A FOCUSED INTELLIGENT CRAWLER FOR DYNAMIC CRAWLING

PROJECT REPORT

Submitted by

PADHMA M [152IT164]

In partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

BANNARI AMMAN INSTITUTE OF TECHNOLOGY
(An Autonomous Institution Affiliated to Anna University, Chennai)
SATHYAMANGALAM-638401

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2019

BONAFIDE CERTIFICATE

Certified that this project report **"A FOCUSED INTELLIGENT CRAWLER FOR DYNAMIC CRAWLING"** is the bonafide work of **"PADHMA M [152IT164]"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.A.BHARATHI	Ms.K.R.SABARMATHI
HEAD OF THE DEPARTMENT	**ASSISTANT PROFESSOR**
Department of Information Technology	Department of Information Technology
Bannari Amman Institute of Technology	Bannari Amman Institute of Technology
Sathyamangalam	Sathyamangalam

Submitted for Project Viva Voce examination held on ……

INTERNAL EXAMINER EXTERNAL EXAMINER

DECLARATION

I affirm that the project work titled **"A FOCUSED INTELLIGENT CRAWLER FOR DYNAMIC CRAWLING"** being submitted in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Information Technology** is the record of original work done by me under the guidance of **Ms.K.R.Sabarmathi**, Assistant Professor**,** Department of Information Technology. It has not formed a part of any other project work submitted for the award of any degree or diploma, either in this or any other University.

PADHMA M

[152IT164]

I certify that the declaration made above by the candidate is true.

Ms.K.R.Sabarmathi

ASSISTANT PROFESSOR

ACKNOWLEDGEMENT

I would like to enunciate heartfelt thanks to our esteemed Chairman **Dr.S.V.Balasubramaniam**, and the respected Director **Dr.M.P.Vijaykumar**, for providing excellent facilities and support during the course of study in this institute.

I am grateful to **Dr.A.Bharathi**, Head of the Department, Information Technology for her valuable suggestions to carry out the project work successfully.

I wish to express our sincere thanks to **Mr.R.Vinothsaravanan,** Assistant Professor, for his constructive ideas, inspirations, encouragement and much needed technical support extended to complete my project work.

I wish to express my sincere thanks to Faculty guide **Ms.K.R.Sabarmathi,** Assistant Professor, Information Technology for her constructive ideas, inspirations, encouragement, excellent guidance and much needed technical support extended to complete my project work.

I would like to thank my friends, faculty and non-teaching staff who have directly and indirectly contributed to the success of this project.

PADHMA M [152IT164]

ABSTRACT

Given the complexity of web and its contents, data in the web poses unpredictable challenges for the crawlers and search engines. Due to the explosive growth of the web pages, generalized crawlers are no longer enough to run on the web efficiently. Hence, the need for selective crawlers becomes necessary. There are many focused crawlers in wide use; however, none of them is suitable for template-customized topical crawling. Focused Crawler's main aim is to selectively seek out pages that are relevant to pre-define set of topics rather than to exploit all regions of web. Web users are increasingly feeling a need for highly specialized and filtered Search interfaces. It filters at the data-acquisition level, rather than as a post-processing step. The system selects work very carefully from the crawl frontier. As a result, it is feasible to crawl to a greater depth. This leads in the discovery of some high-quality information resources that might have otherwise been overlooked. Modern websites make use of complex scripts rather than just HTML code to render data. These scripts make the webpage to load dynamically and provides greater user experience. In such cases, the complexity of the website is greater than traditional sites. Hence, traditional crawlers cannot be used in this process. We need an intelligent crawler which will understand the type of the website, its content and its behavior and react accordingly while following the politeness policy, efficiency, reliability and synchronous transmission of data. The proposed work aims to achieve all these functionalities.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The capacity to browse through the content of World Wide Web in a mechanized way is performed by web crawler. It is additionally alluded to as the spiders, web robots, programmed indexers, bots and ants. The procedure that is executed by a web crawler is frequently alluded to as web crawling or spidering. Many web crawlers use the procedure so as to give refreshed data about the quantity of sites that are added to the World Wide Web. A web index utilizes the program so as to store the most visited pages of a site which are handled for later utilization. A web crawler is frequently utilized by real web indexes as in mechanized upkeep procedure to look at an approval of HTML code. It likewise can look at for data from various WebPages so as to gather email addresses. The crawler as a rule visits the URLs of a site, which are regularly characterized as hyperlinks in the page. It can download a constrained amount in a given arrangement of time. Because of this reason, a web crawler is frequently made to organize the rundown of downloads. Any adjustment in the sites will regularly make such a circumstance. Because of the huge development of World Wide Web, web indexes are nearly covering a little measure of the complete substance that is accessible openly. A review did in the year 2005 demonstrated that significant web crawlers file around a limit of 70% of the all-out web zone. The need of the site depends on the usefulness, the traffic, the prominence of connections present in it and the page rank. Web crawlers are additionally known to download WebPages that are especially in comparative. This procedure is named as topical crawlers or cantered crawler. The execution of this specific strategy will incredibly rely upon the fame of the connection for the point that is being sought. It for the most part relies upon the web index for the underlying beginning of the hunt. Web crawler can likewise be utilized for limiting connections, normalizing a URL, execute way climbing creeping and return to arrangement that helps isolate the measure of data present over the Internet. One of the most popular search engines, Google uses more than a billion crawlers. Once the user types their queries, millions of crawlers are released at a time which crawls through the world wide web one step at a time and returns the user entered information. In the beginning phases, google was an academic search engine which used only 21 crawlers to crawl through web pages. Since then, there has been rapid growth across various domains. Hence, the need for an efficient crawler also keeps on increasing. To perform website streamlining, web crawlers assume an urgent job. At the point when the web bug returns home, the information is recorded by the web index. All the principle web crawlers, for example, Google and Yahoo, use crawlers to

fabricate and amend their lists. In addition, the bugs or crawlers are utilized for computerizing upkeep errands on the site. For instance, the crawlers are utilized to approve HTML code, accumulate particular sorts of information from the sites, and check connects on the site page to different locales. Besides, what the crawler sees on a site will decide how the site is recorded in its list. The web search tools discover a site's significance dependent on a complex scoring framework, which they attempt to keep mystery. They are a virtual answer for "creeping and scouring the web" to locate the best locales for motivations behind ordering and rank. Utilizing watchwords normally, making standard extra and changes, and concentrating on consistency go far toward supporting the web crawlers into ordering your webpage such that will support you. Besides Web crawling is the automated traversal of web to collect all the useful informative pages, effectively and efficiently to gather information about link structure interconnecting those informative pages. Its working is explained as follows: Its pre-samples few pages to discover the repetitive regions.

Using some reliable crawling algorithms, a focused web crawler is developed which will retrieve and download the content of the webpage which is dynamically loaded. In the past decade, most of the websites were static. Developing a crawler for a static webpage was challenging in those days. But the webpages developed nowadays are dynamically loaded which uses dynamic content due to various requirements. This approach focuses mainly to retrieve data from a dynamically loaded web page by making use of the concepts of virtualization and containerization. The virtualized platform enables a way to access data which is dynamically loaded by JavaScript by making use of some light weight web browsers. The obtained data can be of several uses including for machine learning and data science. Also keeping in mind some traditional crawling strategy such as politeness policy, reliability, accessibility and approach. The main approach is to create a crawler which packages all the parts of the engine into one single package and handles them appropriately.

CHAPTER 2
LITERATURE SURVEY

2.1 ANATOMY OF WEB CRAWLERS

World wide web is the source of massive collection of information which has different types of data which has potential value. To access and use this information, search engines are used. For this process, the search engines rely on massive number of crawlers. The crawler also known as the spider or a bot crawls through web pages and retrieves the user entered data. The crawlers download the webpages by traversing the ever-changing web content which is dense and hyperlinked by nature. It stores the downloaded pages in large repositories which are indexed for execution upon queried by the user. Various web crawler architectures are proposed over the years. This paper proposes a survey on different crawler architectures by analysing some features such as scalability, manageability, page refresh policy, politeness policy etc. Since the web and its content is growing, it becomes extremely important to build efficient crawlers by considering different perspectives and their impacts.

2.2 FOCUSED WEB CRAWLERS AND ITS APPROACHES

With the evolution of the world wide web each and every second, a challenge is put forth which poses unpredictable. Hence, focused web crawlers can be used for the purpose of retrieving data which is limited to specific sites and predefined topics rather than searching for the entire web which save large amount of time and space. This paper focuses on five different types of crawlers such as: Priority base crawler, Structured base crawler, Leaning base crawler, Context base crawler and Other focused crawler. An experimental approach has been proposed, which exposes the secrets of the search engine and its impacts in the ranking functions which is a basic priority for crawling a web page. In case of precision, efficiency, accessibility, topical crawling, a general crawler has so many limitations because of no specificity. Assessment is performed among trendy and focused web crawlers to apprehend which one is higher and also discussed about the merits of numerous strategies like precedence based in addition to contextual based targeted crawling. The benefits of targeted crawler are that we spend less cash, time & attempt processing WebPages which are most not likely to be of cost or worth.

