**WEB SCRAPING**

*Blair Hall, Logan Bannister, Tony Ramirez*

University of Kentucky

# ABSTRACT

This is a program based on web scraping and the SMTP library in Python. It alerts the user automatically with an email when the lowest price of the product changes. The email message is formatted by the product with the lowest price to the highest price. That way the user does not have to manually check the prices, ratings, number of reviews, and availability of the products. This can save time and be very useful when doing online shopping.

# 1. INTRODUCTION

Web scraping is an emerging service to extract information and advertisement by using different information retrieval methods such as copy-pasting, HTML parsing, DOM parsing, vertical aggregation, Xpath, and text pattern matching. Its main purpose is to suggest products and services to the ever-growing population of internet users.

Web scrapers simulate exploration on the internet by implementing low-level hypertext transfer protocol or embedding suitable web browsers. It transforms unstructured data on the web, usually in HTML format, to structured data stored in a database or other form of information.

Currently, web scraping is used for online price comparison, weather data monitoring, website change detection, and web research. Gathering data has become a business for multiple search engines and applications. The usage of web scraping allows startups and other companies to gather information quickly. However, not all web scraping is positive since it could also lead to harmful activities such as service attacks, data theft, spam, and digital ad fraud.

In this paper, we focus on creating a legal and positive web scraper to gather different prices of a product on the Amazon website. Given an item by the user, the web scraper would use BeautifulSoup, a

Python package for parsing HTML and XML documents. The different prices of each item were stored in a dictionary. The key was the item name and the value was the price. The dictionary elements were sorted by price from lowest to highest. The information was then placed in an email format and sent to a user, specified by the programmer. The web scraping ran continuously and an email was resent every time a new price changed.

# 2. BACKGROUND

In this section, we give a brief description of the main topics addressed in this paper. BeautifulSoup and Simple Mail Transfer Protocol (SMTP).

## 2.1 Beautiful Soup

Beautiful soup is used for web-scraping and it’s a great tool to extract large amounts of unstructured data. It’s a Python library used to pull data from HTML and XML. It can extract articles and content from browsers and turn them into a Python list of dictionaries. **2.2 Simple Mail Transfer Protocol (SMTP)**

The SMTP is an internet standard communication protocol for electronic mail transmission. In this project, GMAIL SMTP was used to send and receive emails. The server address is smtp.gmail.com.

# 3. LITERATURE REVIEW OF DIFFERENT APPROACHES

When developing our web scraping project, it was important that we research to find existing solutions to our problem. From this research, we found 2 different resources that used web scraping to find price information from an Amazon webpage. These two resources were from Anish Malla via GitHub and JournalDev. Both of these web scrapers work with Amazon but have some similarities and differences to our project. By comparing our project with these existing resources, we were able to create a better web scraping project.

**3.1. Anish Malla’s Amazon price tracking web**

# scraping

Malla’s Amazon price tracking web scraping uses Beautiful Soup to regularly check the price of an item and when the price does go down they send an email to the user alerting them that the price has decreased [2]. This differs from our project in that price is the only information that Malla’s program is scraping for. Another difference is that Malla’s web scraper only sends an email saying “The price has decrease please check the item. The price decrease by {decrease} rupees.” [2]. This does not provide the user with the initial price information and will only say the amount of currency that the price has decreased by. This may provide some confusion for the user when utilizing this program.

## 3.2. JournalDev

Like our project, JournalDevs’s web scraper uses Beautiful Soup to find the product title, price, rating, number of reviews, and availability of a list of products on an Amazon page [3]. This information is the same information that our project obtains. The key difference between this code and ours is that our code will continue to run, while JournalDev’s only runs once. This means that any change in price information will never be updated for the user. Another flaw with JournalDev’s code is that it occasionally does not provide price information for certain products. This was an error that we were able to find and fix with our code that involved how the HTML of Amazon’s website contained the price. JournalDev also does not implement SMTP to send an email, meaning the information is only shown where the Python code is executed.

