Background of the Study:

The study of web scraping has its origins in the growing importance of data in the modern world. Data has evolved into a critical asset for businesses and organizations, and the ability to collect, analyze, and interpret data has evolved into a competitive advantage. However, web scraping solves this problem by automating data collection and making it easier to access, analyze, and automate. Web scraping has evolved into a sophisticated and powerful tool for data extraction over time. There are numerous web scraping tools and techniques available today, ranging from simple scripts to complex software platforms. As the demand for web scraping has increased, so has the requirement for data privacy and security. Many websites now use anti-scraping measures to prevent unauthorized access to their data.

Research Problem:

One of the primary challenges in web scraping is dealing with the vast amount of unstructured data available on the internet is one of the most difficult aspects of web scraping. This data is frequently dispersed across multiple websites and in various formats, making it difficult to collect and analyze. The manual extraction process, on the other hand, is time-consuming, inefficient, and prone to errors. To address these issues, effective and dependable web scraping techniques and tools that can automate data extraction are required. Furthermore, legal and ethical considerations regarding web scraping must be considered. Addressing these issues will result in the development of efficient and accurate web scraping techniques that follow legal and ethical guidelines. This will enable businesses to extract valuable data from websites in a timely and ethical manner.

Research Question:

- What are the most effective methods for web scraping and processing substantial amounts of unstructured data from the internet?
- How can the impact of web scraping on the websites being scraped be reduced to avoid performance issues or increased server load?
- What is the role of web scraping in tracking market trends and gathering competitive intelligence to inform business strategies?

- How does Beautiful Soup compare to other web scraping tools, such as Scrapy or Selenium?
- What are some best practices for web scraping with Beautiful Soup, such as avoiding overloading servers or respecting website terms of service?
- What are the best tools and technologies for web scraping, and how do they compare in terms of performance and ease of use?
- How can Beautiful Soup web scraping be optimized for speed and efficiency when dealing with substantial amounts of data?
- How does Beautiful Soup navigate through the HTML structure of a website to extract data efficiently?

Data:

There are several types of data sets that can be used for web scraping using Kaggle website.

- E-Commerce product data – Amazon, eBay, and Walmart.
- Job Posting data – Job title, location, salary, and company information.
- Sports data – Game scores, Player Statistics and Team ranking.

I used Beautiful Soup, a Python web scraping toolkit. I began by determining the URLs of the pages I intended to scrape. Then I created a Python script to extract data from these pages using Beautiful Soup. From each product page, I gathered the product name, price, rating, and user reviews. The product category and subcategory were also taken from the product page URL.

Analysis:

One of the primary advantages of web scraping is its capacity to capture enormous amounts of data in a short period of time. This data can be utilized for a variety of purposes, including market research and pricing monitoring. Web scraping can also be used to automate data collection procedures, saving businesses and organizations time and resources.

Beautiful Soup is a Python web scraping library. It parses HTML and XML documents and provides a simple method for extracting data from them. Beautiful Soup is a powerful library that web scraping professionals rely on. It is easy to use and has a lot of features. The pip package installer for Python can be used to install the library. To begin using Beautiful Soup, we

must first import it into our Python script. After we import the library, we can use it to parse HTML and XML documents. The library includes methods for parsing HTML and XML documents, such as parse (), prettify (), and find_all ().

- Target Product Selection:

  Determine the specific Amazon product for which you want to track the price drop. Obtain the product page's URL.

- Installing Python Libraries:

  Install the necessary Python libraries, such as Beautiful Soup, smtplib, and requests. Import the necessary modules into your Python script.

- Scraping the Amazon Product Page:

  Send an HTTP request to the product page URL and retrieve its HTML content using the requests library. Make a Beautiful Soup object and parse the HTML.

- Extracting Price Information:

  Examine the HTML structure of the product page to locate the price element. Beautiful Soup can be used to extract price information from HTML code.

- Defining the Target Price:

  Determine the price at which you want to be notified. Set the desired price as a variable in your Python script.

- Comparing Prices:

  Compare the extracted price to the target price. If the extracted price is less than the target price, proceed to the next step. Otherwise, terminate the script.

- Emailing an Alert:

  Configure an SMTP server and verify the sender's email credentials. Create an email message that includes the product details as well as a price decrease announcement. Use the smtplib library to send an email alert to the recipient's email address.