2.3 KEYWORD FOCUSED WEB CRAWLER

Users and purpose of web is developing enormously nowadays which causing an extraordinary inconvenience and endeavours at client side to get website pages sought which are according to concern and significant to client's requirement. Generally, clients way to deal with pursuit site pages from a huge accessible pecking order of ideas or utilize a question to peruse pages from accessible internet searcher and get results dependent on inquiry design where few of the outcomes are pertinent to hunt and the greater part of them are most certainly not. Web crawler assumes a critical job in web search tool and go about as a key component when execution is considered. This paper incorporates area designing idea and catchphrase driven creeping with importance choice system and utilizations Ontology ideas which guarantees the best way to improve crawler's execution. This paper presents extraction of URLs dependent on watchword or pursuit criteria. It separates URLs for website pages which contains sought catchphrase in their substance and considers such pages just as vital and doesn't download site pages immaterial to seek. It offers high optimality contrasting and conventional web crawler and can improve look proficiency with more precision.

2.4 A DISTRIBUTED VERTICAL CRAWLER USING CRAWLING-PERIOD BASED STRATEGY

Because of the unstable development of the pages, concentrated crawlers are never again adequate to keep running on the web effectively. There are many circulated crawlers in wide use; be that as it may, none of them is appropriate for layout tweaked vertical creeping. In this paper, an approach is presented for an appropriated format of vertical crawler which is exceptionally utilized for crawling Internet discussions. The Client-Server engineering of the framework and the capacity of each module are depicted in detail which can be stretched out to different fields effectively. A crawling period-based appropriation methodology is additionally proposed, with which the crawler chief can facilitate the number of crawling errands and the assets of every crawler, and the crawler can process sites with various refreshing recurrence adaptably. They likewise characterize a correspondence convention among crawlers and crawler administrator and portray how to take care of the copied creeping issue in the appropriated framework. The execution of unified vertical crawler and disseminated vertical crawler are thought about in the test. Test results exhibit that the parallel task of the considerable number of crawlers in the disseminated framework can incredibly improve the crawling effectiveness. The authors have depicted the design and execution

subtleties of our dispersed framework and introduced some fundamental exploratory outcomes. This disseminated crawler has been utilized to gather the Internet discussion data and performed well. This crawler can be effectively stretched out to gather websites. Contrasted with the incorporated vertical crawler, the disseminated one procedures URL more effectively and adaptably. The crawling time frame-based technique is demonstrated powerful in preparing sites with various update recurrence, which can spare the assets of re-creeping. Also, the correspondence convention between the crawler director and crawlers characterized in this paper can be a general convention for the circulate arrangement of format-based crawlers.

2.5 DESIGN AND APPLICATION OF INTELLIGENT DYNAMIC CRAWLER FOR WEB DATA MINING

Web information procurement is the establishment of Web information mining. Web crawler is a critical device for Web information procurement, yet the successive updates of Web information structures, information sources and dispersion channels, brought about mind-boggling expenses of crawler program improvement and support. To take care of this issue, this paper structured and executed an astute unique crawler, which put away the information extraction standards of XPath in database, stacked the principles progressively as indicated by the objective, and utilized TF-IDF technique to ascertain the pertinence. The Web creeping principles can be consequently procured, which made the crawler wise and dynamic, improved the flexibility of the crawler for the mind-boggling web condition, and decreased the support and update cost. At long last, this paper applies the astute unique crawler to the danger attention to open vulnerabilities, with the technique for information accumulation and investigation of the weakness network and the system hub web crawler. The analysis utilized the model framework on three powerlessness networks to gather and dissect the information. The outcomes demonstrated that the shrewd unique crawler can understand the high-productive and adaptable information accumulation work and established the framework for Web information mining.

CHAPTER 3
SYSTEM ANALYSIS

EXISTING SYSTEM:

In the current system, there is an approach for a focused crawler and a key focused crawler. A crawler to retrieve the dynamic contents are also developed. But there is no effective combination which a focused crawler and retrieves the dynamic content by taking into account the complexity of complicated websites. The process of Focused web crawling is used for finding pages which is satisfying some property that is related to some specific topics. With the help of Focused crawler approach, it tries to fetch as much relevant page as possible with the higher accuracy level. The goal is achieved by, precisely prioritizing the already crawled pages and managing the exploration of hyperlinks. A focused crawler ideally would like to download only web pages that are relevant to a particular topic and avoid downloading all others. It predicts the probability that a link to a particular page is relevant before actually downloading the page. A possible predictor is the anchor text of links. In another approach, the relevance of a page is determined after downloading its content. Relevant pages sent to content indexing and their contained URLs added to the crawl frontier; pages that fall below a relevance threshold are discarded. The previous approach of information retrieval is like a black box; Search system has limited information of user needs. The user context and their environment are ignored resulting in irrelevant search result. This type of system increase overhead to the user in filtering useful information. In fact, contextual relevance of document should also be considered while searching of document. represents the outer-view of contextual driven search system. In terms of precision and efficiency General Crawler has some limitation because of its generality, no specialty. Although some focused crawler overcome this limitation, they lack efficiency.

CHAPTER 4

PROPOSED SYSTEM

Although there are several types of crawlers available, focused crawler finds its importance by being selective and topical in its approach. It becomes extremely important to crawl selective websites and obtain its content which finds its application in various disciplines such as machine learning, deep learning, data science, etc., Websites and its content are loaded dynamically due to the growing importance of dynamic websites. Hence, a focused crawler which can load the web pages dynamically and retrieve its content is mandatory. The approach of a focused crawler is more straightforward than other types of crawlers. Hence, in this approach, a focused crawler is developed with efficiency and certainty in the retrieved content while following the politeness policy which every crawler must follow. The process is developed in a virtual environment in order to increase the efficiency of the crawler and also to segregate the crawler from being affected by the outside environments. It becomes necessary to create this application in a virtual environment. Given the advantage of virtual environment, the crawler can function independently. In sustenance with the virtual environment, the crawler is also containerized properly by appropriate packaging. This helps in facing future challenges. Once the URL is specified for the crawler, it starts crawling based on crawling period strategy. The crawler can also work in a dynamically loaded environment which is rendered by JavaScript where the script loads in the client side directly by fetching data from the server. Hence, this approach can also serve modern web pages and their behaviour. With the evolution of complex data in the world wide web, this crawler can help in being intelligent while retrieving the web and its content.

CHAPTER 5

SYSTEM REQUIREMENTS

5.1 HARDWARE REQUIREMENTS

- RAM – 4.00 GB
- System Type – 64-bit Operating System, x64-based processor
- Processor – Dual core or above
- Keyboard – Standard Keyboard

5.2 SOFTWARE REQUIREMENTS

- Windows 10 Home
- Python 3.7
- Terminal or PowerShell
- IDE – Atom 1.35.1
- python virtual environment 2.0
- Anaconda 4.6.8
- Docker 18.09.3
- Oracle VM virtual Box – 6.0.4
- pip3
- scrapy 1.6.0
- splash 2.0
- scrapy-splash 1.8

CHAPTER 6
SYSTEM DESIGN AND IMPLEMENTATION

6.1 LIST OF MODULES

6.1.1 Virtual Environment

6.1.2 Conda Package Manager

6.1.3 Scrapy overview and Architecture

6.1.4 Crawler Architecture

6.1.5 Containerization

6.1.6 Splash

6.1.1 VIRTUAL ENVIRONMENT

The entire system is implemented using python. Python is preferred due to its enormous standard libraries and the simplicity in implementation. Python has notable versatile features such as being dynamic, interpreted, modular, portable, high level, object-oriented, interactive and it is also extensible in C and C++. It also makes integration much easier for web services and tools. Python also provides support for developing most of the engine parts of the spider through its incredible framework. Hence, the system design begins with installation of Python 3.7. The latest version of python is mostly preferred. Given the complexity of spider and its functionality, python as a multi paradigm programming language provides feasibility and accountability by making complicated code easy to write and implement. After the basic installation of python, a virtual environment is created for the project.

A virtual environment is a software that keeps conditions required by various activities separate by making segregated python virtual situations for them. This is a standout amongst the most vital devices that most of the Python designers use. As a matter of course, every task on your framework will utilize the equivalent indexes to store and recover site bundles (outsider libraries). What if two versions of the same software or library is used? This is a genuine issue for Python since it can't separate between adaptations in the "site-bundles" index. So, both the versions would live in a similar catalogue with a similar name. This is the place virtual conditions become possibly the most important factor. To tackle this issue, we simply need to

make two separate virtual conditions for both the projects utilizing different version of the same library or software. The incredible thing about this is there are no restrictions to the quantity of situations you can have since they're simply registries containing a couple of contents. The working and installation of virtual environment is as follows:

1. Open a terminal directing to the specific project which you want under the virtual environment.
2. In this project, a separate virtual environment is created for only the crawler and its dependencies.
3. Next, the pip package manager is setup.
4. The virtual environment is created using the *virtualenv* package and by typing the following command in the prompt.

