Cryptocurrency Website Scraper

Divaker Garkoti[#1], Yash Sharma[*2], Yogesh Singh Rajput[#3]
MCA Department, School of Computer Application and Technology
Galgotias University, Greater Noida, Uttar Pradesh, India
[1]garkotid2312@gmail.com, [2]yashkaushik054@gmail.com,
[3]Yogeshsinghrajput54@gmail.com

Supervised by: Mr. Rajesh Sharma, Assistant Professor,
School of Computer Application and Technology
Galgotias University, Greater Noida, Uttar Pradesh, India
rajesh.sharma@galgotiasuniversity.edu.in

***Abstract*— In this modern technological and internet age, data is an integral asset driving innovation and decision-making in every discipline. With the introduction of web scraping as a new powerful tool to extract data, researchers and developers can retrieve huge masses of information dynamically from websites that are not necessarily structured. This paper go through web scraping properties and mechanisms in the context of cryptocurrency, focusing on extracting real-time data concerning cryptocurrencies posted on websites like CoinMarketCap. Using Python libraries such as Selenium, Scrapy, and BeautifulSoup offers a rich set of functionalities to automate tasks and efficiently sift significant data, such as volume, price, and market capitalization. The extracted data is then stored in a MongoDB database and in CSV files for further analysis.**

Integration with Pymongo for linking Python script with the MongoDB database; whereas, data storage is made in the form of CSV files, and hence the Pandas library has been utilized. Moreover, to make interaction possible between the users, a basic GUI was developed using the combination of HTML, CSS, and JavaScript. This interface is connected to the Python script via PHP, providing a seamless workflow for triggering data scraping processes. This study also highlights the problems encountered during web scraping, such as handling dynamically loaded websites that load data using scrolling techniques to load additional data asynchronously and frequent updates of WebDriver to ensure compatibility with browser updates. In addition, it deals with the issues related to managing data accuracy and optimizing the scraping process for consistent results.

***Keywords*— Web scraping, cryptocurrency, CoinMarketCap, real-time data, Selenium, Scrapy, MongoDB, Pymongo, CSV, Pandas, GUI development, HTML, CSS, Javascript, PHP, integration, dynamic websites, data accuracy, asynchronous data loading, data analysis, Web driver.**

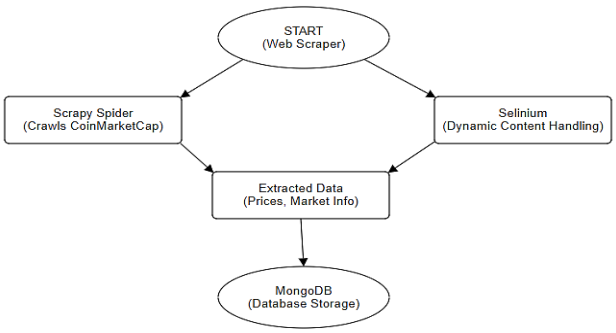
I.INTRODUCTION

Overview of project:

For every new emerging technology, data has slowly and incrementally become a resource for most businesses around the world. Having access to real-time information can greatly empower smarter decisions, especially in today's financial world, where combining cryptocurrency data with solid market analysis is highly valuable. However, much of this data stays on dynamic and unstructured websites that cannot easily be accessed through traditional access means. Web scraping is a powerful technique that will allow for the effective extraction of large volumes of data from such websites.

This paper focuses on web scraping in cryptocurrencies, targeting the extraction of real-time data from coinmarketcap. Using selenium and scrapy with relevant python libraries, it explains how to automate the download of essential financial metrics like price, volume, and market capitalization. Subsequently, the scraped data is stored in the database MongoDB and also CSV files to create structured data that could be analyzed further. It integrates MongoDB into the Python script with the help of Pymongo and uses Pandas for exporting data in tabular format.

HTML, CSS, and JavaScript are also used to create a user-friendly interface for extracting the data. That application can be used to interact with the scraping tool. This interface is linked to the Python script using PHP, creating a seamless process for triggering web scraping tasks. The paper also discusses key challenges, such as managing dynamically loaded websites that require scrolling for additional data and the frequent need to update WebDriver due to browser updates.



