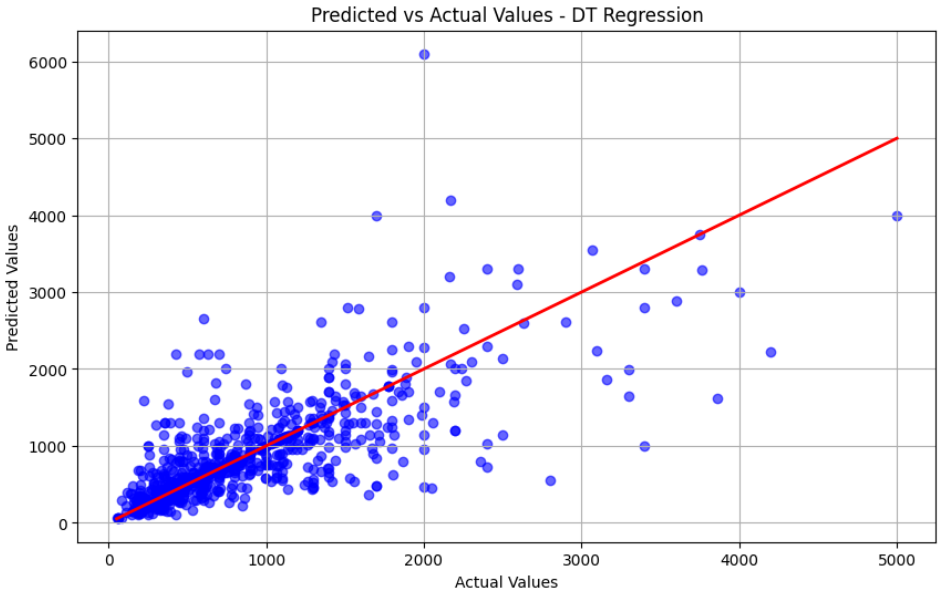

Figure 33: Model Performance

The Decision Tree model showed an R^2 score of 0.355856, indicating that it explained a moderate portion of the variance in laptop prices.

4.7 RANSAC Regression

In [30]:

```
# RANSAC Regression
from sklearn.linear_model import RANSACRegressor, LinearRegression

# Define categorical and numerical columns
RANSACcategorical_cols = ['Brand','Processor_Brand',  'Storage_Type','Stock']
RANSACnumerical_cols = ['Display_Size(Inches)','RAM(GB)', 'Laptop_Weight(Pounds)','No_Of_Reviews','Price']

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

# Define features (X) and target (y)
X = RANSAC_df_cleaned[RANSACcategorical_cols + RANSACnumerical_cols[:-1]]  # Exclude Price from features
y = RANSAC_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())
        ]), RANSACnumerical_cols[:-1]),
        ('cat', OneHotEncoder(handle_unknown='ignore'), RANSACcategorical_cols)
    ]
)
```