   $ pip install virtualenv

5. To test the Installation, we type in the following command:
   $ virtualenv --version
6. To create a virtual environment, a path is specified.
   $ virtualenv scrapy_venv
7. To specify the interpreter of our choice, the following command is used:
   $ virtualenv -p /usr/bin/python3 scrapy_venv
8. The virtual environment is activated by the following command:
   $ source scrapy_venv/Scripts/activate
9. After the setup, the virtual environment is deactivated using:
   $ scrapy_venv deactivate

6.1.2 CONDA PACKAGE MANAGER

Conda is a package management system which is used in cross platform fashion for working with python packages. Although several projects use anaconda and it is the highly recommended package manager, our project requirements can be met by miniconda as it consumes less memory and installs the right package. Miniconda contains only python and other libraries needed to run conda. Its genuine quality comes in dealing with Python bundles which require aggregated code– such bundles are particularly predominant in the logical Python biological community. The miniconda package manager is downloaded from the

official conda website for windows and it is installed by agreeing to the terms and conditions. Once miniconda is installed, the following command can be used to verify its installation:

$ conda -- version

After the requirements are met by installing miniconda, the scrapy framework is installed for creating the spider.

6.1.3 SCRAPY OVERVIEW AND ARCHITECTURE

Web scraping has turned into a powerful method for extricating data from the web for basic mining and examination. It has turned into a basic piece of the information science toolbox. Information researchers should realize how to accumulate information from website pages and store that information in various configurations for further investigation. Any website page you see on the web can be crept for data and anything unmistakable on a site page can be separated. Each site page has its very own structure and web components that because of which you have to compose your web crawlers/bots as per the website page being extricated. Scrapy is a quick, open-source web crawling system written in Python, used to extricate the information from the page with the assistance of selectors dependent on XPath or CSS. Scrapy project architecture is built around "spiders", which are self-contained crawlers that are given a set of instructions. Following the spirit of other don't repeat yourself frameworks, such as Django it makes it easier to build and scale large crawling projects by allowing developers to reuse their code. Scrapy also provides a web-crawling shell, which can be used by developers to test their assumptions on a site's behaviour. Scrapy provides a powerful framework for extracting the data, processing it and then save it. Scrapy provides a complete package where code maintenance is not an issue. Scrapy can get tough and larger jobs done easily. Scrapy utilizes spiders, which are independent crawlers that are given a lot of directions. In Scrapy it is simpler to construct and scale substantial slithering undertakings by enabling engineers to reuse their code. It can crawl and process a set of URLs in less than a minute based on the size of the group and does this process subtly which works in an asynchronous manner for concurrent processing. Scrapy also supports by providing item pipelines that process URL and data by validating, removing HTML tags and saving the content to a storage device. Scrapy can transfer data to a CSV file, JSON file or even to a database. It can move from one URL to another even before the previous URL returns the response. This makes scrapy more reliable. It is an open source framework, which has built in support to extract data by making use of CSS Selectors or XPATH selectors. Since it is based on crawler, it can download and save the

webpage in an automatic manner. It comes with built in support called scrapyd which can unload projects and control the spider using JSON. The framework also provides a shell called scrapy shell which can help in debugging and test the assumptions on a website's behaviour. Scrapy shell can be used to check what components the page returns and how it can select them for crawling. The installation of scrapy can be done with miniconda using the following command:

```
$ conda install -c conda-forge scrapy
```

With python 3.0 or greater, this command can be used. While the lesser versions make use of pip which is a python package manager itself.

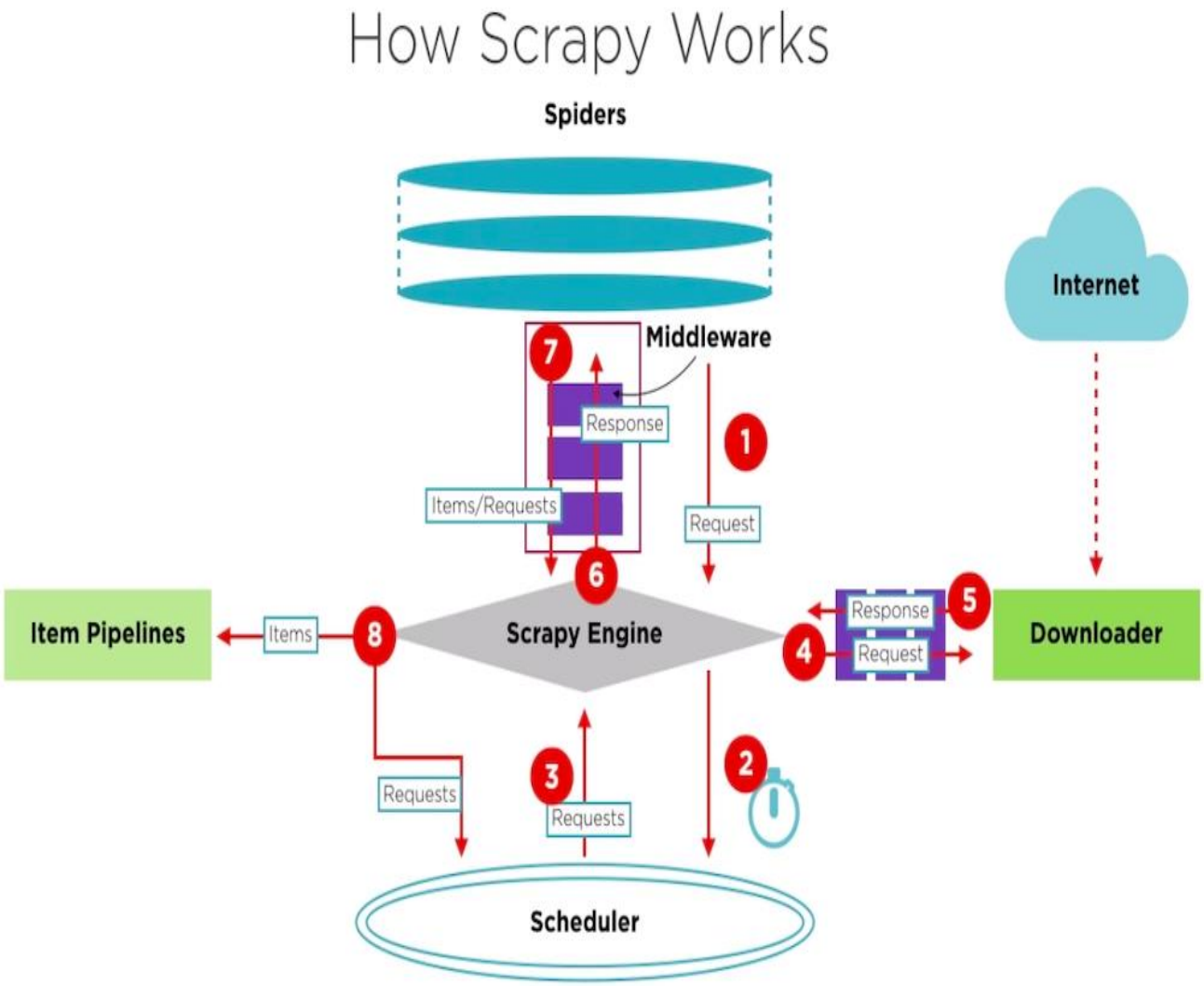

Figure 6.1 Scrapy Architecture

6.1.4 CRAWLER ARCHITECTURE

The crawler consists of various modules such as:

6.1.4.1 Spider

6.1.4.2 Engine

6.1.4.3 Downloader

6.1.4.4 Scheduler

6.1.4.5 Item pipelines

6.1.4.6 Items

6.1.4.1 SPIDER

Spiders are classes which define how a certain site (or a group of sites) will be scraped, including how to perform the crawl (i.e. follow links) and how to extract structured data from their pages (i.e. scraping items). In other words, Spiders are the place where you define the custom behaviour for crawling and parsing pages for a site (or, in some cases, a group of sites). For spiders, the scraping cycle goes through something like this:

1. You start by generating the initial Requests to crawl the first URL and specify a call-back function to be called with the response downloaded from those requests.

   The first requests to perform are obtained by calling the start_request() method which (by default) generates request for the URLs specified in the start_urls and the parse method as callback function for the Requests.
2. In the callback function, you parse the response (web page) and return either dicts with extracted data, item objects, request objects, or an iterable of these objects. Those Requests will also contain a callback (maybe the same) and will then be downloaded by Scrapy and then their response handled by the specified callback.
3. In callback functions, you parse the page contents, typically using Selectors (but you can also use BeautifulSoup, lxml or whatever mechanism you prefer) and generate items with the parsed data.
4. Finally, the items returned from the spider will be typically persisted to a database (in some Item Pipeline) or written to a file using Feed exports.

Scrapy.spider is the simplest spider, and the one from which every other spider must inherit (including spiders that come bundled with Scrapy, as well as spiders that you write yourself). It doesn't provide any special functionality. It just provides a default start_request() implementation which sends requests from the start_urls() spider attribute and calls the spider's method parse for each of the resulting responses. A string which defines the name for this spider. The spider name is how the spider is located (and instantiated) by Scrapy, so it must be unique. However, nothing prevents you from instantiating more than one instance of the same spider. This is the most important spider attribute and it's required. If the spider scrapes a single domain, a common practice is to name the spider after the domain. An optional list of strings containing domains that this spider is allowed to crawl. Requests for URLs not belonging to the domain names specified in this list (or their subdomains) won't be followed if offsite middleware is enabled. A list of URLs where the spider will begin to crawl from, when no particular URLs are specified. So, the first pages downloaded will be those listed here. The subsequent request will be generated successively from data contained in the start URLs. This attribute is set by the from_crawler() class method after initialization the class, and links to the crawler object to which this spider instance is bound. Crawlers encapsulate a lot of components in the project for their single entry access (such as extensions, middlewares, signals managers, etc).