Libraries Used:
1. Scrapy: This is an open-source Python framework quite extensively employed in web crawling. That brings a much better method of extracting all the required data from websites and then processing further. In the course of this project, Scrapy will mainly be used for starting the entire process of scraping, managing data flow for collection, and integrating with Selenium in order to scrape the dynamic content.

2. Selenium: This is one of the most powerful browser automation tools, enabling interactions on a web page just as if a user were to act on it in real-time. Its main use is for scraping dynamic content, such as information within JavaScript rendered. Selenium was used in this project to load the coinmarketcap page and scroll through it, so all the data would be rendered and thus be available for scraping.

3. Pymongo: The official Python client for MongoDB is called Pymongo, and it allows Python scripts to communicate with MongoDB databases. It provides database operations like document insertion, querying, and updating. Pymongo is used in this project to harvest cryptocurrency data from CoinMarketCap and store it in MongoDB for later analysis.

4. Pandas: Pandas is a powerful data manipulation and analysis library in Python, especially known for its structure of DataFrame, which makes it easy to deal with tabular data. So in this project, the use of Pandas to store the scraped data in some tabular format can easily convert it into CSV files to export for better analysis and retrieval of the cryptocurrency data in some structured form.

5. CSV- This module is used to read from and write to CSV files that contain tabular data. It is used to store information related to bitcoin scrapes as CSV files, which you can then access and further examine using spreadsheet software.

Objective of the research:
Designing, implementing, and assessing a platform that allows users to mine bitcoin data for analysis and research is one of the primary goals of this study. With data from sites like CoinMarketCap, we hope to create a more practical solution for real-time cryptocurrency data scraping with Python-based frameworks like Selenium and Scrapy. This project will help in achieving the following major goals:

1. Realtime data scrape: Getting volume, market capitalization, price, rank, and circulating supply of such key cryptocurrency metrics from dynamic websites.

2. Data Organization: The scraped data will be in the structured table formats like CSV files to avail for further analyses.

3. Efficient data storage: MongoDB utilization for data storage, so that fast retrieval and efficient management are possible, for more in-depth analytical tasks.

This novelty of research not only helps to extract real-time cryptocurrency data and store it but gives out the fundamental understanding of data scraping techniques that can be employed in a wide variety of applications in the rapidly evolving field of data science and technology.

Literature Survey:
Web scraping has been a critical focus of research in the context of data extraction and automation.

Mdpi.com:
This paper discusses the technological advancements in data extraction and automation, particularly in handling dynamic web content. It was instrumental in shaping the methodology used in scraping cryptocurrency data from dynamic websites such as CoinMarketCap.

ResearchSquare:
This study looks at how to optimise scraping technologies, particularly for asynchronously loading websites. The choice to use Selenium was to manage dynamic scrolling and real-time data loading on cryptocurrency webpages was directly affected by this research.

CoinMarketCap:
The market capitalisation, volume, and real-time pricing data for the majority of cryptocurrencies are available on CoinMarketCap. Recognising the structure and behaviour of the website was essential for this paper's initial emphasis in order to create a suitable scraping approach using the Python tools Scrapy and Selenium.

SCRCS Publications (2023):
This paper discusses the web scraping together with the databases, with the emphasis on the elasticity and scalability of MongoDB. It provided useful data management features that were used to store data from cryptocurrency scraping. The automated online scrapping of such financial data in real time is the objective of this study. The outcomes of the paper were able to make sure that the scraping and the storage of the cryptocurrency data was effective by simplifying the automation and data extraction processes of the project.

Python.org:
The official Python website contains an how-to document about the uses of libraries such as Selenium, Scrapy and Pandas. This platform was crucial during the setup of the

project and in the implementation of the Python based scraping and data storage solutions.