Findings:

Pricing data: Web scraping can be used to collect pricing data from e-commerce websites, which can be used to monitor price trends, identify rivals, and inform pricing plans.

The project scrapes Amazon's website for a specific product and sends an email alert if the price falls below a certain level. The code specifies the product's URL, the desired price threshold, as well as the email sender and receiver. It then scrapes the product page for the current price, determines whether it has dropped below the threshold, and sends an email notification.

Beautiful Soup:

Beautiful Soup includes a variety of strong capabilities, such as search and filter methods, tree traversal methods, and ways for editing and cleaning up HTML content. It is widely used in data science, machine learning, and web development, and it has a big developer community behind it. Beautiful Soup has some limitations, such as restricted web scraping capabilities and performance difficulties. Version compatibility and dependency management might also be difficult.

Drawbacks of Beautiful Soup:

Beautiful Soup is a popular Python library used for web scraping and parsing HTML and XML documents. However, like any tool, it has some drawbacks that you should be aware of before using it:

- Limited web crawling support: Because Beautiful Soup is primarily intended for parsing HTML and XML documents, it lacks built-in capability for crawling multiple pages or websites. It can be used in conjunction with other libraries, such as Requests, to crawl webpages, but it may necessitate additional configuration and setup.
- Limited support for JavaScript rendering: Because Beautiful Soup lacks built-in functionality for rendering JavaScript, it may be unable to scrape data from some JavaScript-powered websites. Parsing errors: Sometimes, HTML and XML documents can have errors that can cause issues with parsing using Beautiful Soup.
- Requires understanding of HTML structure: In order to be utilized successfully, Beautiful Soup requires some understanding of the structure of HTML and XML documents. This can be a challenge for those who are new to web programming or have no prior knowledge.

- Performance: Due to its pure Python implementation, Beautiful Soup may not be the most performant option for large-scale web scraping tasks. Other libraries, like XML, have quicker parsing speeds in comparison.

Overall, Beautiful Soup is a useful tool for parsing HTML and XML documents that may be used for a variety of online scraping projects. However, there are some drawbacks to consider, particularly for larger-scale web crawling and scraping applications that demand more complex functionality such as JavaScript rendering and higher parsing speeds.

Discussion:

I used Amazon's website to web scrape my data and notify the product price drop via email using Beautiful Soup in Python. I identified several reputable journals in the field of Web Scraping, including Web Scarping tools and Techniques in Python with After identifying the journals, I conducted a search for peer-reviewed articles related to the topic of interest. I filtered out at least sixteen relevant articles based on the title and abstract.

Selenium is another famous Python package for web scraping. It can extract data from websites that need user involvement, such as those requiring login credentials, and is used to automate web browsers. Selenium may be used to interact with a website in the same way that a human would. The library can be installed with Pip.

To get started with Selenium, we need to install a web driver for the browser we want to automate. For example, to automate the Chrome browser, we must first download and install the Chrome Driver. We can use Selenium to automate the browser after installing the web driver.

Selenium is a well-known technology for automating web browsers, such as web scraping operations that need interaction with JavaScript-driven websites. However, like with any technology, there are some limitations to be aware of before utilizing it.

Selenium can be slower than alternative web scraping libraries, such as Beautiful Soup and Requests-HTML, because it needs launching and communicating with a web browser instance. This can increase the time necessary for each scrape, especially if there are a lot of sites to scrape.

Excessive resource consumption: Running a web browser instance can be time-consuming, especially for large-scale web scraping operations. As a result, the Selenium machine may execute slower and consume more memory.

Knowledge of Web Browser Automation: To use Selenium efficiently, some knowledge of web browser automation and web page structure is required. For people who are new to web programming or have no prior experience, this can be a hindrance.

Some websites only have limited support: To restrict automated access, several websites use CAPTCHA challenges or IP blocking. Selenium may be unable to bypass these checks and may be denied access to the website.

A recent paper by Bricongne, Jean Charles (2023) used Beautiful Soup for web scraping of COVID-19-related news articles from Chinese news websites. The study's goal was to examine the sentiment of news articles and identify the topics covered in the articles. The authors used Beautiful Soup to extract the title, content, date, and source of the news articles. The extracted data were then analyzed using sentiment analysis and topic modeling techniques.