6.1.4.2 ENGINE:

The engine is responsible for controlling the data flow between all components of the system, and triggering events when certain actions occur. The execution engine, which coordinates the core crawling logic between the scheduler, downloader and spiders. Some extension may want to access the Scrapy engine, to inspect or modify the downloader and scheduler behaviour, although this is an advanced use and this API is not yet stable.

6.1.4.3 DOWNLOADER:

The Downloader is responsible for fetching web pages and feeding them to the engine which, in turn, feeds them to the spiders.

6.1.4.4 SCHEDULER

The Scheduler receives requests from the engine and enqueues them for feeding them later (also to the engine) when the engine requests them.

6.1.4.5 ITEM PIPELINES

After an item has been scraped by a spider, it is sent to the Item Pipeline which processes it through several components that are executed sequentially. Each item pipeline component (sometimes referred as just “Item Pipeline”) is a Python class that implements a simple method. They receive an item and perform an action over it, also deciding if the item should continue through the pipeline or be dropped and no longer processed. Typical uses of item pipelines are:

- cleansing HTML data
- validating scraped data (checking that the items contain certain fields)
- checking for duplicates (and dropping them)
- storing the scraped item in a database

The Item Pipeline is responsible for processing the items once they have been extracted (or scraped) by the spiders. Typical tasks include cleansing, validation and persistence (like storing the item in a database).

6.1.4.6 ITEMS

Scrapy process can be used to extract the data from sources such as web pages using the spiders. Scrapy uses Item class to produce the output whose objects are used to gather the scraped data. The item objects can be specified using the following class which provides the new initialized item from the given argument

class scrapy.item.Item([arg])

The field objects can be specified using the following class in which the Field class doesn't issue the additional process or attributes

Class scrapy.item.Field([arg]

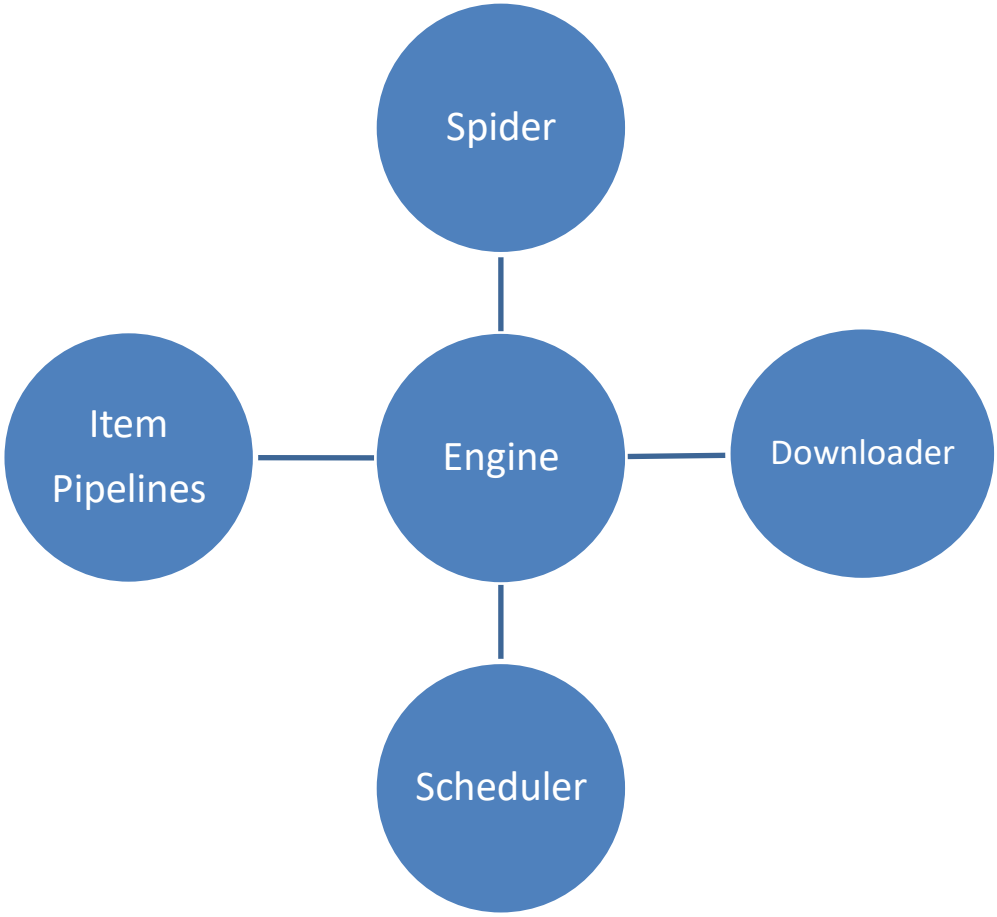

Figure 6.2 crawler Architecture

After the project is setup, a project is created using scrapy from the terminal by using the following steps:

$ scrapy startproject crawler

This will create a folder for the crawler structure. After the folder is created, the spider is generated using the following command which will create all the dependencies for the spider.

$ scrapy genspider MarketStats

Few things to note here:

- **Name:** Name of the spider, in this case it is “MarketStats”. Naming spiders properly becomes a huge relief when you have to maintain hundreds of spiders.

- **allowed_domains**: An optional list of strings containing domains that this spider is allowed to crawl. Requests for URLs not belonging to the domain names specified in this list won't be followed.
- **parse (self, response):** This function is called whenever the crawler successfully crawls a URL. After every successful crawl the ***parse(..)*** method is called.

The crawl method is called so that the spider starts crawling the specified webpage. Since the crawler is focused in this case, we restrict its accessibility within Economic Times which is a stock data and financial data news listing website.

```
$ scrapy crawl MarketStats
```

After this command, a challenge is encountered which is fixed in the following section.

6.1.5 CONTAINERIZATION

Containerization is a lightweight alternative to a virtual machine that involves encapsulating an application in a container with its own operating system. A container takes its meaning from the logistics term, *packaging container*. When we refer to an application container, we mean packaging software. The foundation of Containerization lies in the Linux Container (LXC) Format. This is why containers only work with Linux based machines and can only run Linux applications. On the other hand, the traditional hypervisors like VMWare, Virtual Box, etc can run on Windows, Linux as well as all those operating systems that support hypervisors. Another difference is that containers share the same kernel as the host machine which is not in the case of hypervisors. Application containers, such as Docker comprised of the application files, dependencies and libraries of an application to run on an OS. Docker is an open source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools. Docker Inc., the company that originally developed Docker, supports a commercial edition and is the principal sponsor of the open source tool. Docker is a tool that packages, provisions and runs containers independent of the OS. Container technology is available through the operating system: A container packages the application service or function with all of the libraries, configuration files, dependencies and other necessary parts to operate. Each container shares the services of one underlying operating system. Docker was created to work on the Linux platform, but has extended to offer greater support for non-Linux operating systems, including Microsoft Windows and Apple OS X. Versions of Docker for Amazon Web

Services (AWS) and Microsoft Azure are available. Docker consists of several major components. Docker Community Edition is open source, while Docker Enterprise Edition is a version offered by Docker Inc. Enterprise Edition versions include Basic, Standard and Advanced. The Docker Engine is the underlying client-server tool that supports container technology to handle the tasks and workflows involved in building container-based applications. The engine creates a server-side daemon process that hosts images, containers, networks and storage volumes. The daemon also provides a client-side command-line interface (CLI) that allows users to interact with the daemon through the Docker application programming interface. Containers created by Docker are called Docker files. Docker Compose files define the composition of components in a Docker container. Docker is a tool that is designed to benefit both developers and system administrators, making it a part of many DevOps (developers + operations) toolchains. For developers, it means that they can focus on writing code without worrying about the system that it will ultimately be running on. It also allows them to get a head start by using one of thousands of programs already designed to run in a Docker container as a part of their application. For operations staff, Docker gives flexibility and potentially reduces the number of systems needed because of its small footprint and lower overhead. With a traditional hypervisor approach, each virtual machine (VM) needs its own operating system, but with Docker, applications operate inside a container that resides on a single host operating system that can serve many different containers. Docker containers are designed to run on everything from physical computers to virtual machines, bare-metal servers, OpenStackcloud clusters, public instances and more. Docker extends the LinuX Containers (LXC) format, which serves to provide an isolated environment for applications, by enabling image management and deployment services.

The installation and setup for docker is as follows:

1. Download docker for windows which requires Microsoft hyper -V to run.
2. After the download is complete, run the setup.
3. After the setup is done, open the docker quickstart terminal which will show the version of the package installed and also the docker symbol is in running status which we can find in the taskbar.

Docker has some prerequisites which are Windows10 64bit, Enterprise or education or later, CPU SLAT-capable feature, at least 4GB of RAM, virtualization enabled in BIOS.