GeeksforGeeks:
GeeksforGeeks additionally provided a step-by-step procedure for scraping tables from websites using the URLSelenium library. Important facets of the tutorial included, empowering automated interactions of a web browser, the ability to parse HTML document and locating the elements of the table and the ability to export efficiently, structured data from the tables into CSV files. Following this guide helped us to better comprehend the internal processes involved in scraping tabulated information from a dynamically generated webpage. This knowledge greatly assisted in collecting and accurate information and systematic data collection for market prices, volumes and rankings of coins from CoinMarketCap.

MongoDB
The MongoDB official documentation of PyMongo driver will be helpful for the establishment of an unchanging link between Python script and MongoDB database. For the operation with the MongoClient object, the guide had simple instructions on how to make a connection, and very detailed descriptions of the features of PyMongo for the execution of operations performed on the database. This documentation has helped us successfully implement the needful connection into the storage of scraped cryptocurrency data into MongoDB; hence, it is very effective for the proper management and retrieval of data towards further analysis purposes.

Mitchell, R. (2020). Web Scraping with Python: Data Extraction from the Modern Web: To begin with web scraping in Python, this book covered, among other things, Selenium, BeautifulSoup or Scrapy, which are essential tools. What was more useful is its practical orientation which concerned creating scraping scripts for garnering data about cryptocurrencies.

Sweigart, A. (2019). Automate the Boring Stuff with Python, 2nd Edition: This book presented a number of important automation methods that assisted one in automating routine operations in the course of this research. It also gives the basis of embedding Python scripts into the interfaces for the MongoDB and for the storage of CSV files with cryptocurrency data.

II.METHODOLOGY

The basis of this research is an efficient collection of cryptocurrency data through web scraping techniques and Python applied as the programming language. All the web scraping processes that are involved in this are fully automated, and hence, the use of Selenium and Scrapy Python libraries is applied. These two well-known libraries provide robust capabilities for data scraping from dynamic websites. The approach utilised to this research is based on what is mentioned below:

Step 1: Configuring the Environment
Version of Python: The tool used for web scraping was Python 3.12.6.
Installing Libraries: This package manager, pip, was used for installing the required libraries; these included Selenium, Scrapy, Pandas, Pymongo, and CSV.

Step 2: Write the Spider using Scrapy
This spider has been created to manage the crawling process of the cryptocurrency website CoinMarketCap. With such a spider, a lot of data may be parsed efficiently with the use of asynchronous requests.

Step 3: Importing Library
Import necessary libraries included Pymongo for interactions with databases, Pandas, which would handle tabular-based data management, Scrapy for the extraction of required data, and Selenium to create an automated browser.

Step 4: Installation and setup of compatible WebDriver
Selenium requires a WebDriver for the interaction with the browser. The WebDriver that would facilitate automation of browsing and scraping was downloaded for Chrome and set up accordingly. The path for the driver was established to ensure it works perfectly.

Step 5: Setting Up the HTTP Request
WebDriver was used to mimic browsing the website, allowing it to send HTTP requests. This helps load dynamic content created with JavaScript.

Step 6: Scraping Data with the Spider
A Scrapy spider was used to explore the CoinMarketCap site, specifically targeting the table displaying cryptocurrency information. It uses XPath to locate specific HTML elements and gather the needed data.

Step 7: Extracting Specific Data
Among many other things, the spider was made to surf the website and scrape all of the primary cryptocurrency data which included rank, name, symbol, market capitalization, price, volume, circulating supply, and percent changes over time.

Step 8: Managing Dynamic Content
In an effort to handle the poor loading of the data, dynamic scrolling was utilized. It ensured that all data had been extracted by slowly scrolling through the webpage as it waited for the loading of more information.

Step 9: Connecting to MongoDB

I used Pymongo to connect to MongoDB. The extracted cryptocurrency data was then stored in a MongoDB collection for future analysis.

Step 10: Storing Data into MongoDB
Every data entry that was extracted was saved to the MongoDB collection to make them easily accessible and processable later.

Step 11: Exporting Data to CSV Files
Along with saving data into the MongoDB, the extracted information was exported to CSV files using the Pandas library. The CSV format provides simple, organized structure that's instantly accessible and then easily analyzed further.