Figure 34: RANSAC Regression sample code

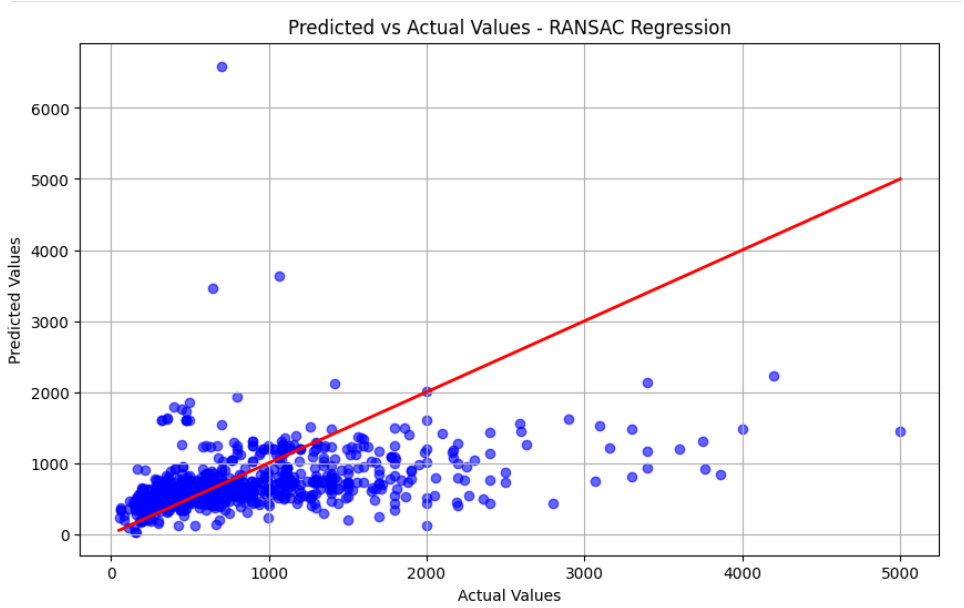

Figure 35: Model Performance

RANSAC did not perform well in terms of R^2 score, yielding a low value of 0.05, indicating that it did not handle the outliers as effectively as anticipated.

4.9 Linear Regression

In [38]:

```
from sklearn.linear_model import LinearRegression


categorical_cols = ['Brand', 'Processor_Brand', 'Operating_System', 'Storage_Type', 'Processor_Model']
numerical_cols = ['Extracted_Rating', 'Storage_Capacity(GB)', 'Display_Size(Inches)',
                  'RAM(GB)', 'No_Of_Reviews', 'Laptop_Weight(Pounds)', 'Price']

laptop_df = laptop_df.dropna(subset=['Price'])

X = laptop_df[categorical_cols + numerical_cols[:-1]]  # Exclude Price from features
y = laptop_df['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),
            ('scaler', RobustScaler())
        ]), numerical_cols[:-1]),  # Exclude 'Price'
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
    ]
)

LR_regressor = LinearRegression()

LR_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LR_regressor)
])
```

Figure 36: Linear Regression sample code

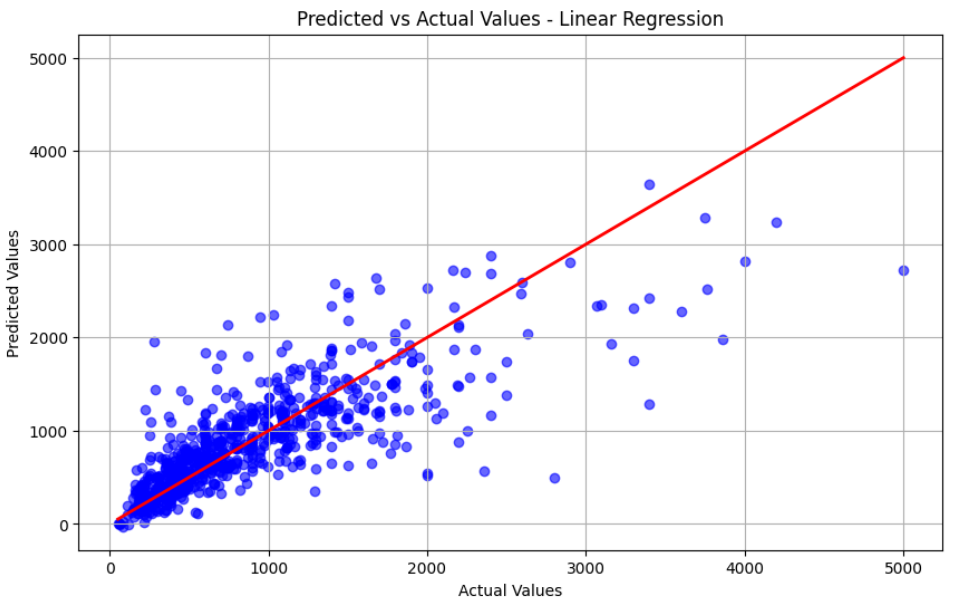

Figure 37: Model Performance

Linear Regression provided a strong R^2 score, indicating that it was able to explain a significant portion of the variance in laptop prices based on the features.

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 38: Lasso Regression sample code

