# FLIPKART SCRAPER

A project report submitted in partial fulfillment of the requirements

for the award of credits to

Python a Skill Enhancement Course of

**Bachelor of Technology**

In

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)**

By

**KASA HARENDRA**

**23BQ1A4274**


**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)**

**VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY**

**(Approved by AICTE and permanently affiliated to JNTUK)**

**Accredited by NBA and NAAC with 'A' Grade**

**NAMBUR (V), PEDAKAKANI (M), GUNTUR-522 508**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM)**

**VASIREDDY VENKATADRI INSTITUTE OF TECHNOLOGY: NAMBUR**

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA**


# CERTIFICATE

This is to certify that the project titled "**FLIPKART SCRAPER**" is a bonafide record of work done by **Mr. KASA HARENDRA** under the guidance of **Mr. M. Pardha Saradhi, Associate Professor** in partial fulfillment of the requirement for the award of credits to **Python -** a skill enhancement course of Bachelor of Technology in Computer Science & Engineering – Artificial Intelligence & Machine Learning (CSM), JNTUK during the academic year 2024-25.

| | |
|---|---|
| **M. Pardha Saradhi** | **Prof. K. Suresh Babu** |
| **Course Instructor** | **Head of the Department** |

# DECLARATION

I, **Kasa Harendra(23BQ1A4274),** hereby declare that the Project Report entitled "**FLIPKART SCRAPER**" done by me under the guidance of **Mr.M. Pardha Saradhi, Associate Professor** is submitted in partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE & ENGINEERING – ARTIFICIAL INTELLIGENCE & MACHINE LEARNING (CSM).**

DATE :

PLACE :

**SIGNATURE OF THE CANDIDATE**

**KASA HARENDRA**

# ACKNOWLEDGEMENT

We express our sincere thanks to the Chairman, **Vasireddy Venkatadri Institute of Technology, Sri Vasireddy Vidya Sagar** for providing us well equipped infrastructure and environment.

We thank **Dr. Y. Mallikarjuna Reddy**, Principal, Vasireddy Venkatadri Institute of Technology, Nambur, for providing us the resources for carrying out the project.

We express our sincere thanks to **Dr. K. Giribabu**, Dean of Academics for providing support and stimulating environment for developing the project.

Our sincere thanks to **Dr. K. Suresh Babu**, Head of the Department, Department of CSM, for his co-operation and guidance which helped us to make our project successful and complete in all aspects.

We also express our sincere thanks and are grateful to our guide **Mr. M. Pardha Saradhi**, Associate Professor, Department of CSM, for motivating us to make our project successful and fully complete. We are grateful for his precious guidance and suggestions.

We also place our floral gratitude to all other teaching staff and lab technicians for their constant support and advice throughout the project.

**KASA HARENDRA**

23BQ1A4274

# Table of Contents

## LIST OF FIGURES



## LIST OF TABLES

# ABSTRACT

The project focuses on scraping product data and descriptions from Flipkart using the **Scrapy framework**, an open-source tool designed to extract data from websites by leveraging HTML tags. Scrapy employs spiders, specialized programs that crawl through webpages within the allowed domain, in this case, *flipkart.com*. These spiders navigate specific sections of the HTML structure, extract the required text and data, and yield it for further use. The scraped data is then stored in CSV files, enabling data analysis.

To begin, all necessary dependencies such as Scrapy and requests are installed. The project is initialized by creating a Scrapy project using the command scrapy startproject <project_name> in the command line or terminal. A spider, which handles the crawling and scraping tasks, is then generated. This spider is configured to crawl through specific locations on the website and collect the desired information.

The process involves obtaining a "search term" from the user. Based on this input, a URL is dynamically generated to locate the webpage containing products related to the search term. The spider traverses all products listed on the URL and, if needed, navigates to subsequent pages for comprehensive data collection.

The scraped data is structured in the form of a Python dictionary and stored in a CSV file. This file is saved on the user's local machine, guided by user instructions, ensuring convenient access for further analysis.

# Chapter 1

# INTRODUCTION

## 1.1 OVERVIEW OF E-COMMERCE DATA AND ITS IMPORTANCE

The rapid growth of e-commerce has transformed the way people shop, making data a central component in understanding and optimizing the digital retail landscape. E-commerce data encompasses a wide array of information about products, customer behavior, sales trends, and marketplace dynamics. Collecting and analyzing this data is crucial for businesses, as it provides insights into consumer preferences, pricing strategies, and product performance.