6.1.6 SPLASH

A common roadblock when developing spiders is dealing with sites that use a heavy amount of JavaScript. Many modern websites run entirely on JavaScript and require scripts to be run in order for the page to render properly. In many cases, pages also present modals and other dialogues that need to be interacted with to show the full page. View Source in browsers typically only displays the downloaded source without running anything at all (including any JS that would modify the DOM). Tools like Firebug for Firefox, WebKit's Web Inspector and Opera's Dragonfly are indispensable tools for front-end Web developers, letting you view the DOM as the browser sees it. And that's where these tools can actually cause some problems, or at least a bit of confusion. When you choose "Inspect element" or otherwise bring up one your browser's DOM inspector, what you're looking at is the document tree after the browser has applied its error correction and after any JavaScript's have manipulated the DOM. It is not necessarily the same as what you see when you choose "View source". DOM is different in all browsers. DOM inspectors are not view source. Browsers use different error correction (at least until they all fully support HTML5), which is one of the reasons Validation matters.

To overcome this challenge, Splash is used. Splash is a JavaScript rendering service. It's a lightweight web browser with an HTTP API, implemented in Python 3 using Twisted and QT5. The (twisted) QT reactor is used to make the service fully asynchronous allowing to take advantage of webkit concurrency via QT main loop. Some of Splash features:

- process multiple webpages in parallel
- get HTML results and/or take screenshots
- turn OFF images or use Adblock Plus rules to make rendering faster
- execute custom JavaScript in page context
- write Lua browsing scripts
- develop Splash Lua scripts in Splash-Jupyter Notebooks
- get detailed rendering info in HAR format.

Setting up splash:

```
$ docker pull scrapinghub/splash

$ docker run -p 5023:5023 -p 8050:8050 -p 8051:8051 scrapinghub/splash
```

Splash will now be running on localhost:8050. Since a Docker Machine is used on Windows, it will be running on the IP address of Docker's virtual machine. In order to run splash through docker, port forwarding is done. Port forwarding, or tunnelling, is the behind-the-scenes process of intercepting data traffic headed for a computer's IP/port combination and redirecting it to a different IP and/or port. A program that's running on the destination computer (host) usually causes the redirection, but sometimes it can also be an intermediate hardware component, such as a router, proxy server or firewall. Of course, even though anyone sending data to a server isn't aware of what's going on, the request will still get to its ultimate destination. Port forwarding is an excellent way to preserve public IP addresses. It can protect servers and clients from unwanted access, "hide" the services and servers available on a network, and limit access to and from a network. Port forwarding is transparent to the end user and adds an extra layer of security to networks. In short, port forwarding is used to keep unwanted traffic off networks. It allows network administrators to use one IP address for all external communications on the Internet while dedicating multiple servers with different IPs and ports to the task internally. Port forwarding is useful for home network users who may wish to run a Web server or gaming server on one network. The network administrator can set up a single public IP address on the router to translate requests to the proper server on the internal network. By using only one IP address to accomplish multiple tasks—and dropping all traffic that is unrelated to the services provided at the firewall—the administrator can hide from the outside world what services are running on the network. When configuring port forwarding, the network administrator sets aside one port number on the gateway for the exclusive use of communicating with a service in the private network, located on a specific host. External hosts must know this port number and the address of the gateway to communicate with the network-internal service. Often, the port numbers of well-known Internet services, such as port number 80 for web services (HTTP), are used in port forwarding, so that common Internet services may be implemented on hosts within private networks.

For the purpose of port forwarding, oracle VM virtualbox is used. VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2. VirtualBox is being actively developed with frequent releases and has an ever-growing list of features, supported guest operating systems and platforms it runs on. Oracle VM VirtualBox is a lightweight application that allows you to run virtual machines (VMs) on a variety of host operating systems. Following Oracle's acquisition of Sun Microsystems, VirtualBox was officially renamed Oracle VM VirtualBox, and in February 2011, the company released version 4.0.4. VirtualBox is free and open source, but there's also a free, closed source extension pack available with additional VirtualBox features. Oracle VM VirtualBox is a lightweight application that allows you to run virtual machines (VMs) on a variety of host operating systems. Following Oracle's acquisition of Sun Microsystems, VirtualBox was officially renamed Oracle VM VirtualBox, and in February 2011, the company released version 4.0.4. VirtualBox is free and open source, but there's also a free, closed source extension pack available with additional VirtualBox features. In this work, port forwarding is done as follows:

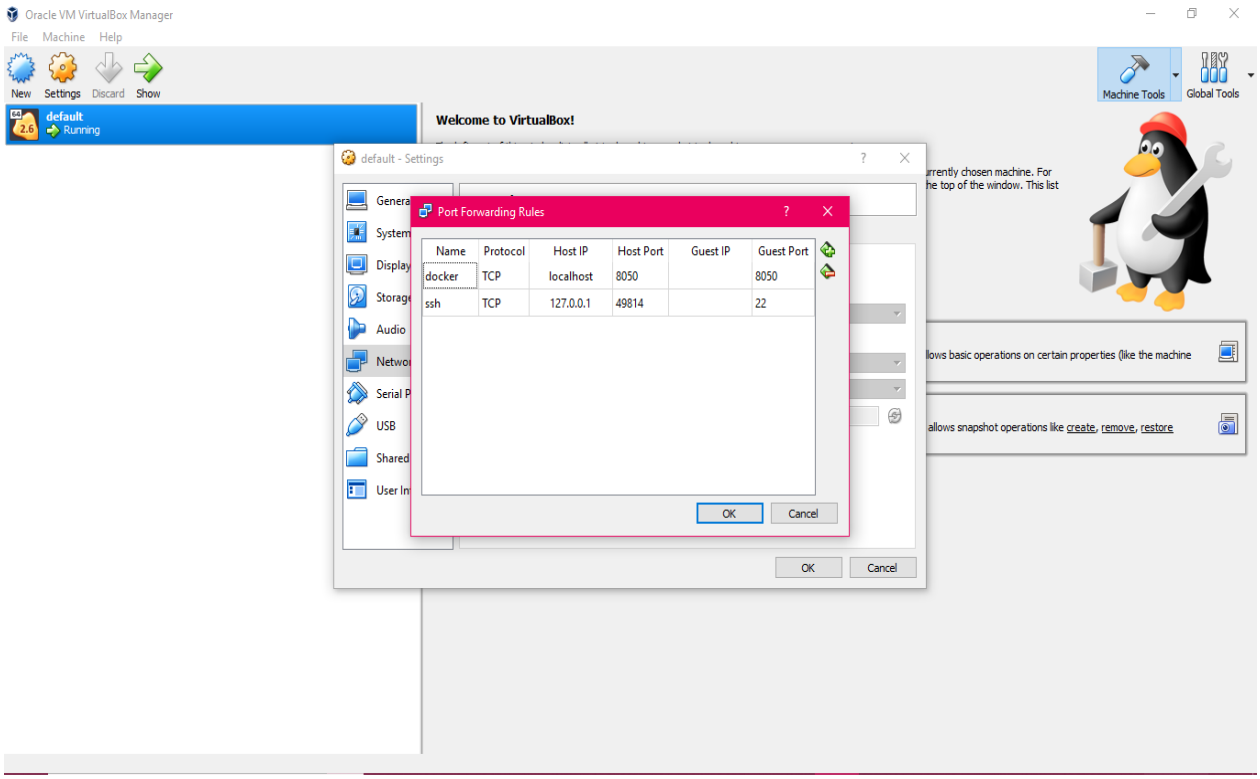

Figure 6.3: Port Forwarding

After this process, we tend to integrate splash with scrapy. The preferred way to integrate Splash with Scrapy is using scrapy-splash. You use the middleware instead of using it manually. One can install scrapy-splash using pip:

```
$ pip install scrapy-splash
```

Add the Splash server address to settings.py of your Scrapy project like this:

```
SPLASH_URL = 'http://192.168.59.103:8050'
```

Enable the Splash middleware by adding it to DOWNLOADER_MIDDLEWARES in your settings.py file and changing HttpCompressionMiddleware priority:

```
DOWNLOADER_MIDDLEWARES = {
  'scrapy_splash.SplashCookiesMiddleware': 723,
   'scrapy_splash.SplashMiddleware': 725,
   'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware': 810,
}
```

Order 723 is just before HttpProxyMiddleware (750) in default scrapy settings. HttpCompressionMiddleware priority should be changed in order to allow advanced response processing.

Enable SplashDeduplicateArgsMiddleware in your splash settings for config purposes by adding it to SPIDER_MIDDLEWARES in settings.py:

```
SPIDER_MIDDLEWARES = {
    'scrapy_splash.SplashDeduplicateArgsMiddleware': 100,
}
```

This middleware is needed to support cache_args feature; it allows to save disk space by not storing duplicate Splash arguments multiple times in a disk request queue. If Splash 2.1+ is used the middleware also allows to save network traffic by not sending these duplicate arguments to Splash server multiple times.

Set a custom DUPEFILTER_CLASS:
DUPEFILTER_CLASS = 'scrapy_splash.SplashAwareDupeFilter'

To use Scrapy HTTP cache then a custom cache storage backend is required. scrapy-splash provides a subclass ofscrapy.contrib.httpcache.FilesystemCacheStorage:
HTTPCACHE_STORAGE = 'scrapy_splash.SplashAwareFSCacheStorage'

In case of other cache storage then it is necessary to subclass it and replace all scrapy.util.request.request_fingerprint calls with scrapy_splash.splash_request_fingerprint. Note

Above Steps are necessary because Scrapy doesn't provide a way to override request fingerprints calculation algorithm globally; this could change in future.