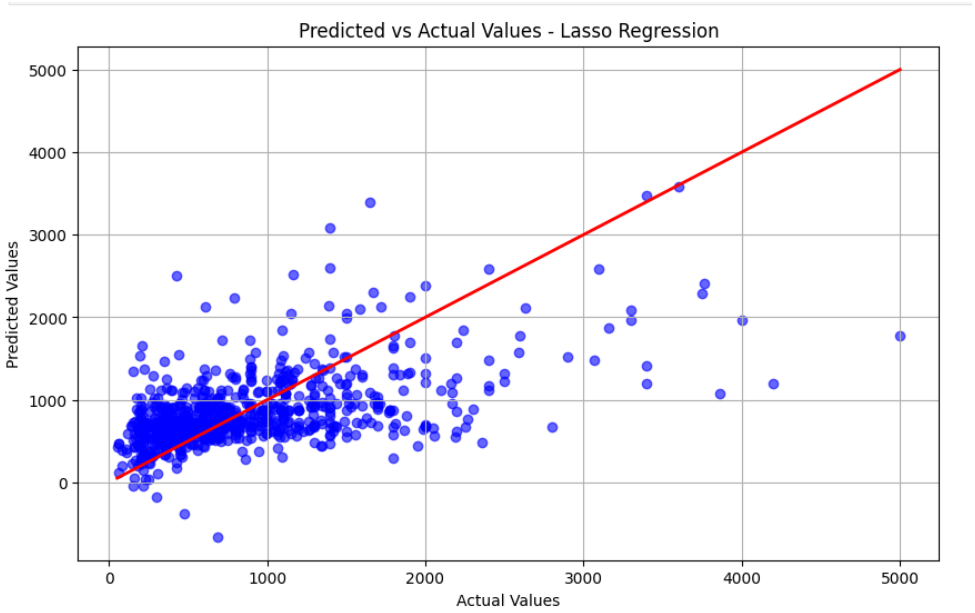

Figure 39: Model Performance

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

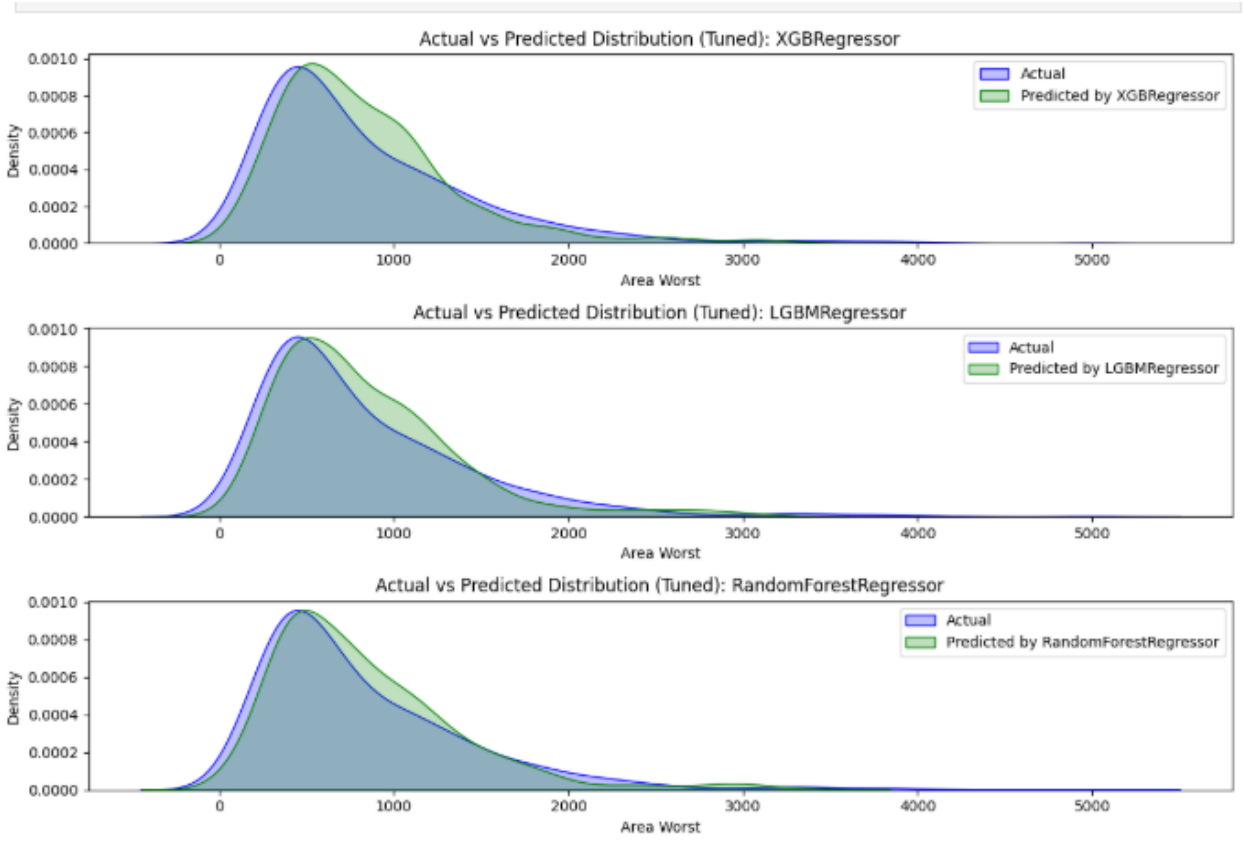

Figure 40: Comparision of model performances among all the models trained.