# 4. TECHNICAL APPROACHES

We decided to use the Beautiful Soup Python library for our web scraping. It extracts data from HTML and XML files. It uses the source code of the file to create a parse tree to extract the data in a hierarchical and readable fashion. In the case of our code, Beautiful Soup extracts the price, the rating, number of ratings, and availability of the product from the Amazon page. We then store that information in Python lists (equivalent to an array in C, C++, Java, etc.) in a for loop that iterates over the number of links in the list of links. Next, we store the price, rating, number of ratings, and availability of the product in dictionaries where the key is the title of the product in a for loop that iterates over the number of products in the list of product names. We sort the dictionary of {title: price} from lowest to highest price and the key-value pairs in a new dictionary. Next, we create the message that will be sent as an email. We did this by creating a function. The message is originally initialized to an empty string. A for loop that iterates over the dictionary of title: price key-value pairs sorted from lowest to highest. In this loop, we have five strings that make up the message for the information for each product. Each string is interpolated into the next string. Each string has one newline character at the end of the string for readability except for the fifth string. The fifth string has two newline characters to separate the product information of the next product. The function prints the message sent to the console to signal that the message has been created. This was used for debugging purposes. Lastly, we had a while true loop. In the loop, we stored the product with the lowest price. We call a function that calls the functions that get all the product information again. We store the most recently calculated lowest price in another variable. In an if statement, we compare the former lowest price product with the current lowest price product. If the current lowest price is different (higher or lower) than the previous lowest price, we create the message again and send the email again.

The email that received the message was a test email account set up for demonstration purposes. It was a GMAIL account. The message was received as output by our code. The message did not include a subject or any additional information outside of the product information.

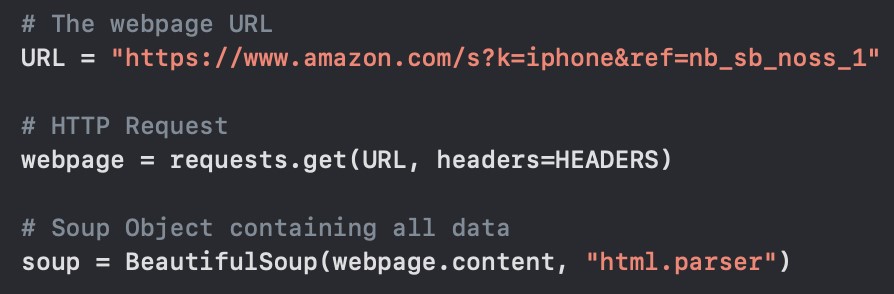
## 4.1 Technical Approach Flowchart/Diagram

Our technical approach is illustrated in the figure below. In this flowchart, you can see each of the steps that the code does in order to form the email containing the product information. The first step is sending a get request for the amazon link. Then, using Beautiful Soup, we parse the HTML website for the desired product information. This information is then stored and sorted based on the product’s price. Using the sorted products, an email message is created and sent. After this step, you can see the loop that is checking for price changes after the initial execution. The code will not stop running unless quit by the user.