In another recent paper by Shreesha M, Srikara S B, and Manjesh R. (2021) Intelligent Smart Parser is a web-based application that displays news from various sources on a single platform. It uses web scraping and natural language processing techniques to retrieve and summarize news articles into three to four lines of text. This allows users to quickly understand the main points of a news story. The app also shows news based on the user's location as well as trending news based on views. Overall, Intelligent Smart Parser makes it easier to access news and saves user’s time. It is a useful tool for those who want to stay up to date on current events and developments around the world.

Thivaharan. S (2020) analyzes three prominent Python libraries used for online scraping: Beautiful Soup, LXml, and RegEx. According to statistical research, RegEx is faster on average but has restricted rule extraction capabilities. Beautiful Soup and Xml, which are based on the DOM model, are slower but more suited for extracting online information in complicated contexts. The report also mentions how online scrapers influence modern content grading systems in social media for regional languages. Overall, RegEx works best in simple contexts.

Web scraping with Python using Beautiful Soup and Selenium has become a key tool for data extraction and analysis in a range of industries, including finance, ecommerce, and social media, according to the literature review. Beautiful Soup and Selenium are two popular web scraping libraries, with Beautiful Soup better suited for parsing HTML and XML files and Selenium for automating web browsers. However, the legal and ethical aspects of online scraping should not be neglected, and necessary measures should be taken to ensure compliance with data privacy laws and ethical principles.

Overall, Python's web scraping capabilities have opened new possibilities for data-driven decision-making, and it has tremendous potential for speeding complex data extraction procedures.

Limitations:

- Legal Concerns: Some websites expressly ban web scraping, and scraping such sites may result in legal ramifications. While performing web scraping activities, it is critical to review and abide with the website's terms of service.
- Web scraping can be hampered by technological barriers such as CAPTCHAs, IP filtering, and anti-scraping mechanisms put in place by websites to prevent illegal access.
- Incomplete or Inaccurate Data: If the scraper is not designed to handle all conceivable situations and exceptions, or if the website structure changes, web scraping may provide incomplete or inaccurate data.
- Ethical Concerns: It is crucial to evaluate the ramifications of data gathering and utilization before scraping personal or sensitive information without consent.
- Beautiful Soup may be unable to parse all sorts of HTML and XML text, especially if it is poorly formatted or incorrect.
- Beautiful Soup's online scraping capabilities are limited: While it is useful for parsing and collecting data from HTML material, it may not be appropriate for more sophisticated web scraping jobs that require more advanced techniques, such as dynamic web scraping or working with JavaScript-based websites.
- Dependency Management: Beautiful Soup relies on additional libraries to work properly, such as XML and html5lib, which can complicate the setup process and make maintenance more complex.

Overall, this project successfully deployed web scraping to extract the price of a product from an e-commerce website and send an email notification if the price fell below the target price. The project uses the requests and Beautiful Soup libraries to extract information from the homepage, as well as the smtplib and SSL libraries to deliver email notifications, it provides a foundation for a pricing monitoring and alert system, which can be extended and customized based on specific requirements and preferences.