There are also some additional options available. Put them into your settings.py to change the defaults:

- SPLASH_COOKIES_DEBUG is False by default. Set to True to enable debugging cookies in the SplashCookiesMiddleware. This option is similar to COOKIES_DEBUG for the built-in scarpy cookies middleware: it logs sent and received cookies for all requests.
- SPLASH_LOG_400 is True by default - it instructs to log all 400 errors from Splash. They are important because they show errors occurred when executing the Splash script. Set it to False to disable this logging.
- SPLASH_SLOT_POLICY is scrapy_splash.SlotPolicy.PER_DOMAIN (as object, not just a string) by default. It specifies how concurrency & politeness are maintained for Splash requests, and specify the default value for slot_policy argument for SplashRequest.

Splash itself is stateless - each request starts from a clean state. In order to support sessions the following is required:

- client (Scrapy) must send current cookies to Splash;
- Splash script should make requests using these cookies and update them from HTTP response headers or JavaScript code;
- updated cookies should be sent back to the client;

- client should merge current cookies wiht the updated cookies.

Now that the Splash middleware is enabled, you can use *SplashRequest* in place of *scrapy.Request* to render pages with Splash. Once a SplashRequest is sent, we will be able to fetch the dynamically loaded JavaScript as Splash runs scripts.

The following command is used to fetch the data.

```
$ scrapy crawl MarketStats
```

The content fetched is displayed in the console. In order to transfer it to a storage file, we can use the following command.

```
$ scrapy crawl MarketStats -o data.csv
```

Which stores the data in a CSV (comma separated values) file. (or)

```
$ scrapy crawl MarketStats -o data.json
```

Which stores the data in a JSON (JavaScript Object Notation) file.

Thus, the implementation of a focused crawler for crawling a dynamic modern website is completed. This crawler finds its applications in various disciplines.

CHAPTER 7

CONCLUSION

In terms of precision and efficiency General Crawler has some limitation because of its generality, no specialty. Precision and recall of expert search on web are improved by focused crawler. With the help of Focused crawler only selective and retrieve relevant page is collected in of all the pages. In search engine application Web crawling is one of the main components. The advantages of focused crawler are that we spend less money, time & effort processing Web Pages that are most unlikely to be of value or worth. Main advantage of using keyword focused web crawler over other web crawler is that it works intelligently, efficiently and doesn't need relevance feedback. It reduces number of extracted web pages thus it takes less time for crawling as it downloads relevant web pages only. Intension is to retrieve relevant web pages and discards the irrelevant web pages. This one also proposes a way to retrieve content from the website rather than HTML tags. This crawler can face several challenges as we keep in mind the development of modern websites while implementing. Hence, this approach finds application in data science, machine learning etc. In future, this crawler can also be used for data mining, data analysis, data prediction and visualization purposes.

APPENDIX 1

SOURCE CODE

MarketStats.py:

```
import scrapy
from scrapy_splash import SplashRequest
class MarketstatsSpider(scrapy.Spider):
    name = 'MarketStats'
    allowed_domains = ['www.economictimes.indiatimes.com']
    def start_requests(self):
        urls = [
            'https://economictimes.indiatimes.com/marketstats/pageno-1,pid-59,sortby-currentYearRank,sortorder-asc,year-2018.cms',
            'https://economictimes.indiatimes.com/marketstats/pid-58,pageno-1,year-2017,sortorder-asc,sortby-CurrentYearRank.cms',
            'https://economictimes.indiatimes.com/marketstats/pid-57,pageno-1,year-2016,sortorder-asc,sortby-CurrentYearRank.cms',
            'https://economictimes.indiatimes.com/marketstats/pid-56,pageno-1,year-2015,sortorder-asc,sortby-CurrentYearRank.cms',
            'https://economictimes.indiatimes.com/marketstats/pid-55,pageno-1,year-2014,sortorder-asc,sortby-CurrentYearRank.cms',
        ]
        for url in urls:
            yield SplashRequest(url, self.parse, args={'wait': 3})
    def parse(self, response):
        for stats in response.xpath("//div[@class='dataList']"):
```

```
        yield {
            'Rank': stats.css(".dataList ul>li:nth-child(n+1)::text").extract_first(),
            'Company': stats.css(".dataList ul>li:nth-child(n+3)>a::text").extract_first(),
            'Revenue': stats.css(".dataList ul>li:nth-child(n+4)::text").extract_first(),
            'PAT': stats.css(".dataList ul>li:nth-child(n+6)::text").extract_first(),
            'MCAP': stats.css(".dataList ul>li:nth-child(n+8)::text").extract_first(),
        }
```

Items.py:

```
# -*- coding: utf-8 -*-
# Define here the models for your scraped items
# See documentation in:
# https://doc.scrapy.org/en/latest/topics/items.html
import scrapy
class StatsItem(scrapy.Item):
    # define the fields for your item here like:
    companyName = scrapy.Field()
    rank = scrapy.Field()
    name = scrapy.Field()
    revenue = scrapy.Field()
    pat = scrapy.Field()
    mcap = scrapy.Field()
```

middlewares.py:

```
# -*- coding: utf-8 -*-
```

```
# Define here the models for your spider middleware
# See documentation in:
# https://doc.scrapy.org/en/latest/topics/spider-middleware.html
from scrapy import signals
class CrawlerSpiderMiddleware(object):
    # Not all methods need to be defined. If a method is not defined,
    # scrapy acts as if the spider middleware does not modify the
    # passed objects.
    @classmethod
    def from_crawler(cls, crawler):
        # This method is used by Scrapy to create your spiders.
        s = cls()
        crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
        return s
    def process_spider_input(self, response, spider):
        # Called for each response that goes through the spider
        # middleware and into the spider.
        # Should return None or raise an exception.
        return None
    def process_spider_output(self, response, result, spider):
        # Called with the results returned from the Spider, after
        # it has processed the response.
        # Must return an iterable of Request, dict or Item objects.
```

```
        for i in result:
            yield i
    def process_spider_exception(self, response, exception, spider):
        # Called when a spider or process_spider_input() method
        # (from other spider middleware) raises an exception.
        # Should return either None or an iterable of Response, dict
        # or Item objects.
        pass
    def process_start_requests(self, start_requests, spider):
        # Called with the start requests of the spider, and works
        # similarly to the process_spider_output() method, except
        # that it doesn’t have a response associated.
        # Must return only requests (not items).
        for r in start_requests:
            yield r
    def spider_opened(self, spider):
        spider.logger.info('Spider opened: %s' % spider.name)
class CrawlerDownloaderMiddleware(object):
    # Not all methods need to be defined. If a method is not defined,
    # scrapy acts as if the downloader middleware does not modify the
    # passed objects.
    @classmethod
    def from_crawler(cls, crawler):
```

```
    # This method is used by Scrapy to create your spiders.
    s = cls()
    crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
    return s
def process_request(self, request, spider):
    # Called for each request that goes through the downloader
    # middleware.
    # Must either:
    # - return None: continue processing this request
    # - or return a Response object
    # - or return a Request object
    # - or raise IgnoreRequest: process_exception() methods of
    #   installed downloader middleware will be called
    return None
def process_response(self, request, response, spider):
    # Called with the response returned from the downloader.
    # Must either;
    # - return a Response object
    # - return a Request object
    # - or raise IgnoreRequest
    return response
def process_exception(self, request, exception, spider):
    # Called when a download handler or a process_request()
```

```
        # (from other downloader middleware) raises an exception.
        # Must either:
        # - return None: continue processing this exception
        # - return a Response object: stops process_exception() chain
        # - return a Request object: stops process_exception() chain
        pass
    def spider_opened(self, spider):
        spider.logger.info('Spider opened: %s' % spider.name)
```

pipelines.py:

```
# -*- coding: utf-8 -*-
# Define your item pipelines here
class CrawlerPipeline(object):
    def process_item(self, item, spider):
        return item
```