Final Model Performance Analysis

After performing extensive hyperparameter tuning, the three ensemble models—XGBoost Regressor, LightGBM Regressor, and Random Forest Regressor—demonstrated strong performance in predicting laptop prices. Among these, XGBoost emerged as the top performer, followed by LightGBM, and then Random Forest. Below is a detailed performance comparison of the models:

- **XGBoost Regressor (Best Performing)**
  - MSE: 109,798.53
  - R^2 Score: 0.738
  - Adjusted R^2 Score: 0.735
- **LightGBM Regressor (Second Best)**
  - MSE: 113,916.90
  - R^2 Score: 0.728
  - Adjusted R^2 Score: 0.725

- **Random Forest Regressor (Third Best)**
  - MSE: 120,103.55
  - R^2 Score: 0.714
  - Adjusted R^2 Score: 0.710

Conclusion and Model Performance Insights

XGBoost Regressor was the best-performing model, with an R^2 score of 0.738, which means it explained around 74% of the variance in laptop prices. The minimal difference between its R^2 and adjusted R^2 scores (just 0.003) indicates excellent generalization without overfitting. This suggests that XGBoost provides a highly reliable prediction with minimal bias or variance issues.

LightGBM Regressor, while slightly behind XGBoost, performed very well with an R^2 score of 0.728, indicating that it can explain about 73% of the variance in laptop prices. The minor difference between its R^2 and adjusted R^2 scores suggests that the model also generalized well, making it a strong contender for this task.

The Random Forest Regressor showed a solid performance with an R^2 score of 0.714, but it ranked third among the models. Although it demonstrated effective predictive capability, the larger gap between the R^2 and adjusted R^2 scores (0.004) points to a slightly higher tendency to overfit compared to XGBoost and LightGBM.

Key Insights

- **High Predictive Power:** All three models achieved R^2 scores above 0.70, indicating strong predictive accuracy for laptop price prediction.
- **Minimal MSE Difference:** The MSE differences between the models were relatively small (approximately 10,305), highlighting consistent performance across all three methods.
- **Ensemble Method Effectiveness:** The close performance of all three models confirms the effectiveness of ensemble methods in predicting complex tasks like laptop price forecasting.
- **Generalization Ability:** The small differences between R^2 and adjusted R^2 scores across all models suggest good generalization, reducing the risk of overfitting.

In summary, while all three models performed well, XGBoost emerged as the most reliable and accurate for laptop price prediction, making it the best choice for deployment.

5 Recommendation System

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 41: 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 the Display Resolution is split into width and height for numerical analysis. A correlation matrix is computed for relevant features like RAM(GB), Price, and Battery Life. The heatmap visually highlights feature relationships, while correlations with

Price are printed to identify significant predictors. This step helps in selecting important features and understanding their influence on laptop prices.