Step 12: Graphic User Interface Design
Using that as the base, HTML, CSS, and JavaScript create a simple GUI. To make the application more understandable to the users, a GUI is implemented.

Step 13: Linking the Python Web Scraper to the GUI
PHP was used to connect the GUI to the Python scraper so that scraping could be initiated directly from the interface. This makes it smooth for the consumers as they can initiate the process of scraping from the front-end.

Implementation:
1. Environment Setup
Installing the necessary libraries to enable data storage and web scraping was the first step in the implementation. Among the libraries used are:

Scrapy is a Python web scraping package that is used to effectively manage the crawling and scraping process.
Installation command for Scrapy:
pip install scrapy

Selenium: An automation tool for a browser that interacts with content on dynamic web pages which load content asynchronously.
Installation command for Selenium:
pip install selenium

Pandas: A library for data manipulation and analysis, used to create and manage data frames and facilitate CSV file writing.
Installation command for Pandas:
pip install pandas

PyMongo: A library for communicating with MongoDB that makes it possible to save the scraped data in an easy-to-use manner.
Installation command for Pymongo:
pip install pymongo

The implementation was executed using Python version 3.12.6, and the Chrome WebDriver was set up to allow Selenium to control the Chrome browser.

2. Database Connection
A connection to the MongoDB database was established using the following code:

```
client =
MongoClient("mongodb+srv://garkotid***:*******@proj
ects.f6mcr.mongodb.net/")
db = client.cryptocurrency
```

3. Data Insertion Function
To manage the insertion of cryptocurrency data that has been scraped into the MongoDB collection, a function called insertToDB was defined. A document with the necessary fields—rank, name, symbol, market capitalisation, price, circulating supply, volume, and percentage changes over various time periods—is created by the function.

```
def insertToDB(rank, name, symbol, market_cap, price,
circulating_supply, volume, percent_1h, percent_24h,
percent_7d):
    collection = db.coinmarketcap
    post = {
        'Rank': rank,
        'Name': name,
        'Symbol': symbol,
        'Market Cap': market_cap,
        'Price': price,
        'Circulating Supply': circulating_supply,
        'Volume': volume,
        '% 1h': percent_1h,
        '% 24h': percent_24h,
        '% 7d': percent_7d
    }
    inserted = collection.insert_one(post).inserted_id
    print(f"Inserted document with ID: {inserted}")
    return inserted
```

4. Scrapy Spider Implementation
The core part of the project is CoinMarketCapSpider - a class inherited from Scrapy's Spider and encapsulating all web scraping logic. A spider goes ahead and initiates a Selenium WebDriver instance for interacting with the CoinMarketCap website.

```
class CoinMarketCapSpider(scrapy.Spider):
    name = 'coinmarketcap'
    CHROME_DRIVER_PATH = 'H:\\chromedriver.exe'
##chrome Driver Path
    start_urls = ['https://coinmarketcap.com/all/views/all/']
## website url
```

The __init__ method creates a CSV file to hold the scraped data locally and initialises the Chrome driver. The following headers make up the structure of the CSV file:

```
with open(self.data_file, 'w', newline='', encoding='utf-8') as f:
        writer = csv.writer(f)
        writer.writerow(['Rank', 'Name', 'Symbol', 'Market Cap', 'Price', 'Circulating Supply', 'Volume', '% 1h', '% 24h', '% 7d'])
    print("CSV file initialized.")
```

5. Data Scraping Logic

The spider's parse method implements the fundamental logic for data scraping. This technique obtains the website, loads all of the content via scrolling, and retrieves the necessary info from the bitcoin information table. Data points including rank, name, symbol, market capitalisation, price, circulating supply, volume, and percentage changes over one hour, twenty-four hours, and seven days are captured by processing each table row.