settings.py:

```
# -*- coding: utf-8 -*-
# Scrapy settings for crawler project
# For simplicity, this file contains only settings considered important or
# commonly used. You can find more settings consulting the documentation:
#     https://doc.scrapy.org/en/latest/topics/settings.html
#     https://doc.scrapy.org/en/latest/topics/downloader-middleware.html
```

```
#     https://doc.scrapy.org/en/latest/topics/spider-middleware.html
BOT_NAME = 'crawler'
SPIDER_MODULES = ['crawler.spiders']
NEWSPIDER_MODULE = 'crawler.spiders'
# scrapysplash settings
SPLASH_URL = 'http://localhost:8050'
DOWNLOADER_MIDDLEWARES = {
   'scrapy_splash.SplashCookiesMiddleware': 723,
   'scrapy_splash.SplashMiddleware': 725,
   'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware':  810,
}
SPIDER_MIDDLEWARES = {
   'scrapy_splash.SplashDeduplicateArgsMiddleware': 100,
}
DUPEFILTER_CLASS = 'scrapy_splash.SplashAwareDupeFilter'
# Crawl responsibly by identifying yourself (and your website) on the user-agent
#USER_AGENT = 'crawler (+http://www.yourdomain.com)'
# Obey robots.txt rules
ROBOTSTXT_OBEY = True
# Configure maximum concurrent requests performed by Scrapy (default: 16)
#CONCURRENT_REQUESTS = 32
# Configure a delay for requests for the same website (default: 0)
# See https://doc.scrapy.org/en/latest/topics/settings.html#download-delay
```

```
# See also autothrottle settings and docs
DOWNLOAD_DELAY = 3
# The download delay setting will honor only one of:
CONCURRENT_REQUESTS_PER_DOMAIN = 16
CONCURRENT_REQUESTS_PER_IP = 16
# Disable cookies (enabled by default)
COOKIES_ENABLED = False
# Disable Telnet Console (enabled by default)
TELNETCONSOLE_ENABLED = False
# Override the default request headers:
DEFAULT_REQUEST_HEADERS = {
'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
'Accept-Language': 'en',
}
Enable or disable spider middlewares
See https://doc.scrapy.org/en/latest/topics/spider-middleware.html
SPIDER_MIDDLEWARES = {
'crawler.middlewares.CrawlerSpiderMiddleware': 543,
}
Enable or disable downloader middlewares
See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html
DOWNLOADER_MIDDLEWARES = {
'crawler.middlewares.CrawlerDownloaderMiddleware': 543,
```

```
}

# Enable or disable extensions

# See https://doc.scrapy.org/en/latest/topics/extensions.html

EXTENSIONS = {

'scrapy.extensions.telnet.TelnetConsole': None,

}

# Configure item pipelines

# See https://doc.scrapy.org/en/latest/topics/item-pipeline.html

ITEM_PIPELINES = {

        'crawler.pipelines.CrawlerPipeline': 300,

}

# Enable and configure the AutoThrottle extension (disabled by default)

# See https://doc.scrapy.org/en/latest/topics/autothrottle.html

AUTOTHROTTLE_ENABLED = True

# The initial download delay

AUTOTHROTTLE_START_DELAY = 5

# The maximum download delay to be set in case of high latencies

AUTOTHROTTLE_MAX_DELAY = 60

# The average number of requests Scrapy should be sending in parallel to

# each remote server

AUTOTHROTTLE_TARGET_CONCURRENCY = 1.0

# Enable showing throttling stats for every response received:

AUTOTHROTTLE_DEBUG = False
```

Enable and configure HTTP caching (disabled by default)

See https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#httpcache-middleware-settings

HTTPCACHE_ENABLED = True

HTTPCACHE_EXPIRATION_SECS = 0

HTTPCACHE_DIR = 'httpcache'

HTTPCACHE_IGNORE_HTTP_CODES = []

HTTPCACHE_STORAGE = 'scrapy.extensions.httpcache.FilesystemCacheStorage'

Scrapy.cfg:

Automatically created by: scrapy startproject

For more information about the [deploy] section see:

https://scrapyd.readthedocs.io/en/latest/deploy.html

[settings]

default = crawler.settings

[deploy]