**Fig. 1.** Technical approach flowchart for our web scraping project.

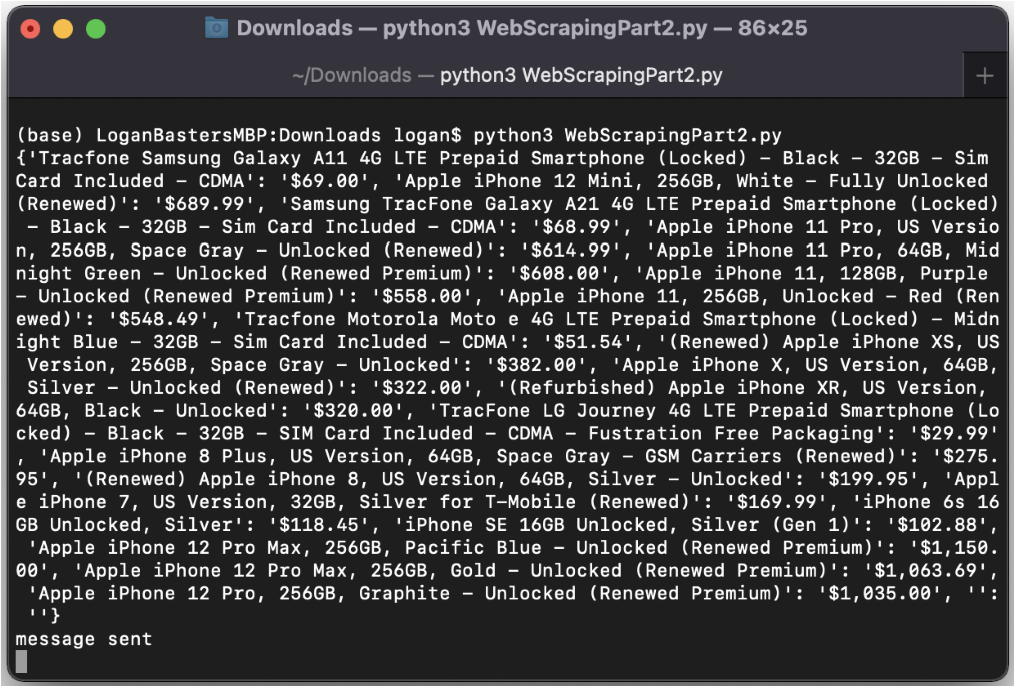
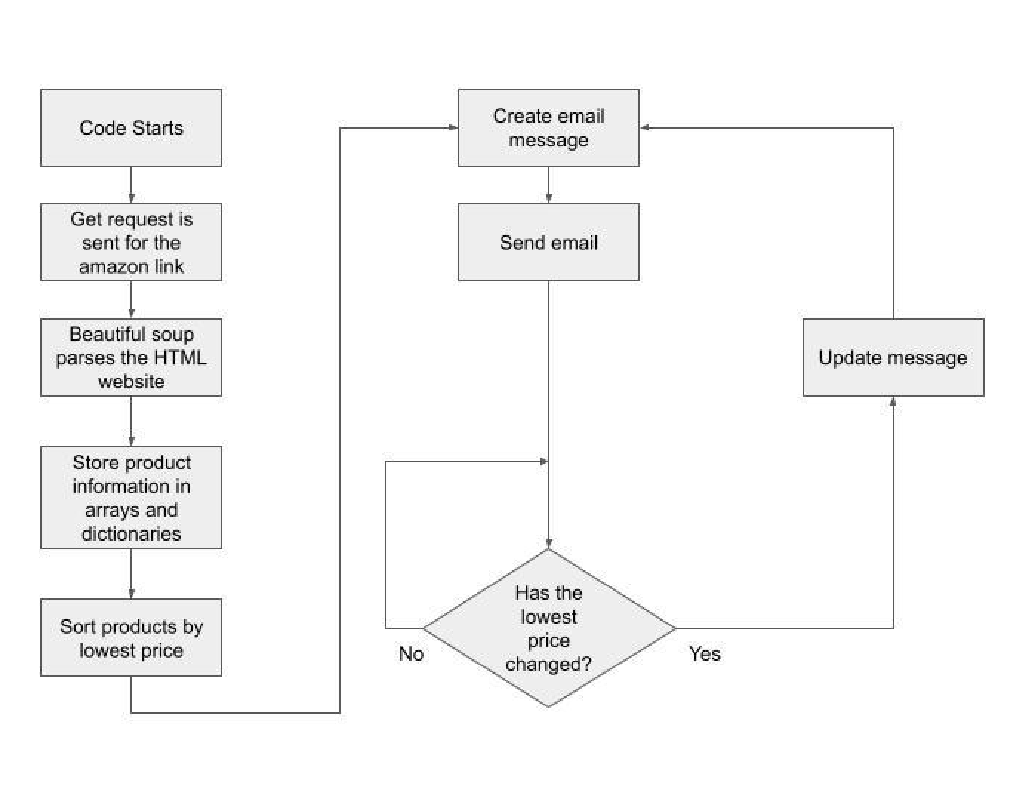
# 5. EXPERIMENTAL RESULTS

The first step in using our web scraping project is to find and insert a URL for an Amazon search page. This allows the user to choose from a variety of product pages which will be where all of the information is pulled from. The figure below shows the webpage URL within our Python code that then gets used by the web scraping code. For all of the shown results, this was the URL that was used.