6. Data Storage

After successfully scraping the data, it is then saved in a CSV file and in the MongoDB database. New points of data will be appended to the CSV file but with MongoDB you can access tha data fast for further analysis.

```
def parse(self, response):
    print("Starting to parse the webpage...")
    self.driver.get(response.url)
    last_height = self.driver.execute_script("return document.body.scrollHeight")
    print(f"Initial page height: {last_height}")
     while True:
            print("Looking for table rows...")
            rows = self.driver.find_elements(By.XPATH, '//table/tbody/tr')
            print(f'Number of rows found: {len(rows)}')
            new_data = []
            for row in rows:
                    try:
                    rank = row.find_element(By.XPATH, './td[1]/div').text
                     name = row.find_element(By.XPATH, './td[2]/div/a[2]').text
                      symbol = row.find_element(By.XPATH, './td[3]/div').text
                     market_cap = row.find_element(By.XPATH, './td[4]/p/span[2]').text
                     price = row.find_element(By.XPATH, './td[5]/div/span').text
                    circulating_supply = row.find_element(By.XPATH, './td[6]/div').text
                    volume = row.find_element(By.XPATH, './td[7]/a').text
                    percent_1h = row.find_element(By.XPATH, './td[8]/div').text
                    percent_24h = row.find_element(By.XPATH, './td[9]/div').text
                    percent_7d = row.find_element(By.XPATH, './td[10]/div').text
                    row_id = (rank, name, symbol, market_cap, price, circulating_supply, volume, percent_1h, percent_24h, percent_7d)
    if row_id not in self.scraped_set:
        self.scraped_set.add(row_id)
        new_data.append({
                'Rank': rank,
                'Name': name,
                'Symbol': symbol,
            'Market Cap': market_cap,
                'Price': price,
        'Circulating Supply': circulating_supply,
                 'Volume': volume,
                '% 1h': percent_1h,
                 '% 24h': percent_24h,
                  '% 7d': percent_7d
                })
         print(f"Scraped data for {name} ({symbol})")
              insertToDB(rank, name, symbol, market_cap, price, circulating_supply, volume, percent_1h, percent_24h, percent_7d)
            except Exception as e:
                print(f'Error scraping row: {e}')
```

7. Completion of Scraping

Upon completion of the scraping process, the Chrome WebDriver is closed, and a message indicating the completion of the scraping is printed to the console.

```
self.driver.quit()
    print('Scraping completed.')
```

III.RESULTS

This study effectively show that real-time cryptocurrency data can be scraped from websites like CoinMarketCap, processed, and stored in a MongoDB database and CSV files for later examination. A number of important results were used to evaluate the implementation:

1. Effective Extraction of Data

Using Python, Selenium, and Scrapy, the web scraping solution we recorded important bitcoin metrics in real time. Information was gathered for a number of fields, including:

Rank: The position of cryptocurrency's to its market capitalisation.
Name: The cryptocurrency's name and associated symbol.
Market Cap: Each cryptocurrency's total market capitalisation.
Price: The cryptocurrency's current price.
Circulating supply: supply available in the market.
Volume: The amount of trading activity during the past 24 hours.
Percentage Change: Over the course of one hour, twenty-four hours, and seven days, the price percentage varies.

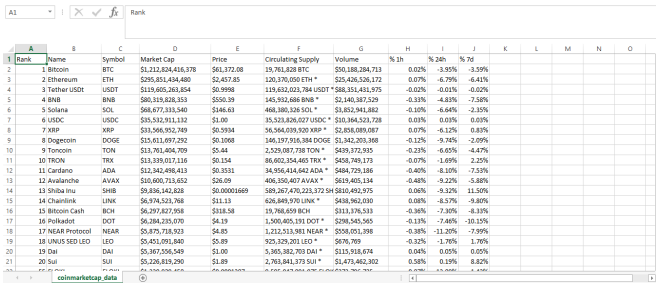
The scraper handler navigates through dynamically loaded content with ease. The automation of scrolling ensures that no piece of information is left out, and thus an immense compilation of bitcoin data was amassed.