Code:

```
import requests                                  # Importing the 'requests' library to make HTTP requests

from bs4 import BeautifulSoup as ms              # Importing BeautifulSoup module to parse HTML and XML documents

import smtplib                                   # Importing the 'smtplib' library to send email using Simple Mail Transfer Protocol (SMTP)

import ssl                                       # Importing the 'ssl' library to provide a secure socket layer (SSL) for network communication

from email.message import EmailMessage           # Importing the 'EmailMessage' class from the 'email.message' module


# The URL of the product page we want to scrape

URL = "https://www.amazon.in/Apple-MacBook-Chip-13-inch-256GB/dp/B08N5T6CZ6/ref=sr_1_1_sspa?crid=2ATG7O2B9WDUK&keywords=macbook%2Bair&qid=1683745223&sprefix=%2Caps%2C167&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th=1"

response = requests.get(URL)


# User-Agent header to be used in the HTTP request to mimic a browser

header = {'User-Agent': 'Mozilla/5.0 (Windows Phone 10.0; Android 6.0.1; Microsoft; RM-1152) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Mobile Safari/537.36 Edge/15.15254'}
```

```
# A function to extract the price of the product from the product page

def product_price():

    page = requests.get(URL, headers=header)

    soup = ms(page.content, 'html.parser')                    # Parse the content of the page using BeautifulSoup

    price_str = soup.find('span', {'class': 'a-price-whole'}).text.split()[0].replace(',',"")                    # Find the HTML element that contains the price and extract the text

    price = float(price_str)                        # Removed the duplicate line and assigned price_str to the extracted price string

    return price


def discount():

  response = requests.get(URL)

  content = response.content

  soup = ms(content, 'html.parser')

  discount_percentage = soup.find('span', {'class': 'a-size-large a-color-price savingPriceOverride aok-align-center reinventPriceSavingsPercentageMargin savingsPercentage'}).text

  return discount_percentage


# A function to send an email notification if the price drops below the target price
```

```
def notify(p):
    port = 465  # For SSL
    smtp_server = "smtp.gmail.com"            # The address of the SMTP server
    sender_mail = "pythonprojectsp23@gmail.com"
    receiver_mail = "pythonprojectsp23@gmail.com"
    password = 'ajvmbpnbbdtbrwje'

    if (p!="stillhigh"):          # If the price has dropped below the target price
      subject = 'BUY NOW!'        # Set the email subject
      # Set the email message
      message = f"""
    Price has dropped! Buy your product now!

    Click the link to check it out!
    Link: {URL}

    Original Price: Rs.{my_price:,}

    Discount: {discount()}
```

```
    Final Price: Rs.{product_price():,}


    """
    else:
      subject = 'DO NOT BUY NOW!'                # If the price is still high, set the email subject
accordingly
      message = f"""
      Price of this product is still very high! It is not a good idea to buy right now."


      Click the link to check it out!
      Link: {URL}


      """


    m = EmailMessage()
    m['From'] = sender_mail
    m['To'] = receiver_mail
    m['Subject'] = subject
    m.set_content(message)
    context = ssl.create_default_context()          # create a default SSL context
```

```
    with smtplib.SMTP_SSL(smtp_server, port, context=context) as server:
        server.login(sender_mail, password)
        server.send_message(m)  # Use send_message instead of sendmail
        print("Message Sent")


my_price = 95000.0
while True:
    if product_price() <= my_price:                 # checking the prices
        notify("buy")
        break
    else:
        print("Price of this product is still very high!")
        notify("stillhigh")
        break
```

Summary of the Code:

This Python code retrieves a product's price and discount percentage from Amazon.in and sends an email notification if the price falls below a user-defined target price.