### *1.1.1. Types of E-commerce Data*

1. *Product Information:* Includes product titles, descriptions, specifications, prices, and availability. This data helps both sellers and buyers by providing detailed insights into product offerings.
2. *Customer Reviews and Ratings:* Feedback from customers reflects product quality and customer satisfaction. Analyzing this data aids in understanding public perception and identifying areas for improvement.
3. *Sales and Pricing Trends:* Monitoring changes in pricing and sales volume over time can reveal patterns and help businesses optimize pricing strategies.
4. *User Behavior Data:* Data on browsing habits, purchase history, and search patterns offers a window into customer preferences and behavior, enabling personalized marketing strategies.

### *1.1.2. Importance of E-commerce Data*

1. *Enhancing Customer Experience:* By analyzing e-commerce data, businesses can tailor their offerings to meet customer preferences. Recommendations, personalized ads, and optimized product pages are some examples of data-driven customer experience enhancements.
2. *Competitive Advantage:* Analyzing competitor data can help businesses position their products effectively, adjust pricing, and identify gaps in the

market. E-commerce data scraping provides insights that can be used to benchmark against competitors.

3. *Inventory and Supply Chain Management*: Product demand forecasts, based on historical sales and current trends, allow for efficient inventory management, reducing overstock and stockout situations.
4. *Data-Driven Marketing and Sales Strategies:* E-commerce data informs advertising strategies, helping businesses focus on high-conversion products, optimize ad spending, and target the right audience segments. Marketing campaigns driven by customer data are generally more effective and yield a higher ROI.
5. *Product Development and Improvement*: Customer reviews and feedback data reveal insights into product quality and user preferences, guiding product development and improvement efforts to align with customer needs.

### *1.1.3. Challenges and Considerations*

1. *Privacy and Ethical Concerns:* As businesses gather vast amounts of customer data, they must address privacy concerns and adhere to data protection laws to avoid misuse of personal information.
2. *Data Reliability and Quality:* The dynamic nature of e-commerce platforms requires constant data updates to ensure accuracy, as prices, availability, and product listings frequently change.
3. *Data Volume and Analysis Complexity:* E-commerce generates massive data volumes that require sophisticated processing and analysis tools. Managing and extracting insights from such large datasets requires advanced analytics techniques and resources.

In summary, e-commerce data is indispensable for driving informed decisions and shaping successful business strategies. Businesses that effectively harness e-commerce data are better positioned to understand the market, meet customer needs, and maintain a competitive edge in the rapidly evolving digital economy.

## 1.2 IMPORTANCE OF DATA SCRAPING IN MARKET RESEARCH AND ANALYSIS

Data scraping has become an essential tool in market research and analysis, enabling businesses to gather large volumes of information from various online sources efficiently. This practice allows companies to gain insights into competitor strategies, customer preferences, and market trends, which are crucial for informed decision-making and strategic planning.

### *1.2.1 Competitive Analysis*

1. *Pricing Strategies:* By scraping product prices from competitors' websites, businesses can benchmark their prices, identify pricing trends, and adjust accordingly to stay competitive. Frequent updates in e-commerce prices make automated scraping especially valuable for real-time adjustments.
2. *Product Offerings and Features:* Data scraping reveals details about competitors' product features, specifications, and reviews. This information helps companies identify product gaps, plan product improvements, and discover emerging features that are popular in the market.

### *1.2.2. Customer Sentiment Analysis*

1. *Reviews and Ratings:* Scraping customer reviews and ratings allows businesses to gauge customer satisfaction and sentiment toward specific products or brands. Through sentiment analysis of this data, companies can identify common issues, strengths, and areas for improvement.
2. *Understanding Trends in Consumer Preferences*: Analyzing the frequency and nature of customer comments highlights trends in consumer preferences. These insights can inform decisions on new product development or updates to existing offerings based on what customers value or demand.

### *1.2.3. Market Trend Analysis*

1. *Tracking Changes in Demand and Popularity:* Scraping e-commerce platforms or social media sites for trending products, keywords, or hashtags helps businesses stay on top of demand shifts. Identifying trending products

early enables a company to capitalize on new opportunities faster than competitors.