**Fig. 2.** Example of placing the desired URL in the code.

When the code is executed, it begins scraping the provided webpage URL. Once all of the product information was scraped from the website, it is output as an array to the Python execution. With the collected product information, a message is formatted with all of the information. This is then sent as an email using SMTP. When this is completed, the Python code outputs “message sent”. The described array and output message can be seen in the figure below.



**Fig. 3.** Example of the code’s output after scraping a given URL.

A major aspect of our project is that we use the product information that we gather using web scraping to send an email containing this information. The **Fig. 4.** Example of the email sent by the code.

email

that

was

sent

using

our

code

is

shown

in

the

figure below.

**Fig.**

**5.**

Example

of

Wireshark

measurement

of

the

response time.

The

highlighted

blue

box

in

Fig

5.

Demonstrates

the

request

to

retrieve

the

different

prices

on

iPhone

products.

Surprisingly,

it

took

approximately

37

seconds

to

gather

all

the

information.

Our

hypothesis

is

for

different

products

it

ranges

on

time

based

on

the

available

items

to

purchase

on

the

website.

For

example,

for

the

Playstation

4

,

it

only

took

1.2

seconds, but there were also fewer products on site.

The

web

scraping

algorithm

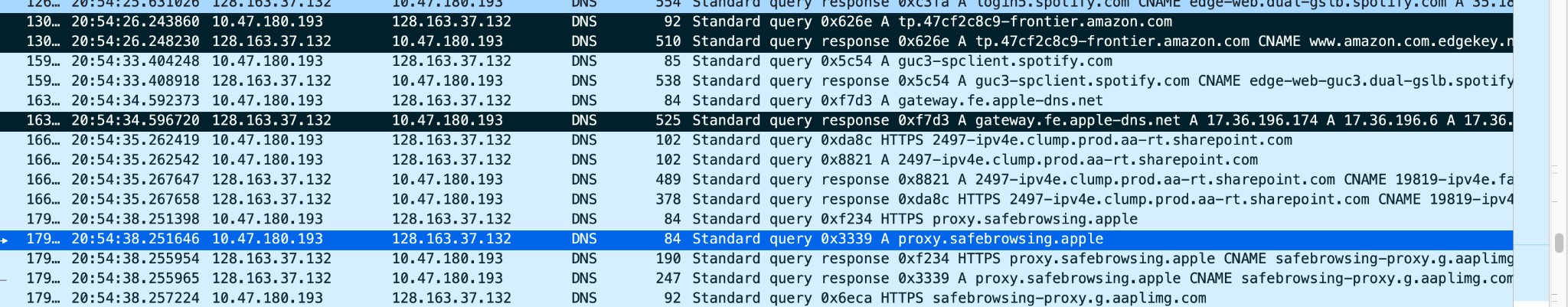
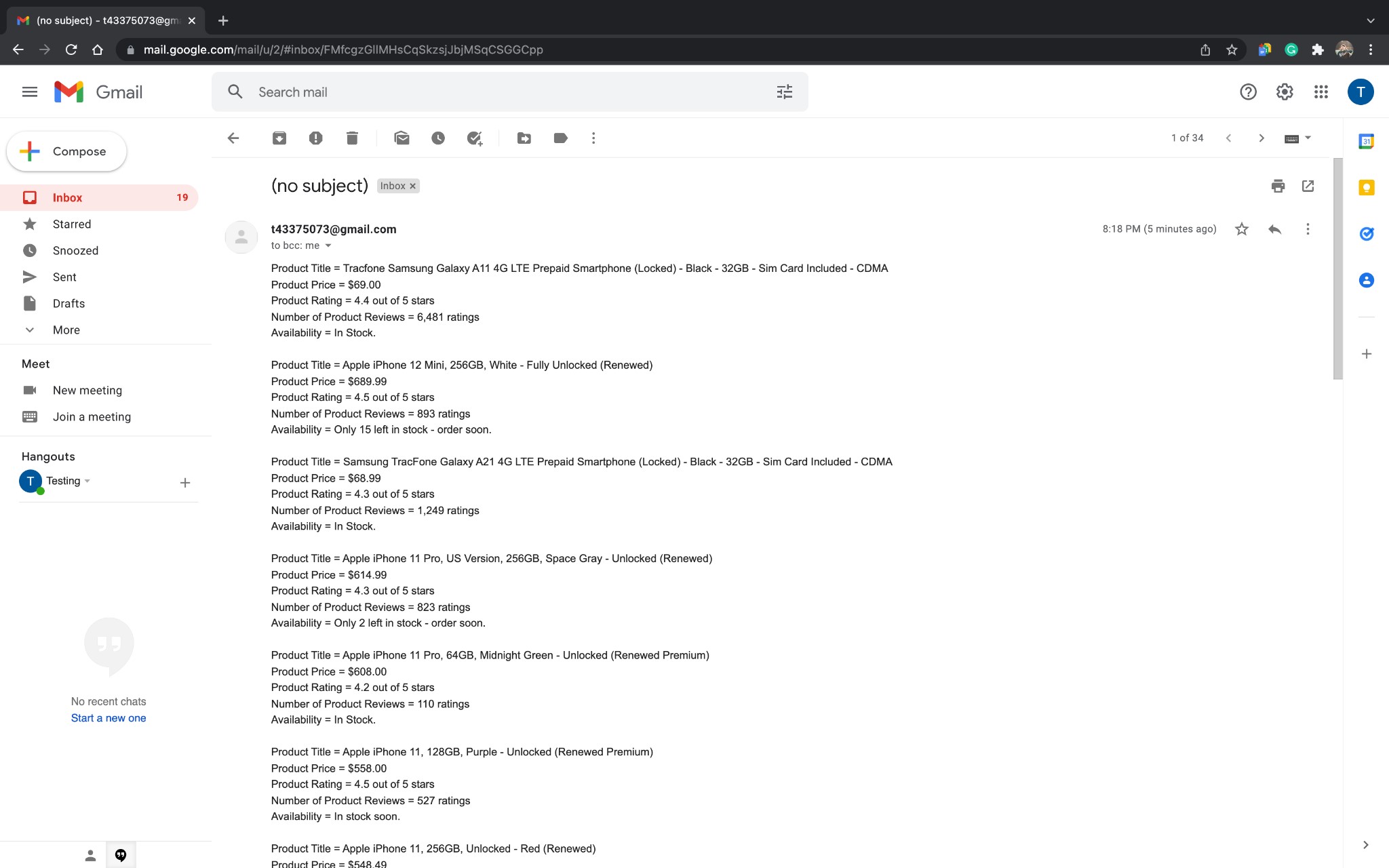
could