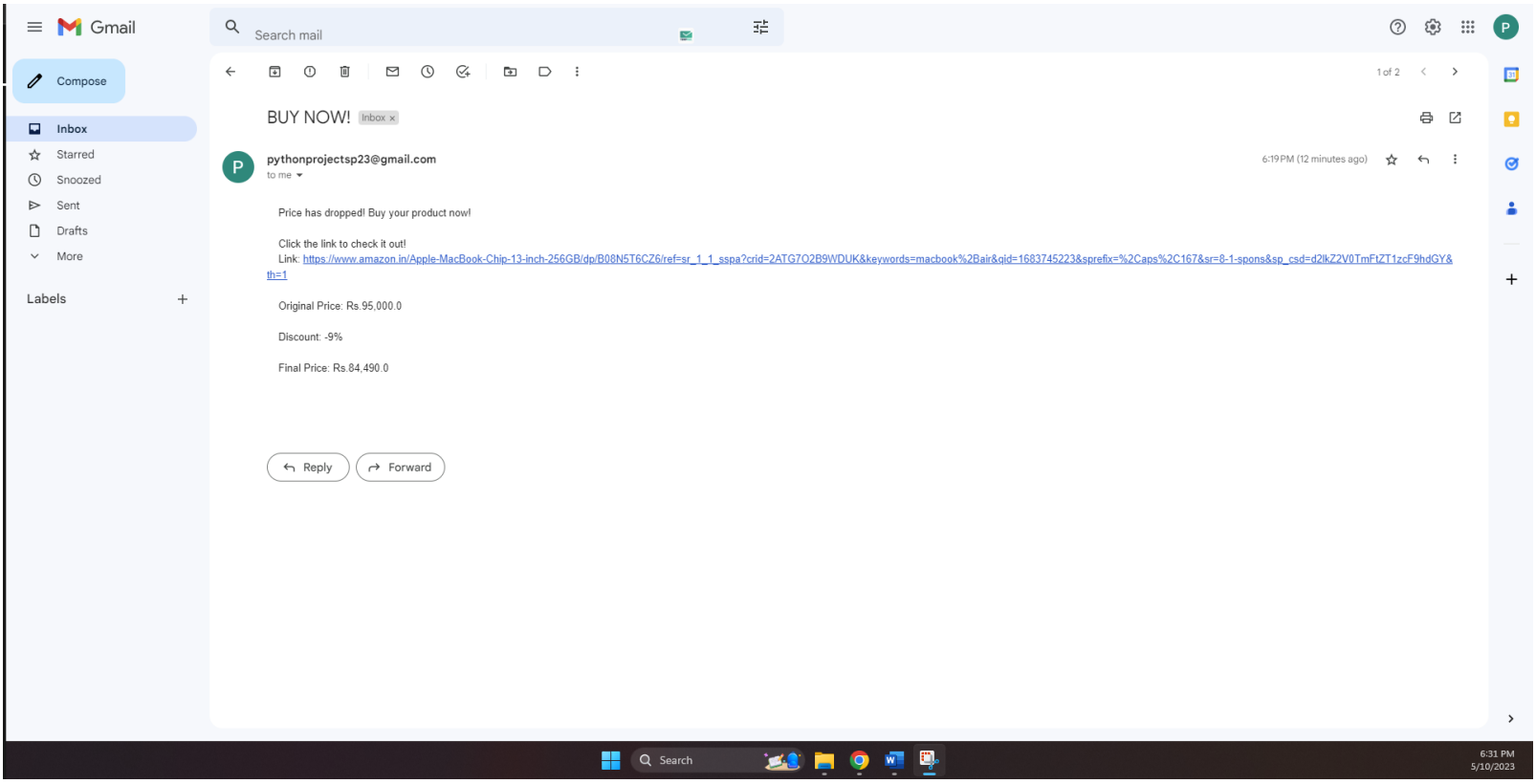
The code makes HTTP calls using the requests library and parses HTML and XML documents with the 'Beautiful Soup' module. It also makes use of the'smtplib library to send email using the Simple Mail Transfer Protocol (SMTP), as well as the SSL library to provide a secure socket layer (SSL) for network connection. The URL of the product page to be scraped is specified, as well as a User-Agent header that simulates a browser. There are two functions defined: 'product_price()' extracts the product's price and 'discount()' extracts the discount %.