2. *Regional Preferences:* By segmenting data by region, companies can understand variations in product demand across different geographical markets. This allows for targeted marketing and customized inventory planning for specific regions.

### *1.2.4. Lead Generation and Target Audience Profiling*

1. *Prospecting and Customer Acquisition:* Data scraping can help identify potential customers, partners, or influencers. For example, scraping contact information or user profiles from public sources (in compliance with data privacy regulations) enables businesses to develop targeted outreach strategies.
2. *Behavioral Profiling and Audience Segmentation:* By scraping user engagement data, such as likes, shares, or comments on e-commerce platforms or social media, businesses can create detailed customer profiles. Understanding the audience's behavior and preferences allows for tailored marketing efforts that are more likely to convert.

### *1.2.5. Inventory and Supply Chain Optimization*

1. *Demand Forecasting:* Data scraping allows companies to monitor product availability, stock levels, and demand fluctuations in near real-time. This information can help optimize inventory levels, reduce overstock and stock out situations, and ensure a smoother supply chain.
2. *Supplier and Product Research:* For businesses that source products from multiple vendors, scraping can help compare suppliers based on product features, price, and quality, ensuring the best choices in the supply chain.

### *1.2.6. Advantages of Automated, Real-Time Data Collection*

1. *Efficiency and Scale:* Manually collecting large volumes of data is time-consuming and resource-intensive. Automated scraping enables faster and more frequent data collection, allowing businesses to analyze up-to-date information continuously.

2. *Data Accuracy and Consistency:* Automated scraping ensures consistency in data collection, minimizing the errors or biases that might arise from manual methods. It allows businesses to analyze data on a large scale, yielding more accurate insights.

In summary, data scraping is a powerful tool that fuels competitive analysis, consumer insight, trend forecasting, and strategic decision-making. For market research and analysis, data scraping allows businesses to leverage vast, real-time data sources, providing actionable insights that can drive growth and maintain a competitive edge.

## 1.3 OVERVIEW OF FLIPKART AS A PLATFORM AND WHY IT'S SELECTED

Flipkart is one of India's largest and most popular e-commerce platforms, offering a wide range of products across numerous categories, including electronics, fashion, home appliances, and groceries. Since its inception in 2007, Flipkart has grown into a digital marketplace that caters to millions of customers, providing a convenient and secure online shopping experience. The platform is known for its competitive pricing, frequent discounts, diverse product selection, and robust logistics network.

### 1.3.1. About Flipkart

1. *Product Diversity:* Flipkart hosts products across multiple categories, from major electronics like smartphones and laptops to essentials like clothing, home decor, and groceries. This diversity makes Flipkart a comprehensive source for studying product availability and pricing across various sectors.
2. *User Engagement:* With millions of active users, Flipkart's platform offers valuable insights into customer preferences, popular products, and shopping trends. The site features extensive customer reviews and ratings, allowing for an analysis of customer sentiment.
3. *Logistics and Service Network:* Flipkart has built a strong logistics and delivery network, reaching even remote regions of India. This makes it an influential player in the Indian e-commerce market, with its trends often reflecting broader consumer behavior.

### 1.3.2. Why Flipkart Is Selected for Data Scraping

1. *Market Representation*: As one of the largest e-commerce platforms in India, Flipkart's data is highly representative of the Indian online retail market. Data scraped from Flipkart can provide insights into consumer behavior, product pricing, and demand patterns across the country.
2. *Access to a Wide Range of Data Points*: Flipkart provides detailed information on products, including descriptions, specifications, prices, customer ratings, and reviews. This rich data source allows for comprehensive market analysis and competitor research.
3. *Competitive and Dynamic Pricing*: Flipkart frequently adjusts prices based on demand, availability, and seasonal discounts, making it an ideal platform for studying pricing strategies. Scraping this data enables business to monitor competitors and adapt their own pricing models.
4. *Customer Reviews and Feedback:* Flipkart's extensive user-generated reviews and ratings offer invaluable feedback on product performance and customer satisfaction. This data is crucial for sentiment analysis, identifying common customer pain points, and assessing product quality.
5. *Relevance for Market Trends:* As a leader in the Indian e-commerce space, Flipkart's sales patterns and product trends are often indicative of broader market shifts. By scraping Flipkart data, companies can stay updated on the latest market trends and consumer preferences, giving them a competitive edge.