#url = http://localhost:6800/

project = crawler

APPENDIX 2

SCREENSHOTS

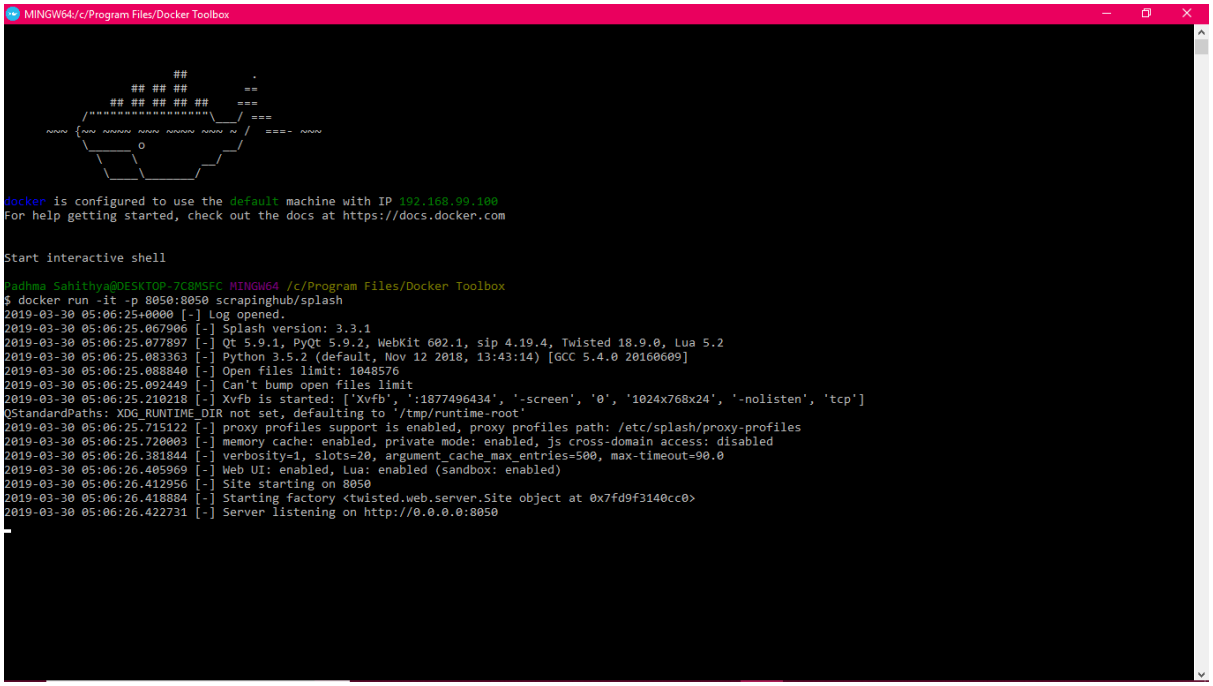

Figure A.1: Splash Server

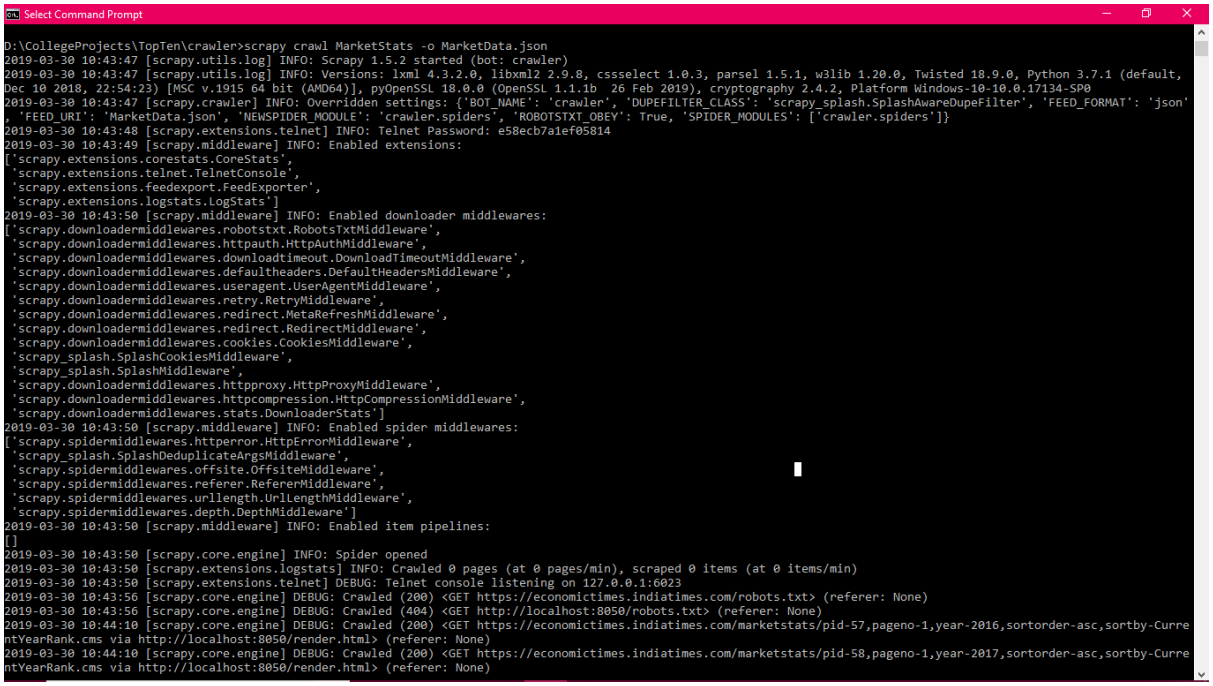

Figure A.2: Crawl Begin

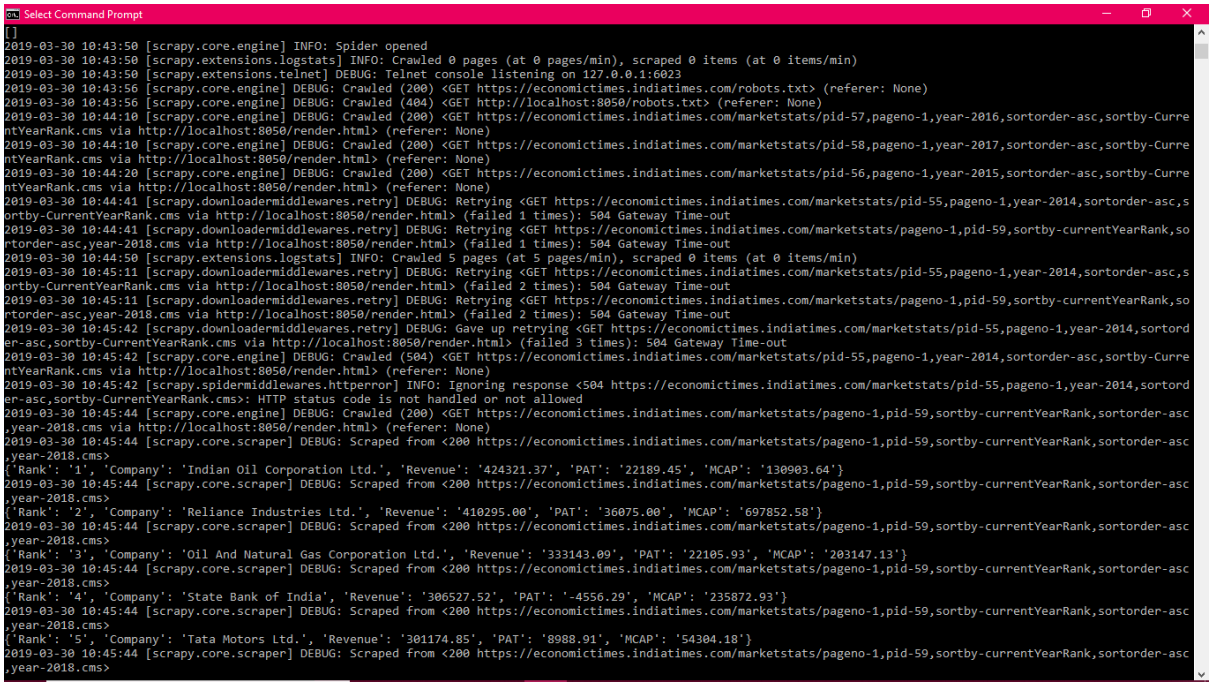

Figure A.3: Opening Spider

Figure A.4: Splash server forward

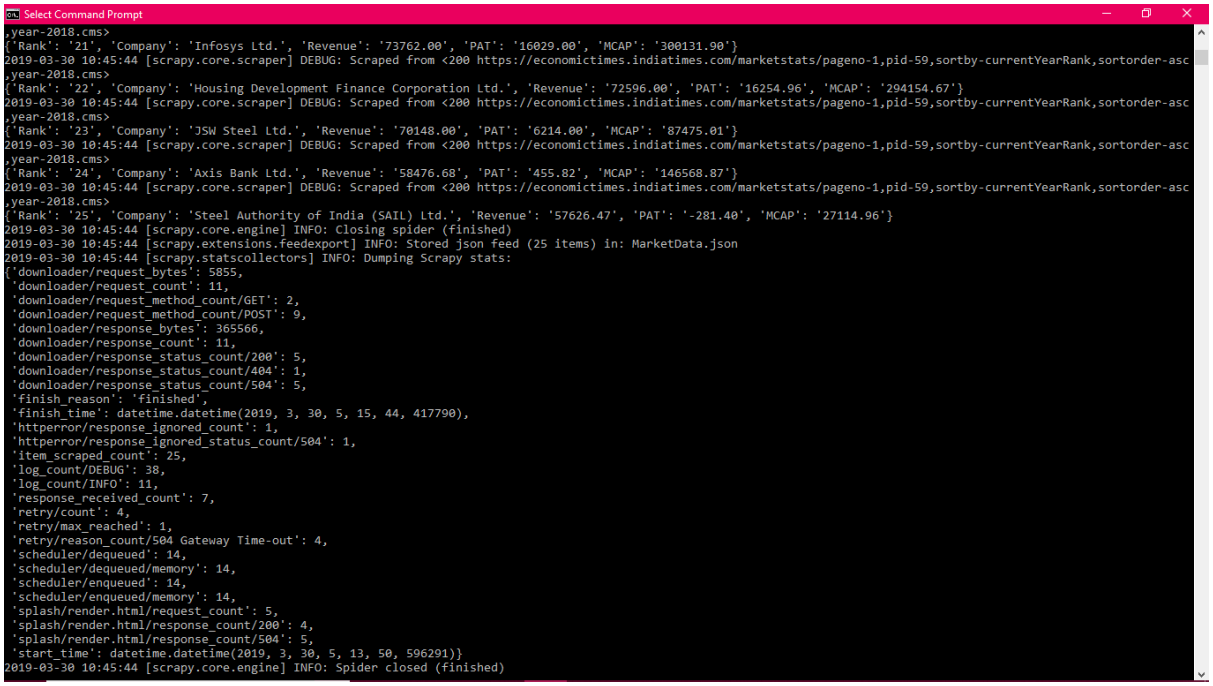

Figure A.5: Closing spider

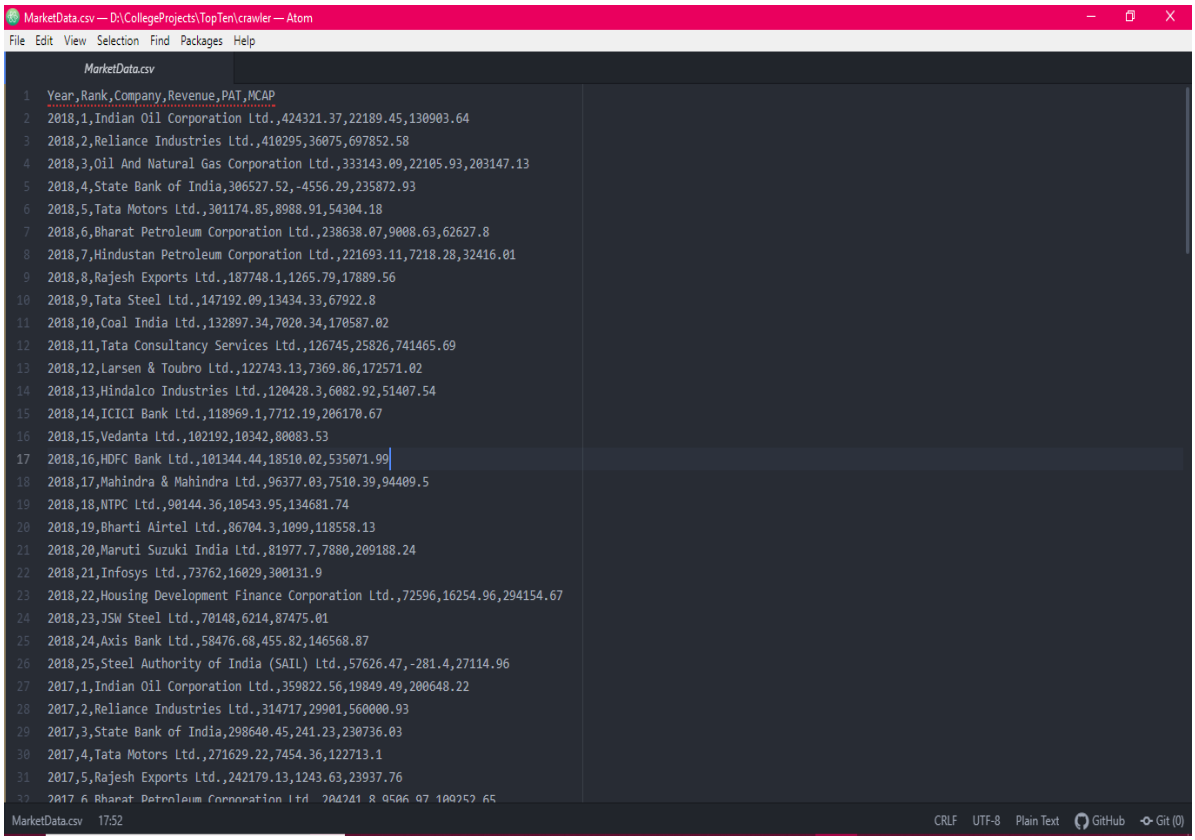

Figure A.6: Output

REFERENCES

[1] Anish Gupta, Priya Anand: Focused Web Crawlers and its Approaches. In: 2015 1st International Conference on Futuristic trend in Computational Analysis and Knowledge Management (ABLAZE-2015)

[2] ZHENG Guojun, JIA Wenchao, SHI Jihui, SHI Fan, ZHU Hao, LIU Jiang : Design and Application of Intelligent Dynamic Crawler for Web Data Mining. In: IEEE Transactions on Industrial Informatics, 2014

[3] Shruti Sharma, Parul Gupta: The Anatomy of Web Crawlers. In: International Conference on Computing, Communication and Automation (ICCCA2015)

[4] Gunjan H. Agre, Nikita V Mahajan: Keyword Focused Web Crawler. In: IEEE Sponsored 2'nd International Conference On Electronics And Communication Systems (ICECS '2015)

[5] Bing Zhou, Bo Xiao, Zhiqing Lin, Chuang Zhang: A Distributed Vertical Crawler Using Crawling-Period Based Strategy. In: Proceedings of the 2009 International Conference on Network Infrastructure and Digital Content.