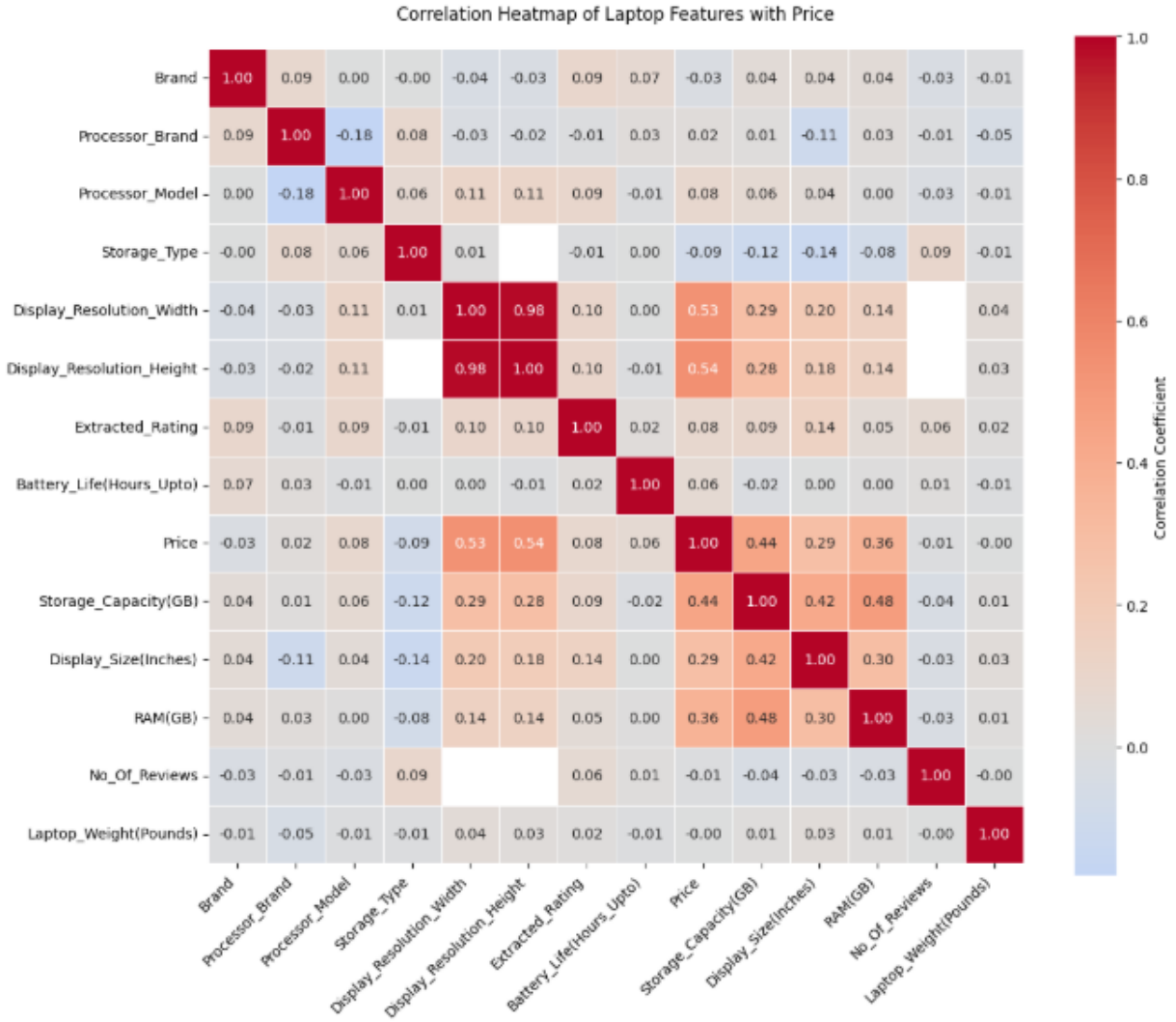

Figure 42: Correlation Heatmap

.

The `heatmap` is a highly effective and insightful visualization tool that displays the correlation between numerical and encoded categorical features, emphasizing how these features interact with the target variable, `Price`. By visualizing correlations in a matrix format, it provides a clear and immediate understanding of how each feature is related to `Price`, whether positively or negatively. Strong positive correlations, such as `Display_Resolution_Height`, `RAM`, and `Storage_Capacity`, are represented in warmer colors, indicating a direct, increasing relationship with the price. On the other hand, negative correlations, like those observed with `Storage_Type`, are shown in cooler colors, indicating an inverse relationship. This color-coding makes it easy to identify which features are most influential in determining the price.

The heatmap not only aids in recognizing these individual relationships but also high-

lights any potential interdependencies or collinearity among the variables. Features with the highest correlation to `Price` become evident, helping to prioritize them for feature selection and model optimization. By simplifying the often complex relationships in a dataset, the heatmap serves as a powerful tool in data exploration. It enables data scientists and analysts to focus on key predictors, refine model inputs, and improve the overall performance of regression models, making it an essential step in the process of building predictive models.