2. Data Management and Storage:
Successfully stored the two formats of scraped data:
CSV Files: All the data is kept in CSV files, which can be analyzed offline with the help of spreadsheet programs like Excel.
Working with data in a table format is simple and convenient.
MongoDB: MongoDB is a NoSQL database that stores data efficiently, making it easy to access and manage. It handled large amounts of cryptocurrency data effectively, allowing for smooth querying and analysis.

3. Real-Time Data Access
Getting real-time cryptocurrency data is crucial for accurate market analysis. With the help of Selenium and Scrapy, the platform always recorded the latest data by efficiently handling dynamic content.

4. Integration of User Interface
In order to make the tool user-friendly, a basic user interface was implemented. The connection would allow users to view and interact with real-time cryptocurrency data by ensuring a smooth and continuous flow of data from collection to display. The tool would be usable by non-programmers through a "Start Scrape" button on the front-end interface, where users may actually initiate the web scraping process.

5. Data Scalability
The tool can scrape large volumes of cryptocurrency data without compromising on performance. It was successful in scaling to capture all entries, even with incremental scrolling, as there is a huge amount of data on the CoinMarketCap website.
Below is a snippet of the extacted data that is stored in the csv file.



Rank	Name	Symbol	Market Cap	Price	Circulating Supply	Volume	% 1h	% 24h	% 7d
1	Bitcoin	BTC	$1,212,824,416,378	$61,372.08	19,761,828 BTC	$50,188,284,713	0.02%	-3.95%	-3.39%
2	Ethereum	ETH	$295,851,434,480	$2,457.85	120,376,050 ETH *	$25,426,526,172	0.07%	-6.79%	-6.41%
3	Tether USDt	USDT	$119,605,263,854	$0.9998	119,632,023,784 USDT *	$88,351,431,975	-0.02%	-0.01%	-0.02%
4	BNB	BNB	$80,319,828,353	$550.39	145,932,686 BNB *	$2,140,387,529	-0.33%	-4.83%	-7.58%
5	Solana	SOL	$68,677,333,540	$146.63	468,380,326 SOL *	$1,852,941,882	-0.10%	-6.64%	-2.35%
6	USDC	USDC	$35,512,911,132	$1.00	35,523,826,027 USDC *	$10,364,523,728	0.03%	0.03%	0.03%
7	XRP	XRP	$33,566,952,749	$0.5934	56,564,039,920 XRP *	$2,858,089,087	0.07%	-6.12%	0.83%
8	Dogecoin	DOGE	$15,611,697,292	$0.1068	146,197,916,384 DOGE	$1,342,203,368	-0.12%	-9.74%	-2.09%
9	Toncoin	TON	$13,761,404,709	$5.44	2,529,087,738 TON *	$439,372,935	-0.21%	-6.65%	-4.47%
10	TRON	TRX	$13,339,017,116	$0.154	86,602,354,465 TRX *	$458,749,173	-0.07%	-1.69%	2.25%
11	Cardano	ADA	$12,342,498,413	$0.3531	34,956,414,642 ADA *	$484,729,186	-0.40%	-8.10%	-7.53%
12	Avalanche	AVAX	$10,600,713,652	$26.09	406,350,407 AVAX *	$619,405,134	-0.48%	-9.22%	-5.88%
13	Shiba Inu	SHIB	$9,836,142,828	$0.00001669	589,267,470,223,372 SH	$810,492,975	0.06%	-9.32%	11.50%
14	Chainlink	LINK	$6,974,523,768	$11.13	626,849,970 LINK *	$438,962,030	0.08%	-8.57%	-9.80%
15	Bitcoin Cash	BCH	$6,297,827,958	$318.58	19,768,659 BCH	$313,376,533	-0.36%	-7.30%	-8.33%
16	Polkadot	DOT	$6,284,235,070	$4.19	1,500,405,191 DOT *	$298,545,565	-0.13%	-7.46%	-10.15%
17	NEAR Protocol	NEAR	$5,875,718,923	$4.85	1,212,513,981 NEAR *	$558,051,398	-0.38%	-11.20%	-7.99%
18	UNUS SED LEO	LEO	$5,451,091,840	$5.89	925,329,201 LEO *	$676,769	-0.32%	-1.76%	1.76%
19	Dai	DAI	$5,367,556,549	$1.00	5,365,382,703 DAI *	$115,918,674	0.04%	0.05%	0.05%
20	Sui	SUI	$5,226,819,290	$1.89	2,763,841,373 SUI *	$1,473,462,302	0.58%	0.19%	8.82%

IV.CONCLUSION

This paper is successful in showing how a trustworthy cryptocurrency web scraping tool can be designed and deployed. Using Python's powerful libraries, including Selenium, Scrapy, and Pymongo, it is possible to scrape dynamic websites such as CoinMarketCap in real time.