be

improved

to

make the operations faster.



# 6. DISCUSSION ON RESULTS (WIRESHARK)

Wireshark is a network protocol analyzer that captures packets from a network connection. It helps understand the network protocols and also it is a tool for debugging protocol implementations. In this study, Wireshark was used to capture packets from the computer where the program was executed. The Wireshark test was used to track two main important features. The request to start web scraping at an amazon website and the retrieval of this information. Lastly, it was used to track the SMTP protocol when an email was sent from our program to a designated email. The results provide the amount of time the web scraping took to retrieve all information and send it back to the program. Also, the timestamps when the email was sent to the designated destination. **6.1 Web Scraping Wireshark Results**

To access the amazon website, an HTTP request was made by the program. A request and reply are needed to initiate a connection. In Figure 5. It can observe the DNS packets retrieved while the program was operating. To maintain a recognizable computer-IP address, amazon.com in our case, is why the DNS protocol was chosen to analyze the packets instead of the HTTP protocol.

## 6.2 SMTP Wireshark Results

The SMTP protocol, Simple Mail Transfer Protocol, is an email protocol used for sending and receiving email messages from a variety of sources such as Outlook, Apple Mail, Gmail, etc. In this study, the receiver and sender were the same email address for testing purposes, and Gmail was used as an account.

An SMTP server is an application that provides service to other applications within the network. Once an SMTP server has been established, email clients can use it to communicate with. It’s built on a TCP connection.