Feature	**Correlation Coefficient**
Display_Resolution_Height	0.539
Display_Resolution_Width	0.533
Storage_Capacity(GB)	0.444
RAM(GB)	0.361
Display_Size(Inches)	0.288
Processor_Model	0.082
Extracted_Rating	0.078
Battery_Life(Hours_Upto)	0.060
Processor_Brand	0.024
Laptop_Weight(Pounds)	-0.001
No_Of_Reviews	-0.008
Brand	-0.027
Storage_Type	-0.088

Table 1: Top Features Correlated with Price

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 43: Sample model training code

.

The `train_and_evaluate_models` function trains and evaluates multiple regression models on the provided training and test data. It iterates over a predefined list of models (e.g., KNN, Decision Tree, Linear Regression, etc.), fits each model to the training data, and makes predictions on the test data. The function then evaluates each model using predefined metrics (calculated by `evaluate_model`), stores the results, and prints them for each model. Finally, the results are returned as a DataFrame for easy comparison. This approach helps to assess and compare the performance of different regression algorithms.

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
Lasso	3.362504e+05	579.871053	0.198081	0.185460
Ridge	3.363633e+05	579.968403	0.197812	0.185187
Linear Regression	3.363712e+05	579.975145	0.197793	0.185168
Huber	3.763781e+05	613.496643	0.102381	0.088254
RANSAC	4.338320e+05	658.659211	-0.034640	-0.050923
SVR	4.571034e+05	676.094257	-0.090139	-0.107297
Theil-Sen	1.123982e+07	3352.584079	-25.805684	-26.227565

Table 2: Model Evaluation Results

5.1.3 Hyper parameter tuning on Top3 models

In [7]:

```python
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 44: Sample Paramter tuning code

.

The hyperparameter tuning is applied to the top three performing models based on their initial evaluation. These models are:

- **XGBoost**: With the best performance in terms of R2 score and MSE among the evaluated models.
- **Random Forest**: Another high-performing model with competitive results.
- **GBDT (Gradient Boosting Decision Trees)**: A robust model known for its ability to capture complex relationships in the data.

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 3: Tuned Model Evaluation Results

The table compares the performance of XGBoost, Random Forest, and GBDT models using MSE, RMSE, R^2, and Adjusted R^2. XGBoost outperforms the other models with the lowest MSE and RMSE, and the highest R^2 and Adjusted R^2, indicating better prediction accuracy and model fit.

5.1.4 Saving the models

In [24]:

```python
def save_models_and_preprocessing(best_models, scaler, folder_path='saved_models/'):
    """Save the trained models and preprocessing objects"""
    import os
    if not os.path.exists(folder_path):
        os.makedirs(folder_path)

    # Save the best models
    for name, model in best_models.items():
        with open(f'{folder_path}{name}_model.pkl', 'wb') as file:
            pickle.dump(model, file)

    # Save the scaler that was used in training
    with open(f'{folder_path}scaler.pkl', 'wb') as file:
        pickle.dump(scaler, file)


scaler = RobustScaler()
numerical_columns = X_train.select_dtypes(include=['float64', 'int64']).columns
scaler.fit(X_train[numerical_columns])

# Save everything
save_models_and_preprocessing(best_models, scaler)
```

Figure 45: Saving models and the preprocess

.

The function `save_models_and_preprocessing` is used to save the trained machine learning models and the preprocessing objects (such as the scaler) to disk. This is essential for future use, such as model deployment or further evaluation, without needing to retrain the models.

Steps in the function:

- First, the function checks whether the specified directory `folder_path` (default is `'saved_models/'`) exists. If not, it creates the directory.
- It then iterates over the `best_models` dictionary, which contains the models that have been tuned and selected as the best performers. For each model, it saves the model to a `.pkl` file in the specified directory using the `pickle` module. The file name is derived from the model name, with the suffix `_model.pkl`.
- Lastly, it saves the scaler object (in this case, a `RobustScaler`) to a file named `scaler.pkl`. The scaler was used to scale the numerical features during the training process, and saving it ensures the same scaling is applied to any new data that is processed later.

Scaler and Models: In the code provided, a `RobustScaler` is instantiated and fitted to the numerical columns in `X_train`. The `RobustScaler` is a preprocessing technique that scales features using the median and the interquartile range, making it less sensitive to outliers compared to other scalers like `StandardScaler`. Once the scaler is fitted, both the trained models and the scaler are saved using `pickle` to ensure the entire model pipeline can be loaded and used in the future.

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]:

```python
# 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 46: Sample Building Recommendation code

.