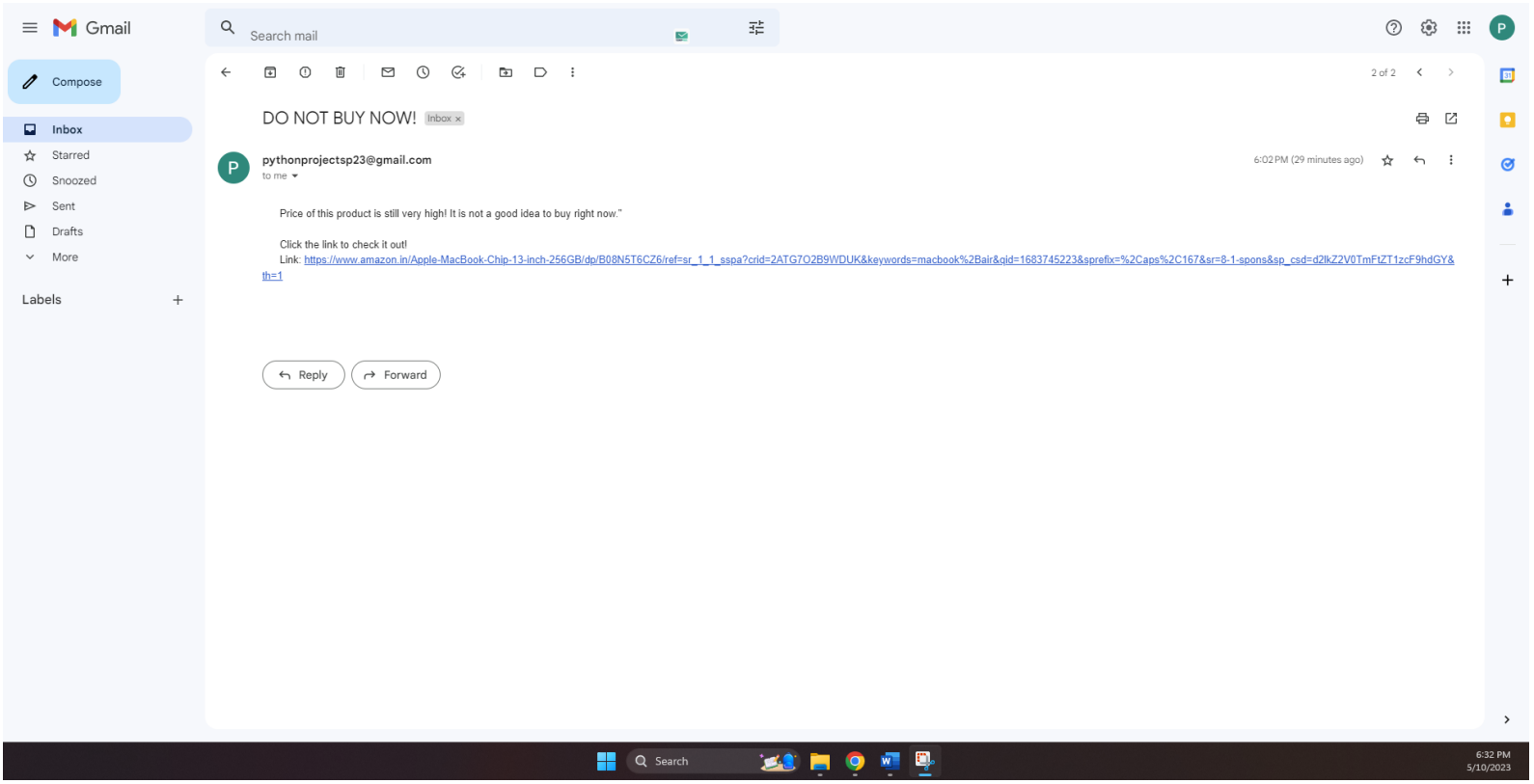
It is important to connect the STMP Server to the email server that we are using and test the STMP Server tool to ensure that it is connected to the server with the required port for the email that we are using. As an alternative to using the same password that I would use for the app to access my mail, I decided to use the app password instead of the password that I would use for the app to access my mail.

If the price falls below the target price, the 'notify()' method sends an email notification. It connects to the Gmail SMTP server and configures the email subject and message accordingly. The URL of the product, the original price, the discount, and the final price are all included in the email message.

The user-defined target price is set at Rs. 95,000, and a while loop checks the product's current price on a regular basis. If the current price is less than or equal to the target price, the 'notify()' method sends an email notification. If the price remains too high, an email is issued stating, “It is not a good idea to buy right now”.

Output:


Gmail
Search mail
Compose
Inbox
Starred
Snoozed
Sent
Drafts
More
Labels
1 of 2
BUY NOW!
Inbox
pythonprojectsp23@gmail.com
to me
6:19 PM (12 minutes ago)
Price has dropped! Buy your product now!
Click the link to check it out!
Link: https://www.amazon.in/Apple-MacBook-Chip-13-inch-256GB/dp/B08N5T6CZ6/ref=sr_1_1_sspa?crid=2ATG7O2B9WDUK&keywords=macbook%2Bair&qid=1683745223&sprefix=%2Caps%2C167&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th=1
Original Price: Rs.95,000.0
Discount: -9%
Final Price: Rs.84,490.0
Reply
Forward
Search
6:31 PM
5/10/2023


Gmail
Search mail
Compose
Inbox
Starred
Snoozed
Sent
Drafts
More
Labels
2 of 2
DO NOT BUY NOW!
Inbox
pythonprojectsp23@gmail.com
to me
6:02 PM (29 minutes ago)
Price of this product is still very high! It is not a good idea to buy right now."
Click the link to check it out!
Link: https://www.amazon.in/Apple-MacBook-Chip-13-inch-256GB/dp/B08N5T6CZ6/ref=sr_1_1_sspa?crid=2ATG7O2B9WDUK&keywords=macbook%2Bair&qid=1683745223&sprefix=%2Caps%2C167&sr=8-1-spons&sp_csd=d2lkZ2V0TmFtZT1zcF9hdGY&th=1
Reply
Forward
Search
6:32 PM
5/10/2023