By utilising HTML, CSS, and JavaScript we created a more user-friendly graphical interface, the tool's integration with the web scraper enabled smooth user-web scraper interaction and further simplified the process to make it accessible and usable for users with comparatively little technical expertise.
It then addressed such challenges as large datasets, dynamic page loading, and error handling and in so doing improved its efficiency and reliability. This gave way to a scalable solution for collecting real-time cryptocurrency data and laid a solid foundation for future developments such as advanced analysis and integration with financial tools.
This platform is a good way of automating cryptocurrency data mining that provides valuable insights to analysts and researchers with minimal manual effort. Reliability, scalability, and cost-effectiveness would characterize collection, storage, and analysis of large volumes of real-time cryptocurrency data.

V.ACKNOWLEDGEMENT

We would like to thank Mr Rajesh Sharma, Assistant Professor, MCA Department, School of Computer Application and Technology, Galgotias University, for the valuable guidance, constant encouragement, and insightful suggestions along the length of this research. The mentorship by him has helped in channelizing the study's direction and in overcoming technical challenges that were many.
Finally, we are happy to thank Galgotias University for providing all of these resources and infrastructure in front of us that led to us completing this project. Most importantly, we want to thank our friends and colleagues, who encouraged us all throughout the study.
We are all greatly thankful for the authors and developers of the various Python tools, libraries, and documentation that we used in this paper. This project would never have been a reality without all of these libraries.

VI.REFERENCES

1. Almalki, M. (2023). The Role of Web-Scraping Technology to Analyze Cryptocurrency Data. Electronics, 13(14), 2700. doi:10.3390/electronics13142700
2. Alotaibi, M., & Alabdulqader, M. (2023). Improving cryptocurrency trading using web-scraping and machine learning strategies. Res. Square. Retrieved from https://www.researchsquare.com/article/rs-3854342/v1
3. Jain, R. K., & Shukla, A. (2023). Automatic Extraction and Analysis of Data in Cryptocurrency. International Journal of Management Research and Social Science,
https://www.publications.scrs.in/uploads/final_menuscript/9ad9e5cc9555d1b867d2d74ed6114724.pdf

4. Patel, V. (2023). CryptoCompetitor “A Review Paper” in Automated Tools Used in CryptoCurrency. International Journal of Modern Research in Science and Technology, 6(1), 17-21. Retrieved from http://www.ijmrset.com/upload/26_Automation.pdf
5. GeeksforGeeks. (2023). Scrape table from website using Python Selenium. https://www.geeksforgeeks.org/scrape-table-from-website-using-python-selenium/
6. MongoDB. Overview of The Driver For Python’s Pymongo Module. Retrieved 2 October 2024, from https://www.mongodb.com/docs/languages/python/pymongo-driver/current/connect/mongoclient/#:~:text=API%20Documentation-,Overview,you%20perform%20operations%20on%20it.
7. Mitchell, R. (2018). Web Scraping With Python: The Next Generation Of Web Data Extraction (2nd ed.). O'Reilly Media.
8. Sweigart, A. (2019). Automate the Boring Stuff with Python: Practical Programming for Complete Beginners.
9. Yuan, S. (2023). Design and visualization of web scraping using python through external libraries and tools of selenium. Academic Journal of Computing & Information Systems. Available at francis-press.com.
10. Nariman, D. (2024, April 10). Increasing the performance as well as the accuracy of customer review data harvesting through a multi threaded web scraping approach.