The `LaptopRecommender` class is designed to provide laptop recommendations based on similarity to other laptops in the dataset or user-specified features. It is initialized with a dataframe containing laptop data, and includes methods for data preprocessing, such as extracting numerical values from `Display_Resolution`, converting categorical features to numerical codes, and scaling features using `MinMaxScaler`. The `get_recommendations` method calculates cosine similarity between laptops and returns the top N most similar ones. Additionally, `get_recommendations_by_features` allows users to input specific preferences (like `RAM`, `Storage_Capacity`) and receive recommendations based on those criteria. The class helps to make personalized recommendations by identifying laptops with the highest similarity to the selected features or laptops.

In [14]:

```
# Get recommendations based on specific features
features = {
    'Brand': 'Dell',
    'Price': 1000,
    'RAM(GB)': 16,
    'Storage_Capacity(GB)': 512
}
feature_based_recommendations = recommender.get_recommendations_by_features(features)
format_recommendations(feature_based_recommendations)
```

Out[14]:

	Brand	Laptop_Model_Name	Processor_Model	RAM(GB)	Storage_Capacity(GB)	Price	Extracted_Rating	Similarity_Score
0	50	ThinkPad P15	259	128	16384	$6,099.99	3.1	61.70%
1	38	Envy 16t-h1000	126	4	16384	$3,429.99	NaN	50.64%
2	52	Titan 18 HX A14VIG-036US	214	128	8192	$5,496.00	4.5	49.10%
3	11	MacBook Pro	250	128	4096	$5,259.00	NaN	46.99%
4	50	Lenovo ThinkPad P16 Gen 2	214	128	8192	$5,000.00	5.0	46.67%

Figure 47: Sample Building Recommendation code

.

This recommendation system is designed to provide laptop suggestions based on content-based filtering, utilizing multiple relevant features of laptops. The system processes both categorical and numerical data, ensuring that a wide range of features such as brand, processor, RAM, storage capacity, display resolution, and price are considered for recommendations. It uses a similarity matrix, calculated through the cosine similarity metric, to evaluate how similar laptops are based on the selected features. This allows the system to find laptops that are similar to a given one or to recommend laptops based on user-defined criteria. The system's flexibility ensures that it can cater to various user needs, whether they are looking for laptops similar to one they are currently considering or specifying certain features such as a preferred brand, price range, or storage capacity.

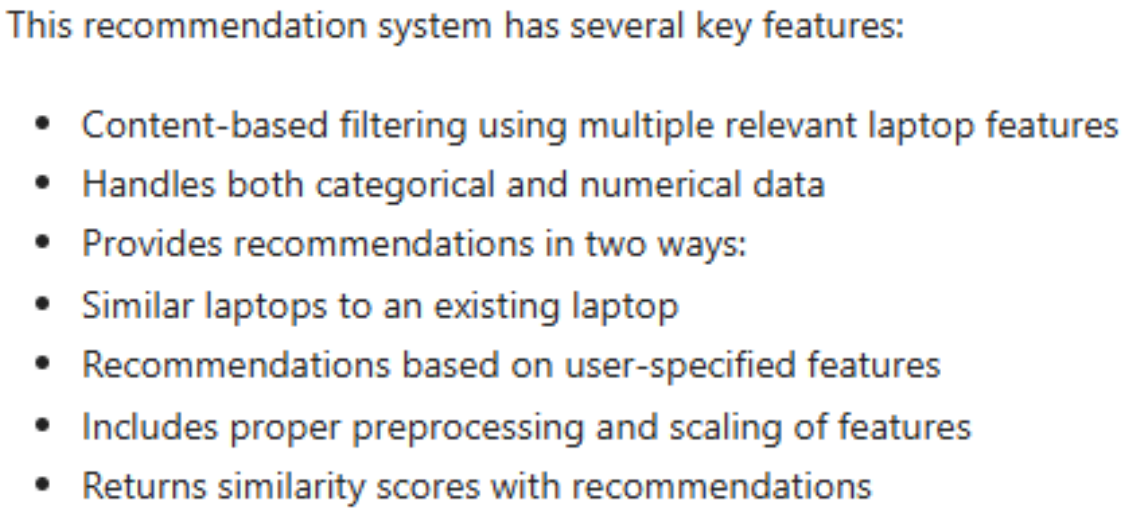

Figure 48: Key Features

.