In Figure 6, it is shown the SMTP connection between the two emails. In the first packet, it can be seen the 220 response, which indicates the service is ready. In the second packet “ehlo” Same as HELO but tells the server that the client may want to use the Extended SMTP (ESMTP) protocol instead. The third packet has 250 responses, which means the message was delivered. The fourth packet, STARTLS indicates Start Transport Layer Security. This communication can be monitored and it is also possible to alter the messages that are sent via the routers. Lastly, the 220 response indicates the SMTP is waiting on the client response.



**Fig. 6.** Example of SMTP being shown in Wireshark.

# 7. DIVISION OF LABOR

To ensure that each member of the group had a similar workload, we found it necessary to divide the labor evenly. From this, it was decided that Blair would focus on the code, Tony would focus on the Wireshark measurements, and Logan would focus on any necessary demonstrations and debugging. When it came to creating the presentation and report for this project, we decided that each member should complete the same number of sections. The sections assigned to each member followed what they worked on for the project. For example, Tony worked on the Wireshark portions of the presentation and report as that was his focus when working on the project.

# 8. CONCLUSION

In conclusion, we built a web scraper that gets the title, price, rating, number of ratings, and availability of a product and sends that information to a user as a Gmail message.

In the future, we would like to see if the code will send another email if the lowest price changes. The code has a while loop that is supposed to collect the data again and resend the email with updated information if the price changes. Amazon prices do not fluctuate that often and we did not see the prices change throughout the designing of our project and we were not able to verify if that part of the code works properly or not. Our code is designed to order the products from lowest to highest price. It does not actually send that way. We would address that in the future. We would also add additional selection criteria when it comes to ordering the products. If the price of two products was the same, we would choose the product with the higher rating of the two and list that product first. If the rating was the same, we choose the product with the higher number of ratings. If the products had the same rating, then we would display the first product.

We would also like to add functionality where the user can send an email to the Python code and the Python code can respond in real-time to the user’s request. For example, our code’s default configuration is to organize the products by the lowest price to the highest price in the email message. If the user wanted to see the highest price first, they could send a predetermined message that the Python code could recognize and it could reformat the product list in the email accordingly. This sort of responsiveness would also apply to the user preferring to organize the products by the rating, number of reviews, availability of the product.

We may also expand the software to collect data for the result of a search query on Amazon instead of just the entries on a particular page. This may result in an extremely long email. If that is the case, we may format the results into a PDF and send the PDF in addition to sending the top 5 desired products based on the user’s selected criteria.

# 8. REFERENCES

1. E. Vargiu and M. Urru, [“Exploiting](https://doi.org/10.5430/air.v2n1p44) [web](https://doi.org/10.5430/air.v2n1p44) [scraping in](https://doi.org/10.5430/air.v2n1p44) [a](https://doi.org/10.5430/air.v2n1p44) [collaborative](https://doi.org/10.5430/air.v2n1p44) [filtering-](https://doi.org/10.5430/air.v2n1p44) [based](https://doi.org/10.5430/air.v2n1p44) [approach](https://doi.org/10.5430/air.v2n1p44) [to](https://doi.org/10.5430/air.v2n1p44) [web advertising](https://doi.org/10.5430/air.v2n1p44),” *Artificial Intelligence Research*, vol. 2, no. 1, Jun. 2012.
2. A. Malla, “Amazon price tracker webscraping,” *GitHub*, 10-Jul-2020. [Online]. Available: https://github.com/Anish-Malla/Amzon-price-trac ker-webscraping-. [Accessed: 12-Dec-2021].
3. “How to scrape Amazon product information using beautiful soup,” *JournalDev*, 10-Dec-2020.

[Online]. Available:

https://www.journaldev.com/44473/scrape-amazon -product-information-beautiful-soup. [Accessed: 12-Dec-2021].

1. “Types of scraping techniques: How to prevent scraping,” *Radware Bot Manager*. [Online].

Available:

https://www.radwarebotmanager.com/what-are-the -different-scraping-techniques/. [Accessed: 13-Dec-2021].

1. M. Nagelberg, “Setting up email updates for your scraper using Python and a Gmail account,” *Medium*, 18-Feb-2019. [Online]. Available:

https://towardsdatascience.com/setting-up-email-u pdates-for-your-scraper-using-python-and-a-gmail

-account-4e678be99ed5. [Accessed:

12-Dec-2021].