Additionally, the system preprocesses the data by encoding categorical variables, handling missing values, and scaling the features to ensure that all attributes are on the same scale. This improves the accuracy of the similarity calculation and leads to more relevant recommendations. The system also provides the ability to return similarity scores along with the recommendations, giving users insight into how closely the recommended laptops align with their preferences. The preprocessing and scaling steps, which include converting categorical data into numerical codes and applying Min-Max scaling to numerical features, are essential for maintaining the effectiveness of the recommendation process. With these capabilities, the system is able to efficiently recommend laptops tailored to the specific needs and preferences of the user, making it 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.

7 Conclusion

This application focuses on analyzing laptop data scraped from three prominent e-commerce websites: Amazon, Flipkart, and BestBuy. The process began by employing web scraping techniques to extract a wide range of laptop-related data, including pricing, specifications, customer ratings, and reviews. This data was then carefully analyzed to identify the key features that influence laptop pricing the most, such as processor type, RAM size, storage capacity, screen size, and brand reputation. By understanding the correlation between these features and their impact on pricing, the application aims to provide both consumers and retailers with valuable insights that can guide purchasing and product strategy decisions.

The next step in the analysis involved the use of various machine learning regression models to predict laptop prices based on different combinations of these features. These models, including decision trees, random forests, and linear regression, were tested and compared to determine the most accurate and reliable model for price prediction. After evaluating the models' performance, the best-performing model was selected to build a robust predictive model that can estimate laptop prices accurately based on real-time data.

The primary goal of this analysis was to better understand how various laptop features affect pricing and how these features differ across different retail platforms. By examining how each platform (Amazon, Flipkart, and BestBuy) prices laptops based on their specifications, this project helps identify the factors that contribute most to the price variation and helps explain why certain laptops are priced higher than others. The insights from this analysis not only benefit consumers by helping them make more informed decisions based on their budget and desired specifications but also assist retailers in optimizing their pricing strategies and tailoring their product offerings to meet customer preferences and market trends.

Furthermore, a web application was developed to display the entire dataset to an admin user, allowing them to modify or update data as necessary. The application also features a user interface where customers can view predicted laptop prices in real-time, all powered by the selected machine learning model. The app connects to a PySpark backend, ensuring efficient data processing and fast predictions, without relying on external platforms like Jupyter Notebooks.

In conclusion, this application serves a dual purpose: it aids consumers in selecting the best laptop within their budget by providing detailed insights into features and pricing, and it offers sellers valuable data to adjust their pricing strategies and improve product offerings to meet customer expectations.

8 References

References

[1] Chada Lakshma Reddy, K Bhargav Reddy, G. R Anil, *Laptop Price Prediction Using Real Time Data*. Available at: https://ieeexplore.ieee.org/document/10085473.

[2] Mohammed ALi,Medicherla Varshith,Sanka SriVyshnavi *Laptop Price Prediction using Machine Learning Algorithms*. Available at: https://www.researchgate.net/publication/369955491_Laptop_Price_Prediction_using_Machine_Learning_Algorithms.

[3] Shashank Tiwari, Akshay Bharadwaj, Sudha Gupta *Stock price prediction using data analytics*. Available at: https://ieeexplore.ieee.org/document/8318783.

[4] Ajinkya Yelne, Dipti Theng *Stock Prediction and analysis Using Supervised Machine Learning Algorithms*. Available at: https://ieeexplore.ieee.org/document/9697162.

[5] Jiahao Yang *Big Data Analyzing Techniques in Mathematical House Price Prediction Model*. Available at: https://ieeexplore.ieee.org/document/9744970.

[6] Amazon. *Data Collection Website*. Available at: https://www.amazon.com/s?k=laptops.

[7] Flipkart. *Data Collection Website*. Available at: https://www.flipkart.com/search?q=laptops.

[8] BestBuy. *Data Collection Website*. Available at: https://www.bestbuy.com/site/searchpage.jsp?st=laptops.

[9] Python. *The Python Programming Language*. Available at: https://www.python.org.

[10] PySpark. *Apache Spark - PySpark Documentation*. Available at: https://spark.apache.org/docs/latest/api/python/.

[11] SQL. *SQL Documentation and Tutorials*. Available at: https://www.w3schools.com/sql/.

[12] BeautifulSoup. *BeautifulSoup Documentation*. Available at: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[13] Docker. *Docker Documentation and Overview*. Available at: https://www.docker.com.