### 1.3.3. Benefits of Using Flipkart Data for Analysis

1. *Competitor Benchmarking*: By analyzing similar products across different sellers on Flipkart, companies can benchmark their products against competitors. This can help in refining pricing strategies and identifying product gaps.
2. *Product Demand and Inventory Management*: Flipkart data provides insights into product demand, helping businesses anticipate high-demand items and manage their inventories more effectively.
3. *Enhanced Marketing Strategies*: Understanding customer reviews and ratings on Flipkart helps businesses understand what customers like and

dislike. This feedback can shape marketing campaigns, product descriptions, and customer engagement strategies.

4. *Data Accessibility*: Flipkart's organized categories and detailed product pages make data relatively accessible for automated scraping. Each product page is structured to display key information clearly, which facilitates efficient data extraction.

In summary, Flipkart's extensive product range, active user base, and prominence in the Indian e-commerce market make it a rich data source for studying pricing strategies, consumer behaviour, and product trends. By scraping Flipkart data, businesses can gather meaningful insights that are highly relevant to the Indian market, aiding in informed decision-making and competitive positioning.

## 1.4 AIM OF PROJECT:

My project aims to provide a user-friendly desktop application for scraping product data from Flipkart and visualizing key insights. It combines web scraping capabilities with an intuitive graphical user interface (GUI) built using PyQt5, making it accessible to users without extensive programming knowledge.

### *1.4.1 Inputs*

1. *File Name:* The user provides a desired file name (without extension) to save the scraped data. This input is taken in the ScraperGUI window.
2. *Search Term:* The user enters a search term (e.g., "shoes," "smartphone") that defines the type of products they want to scrape from Flipkart. This input is also taken in the ScraperGUI window.
3. *Product Count:* The user specifies the number of products they want to scrape for the given search term. This input determines how many product listings are fetched and processed.

### *1.4.2 Outputs*

1. *CSV File:* The primary output is a CSV file containing the scraped product data. The file name is based on the user-provided input, and it includes

information such as product name, price, rating, and other relevant details extracted from Flipkart.

2. *Data Visualizations:* The application generates two types of visualizations:
    - Sales Visualization: A line plot displaying the maximum and minimum sale prices for each brand (Company_name) found in the scraped data.
    - Rating Visualization: A bar chart showing the average rating for each brand.
3. *User Feedback:* Throughout the process, the application provides feedback messages to the user, including:
    - Confirmation of scraping completion.
    - Notifications about saved file names.
    - Error messages in case of issues during scraping or file operations.
4. *Optional Data Deletion:* Before closing the Window, the user is prompted to choose whether to save or delete the generated CSV file, giving them control over data persistence.

### *1.4.3 Possible Steps involved:*

1. *Data Acquisition:*

    1. Targeted Scraping: The application allows users to specify a search term and the number of products they want to scrape from Flipkart. This targeted approach ensures users obtain relevant data without being overwhelmed by irrelevant results.
    2. Scrapy Integration: Extracting data from the website by making use of Scrapy framework, a powerful and efficient web scraping tool.
    3. Structured Data Storage: Scraped data is organized and stored in a CSV (Comma Separated Values) file, a widely compatible format that can be easily opened and analyzed in various applications.

2. *User-Friendly Interface:*

1. Intuitive Design: The GUI, built with PyQt5, presents a clean and straightforward interface. Users can easily input their desired search term, file name, and product count.
2. Clear Feedback: The application provides informative messages throughout the process, indicating scraping progress, completion status, and potential errors.
3. Data Visualization Options: Once scraping is complete, a dedicated window offers buttons to visualize sales data (maximum and minimum prices) and product ratings through interactive plots. This visual representation helps users quickly grasp trends and patterns within the scraped data.

3. *Data Management:*

1. File Saving and Opening: Users have the flexibility to specify a file name for saving the scraped data. Additionally, they can directly open the CSV file from the application using their system's default program.
2. Optional Data Deletion: Upon closing the data visualization window, the application prompts users whether they want to save the CSV file or discard it, providing control over data retention.
3. Overall, my project successfully combines web scraping, data processing, and visualization into a user-friendly application. It empowers users to gather valuable product information from Flipkart, analyse pricing trends, assess product ratings, and gain insights for informed decision-making, whether for personal research, competitive analysis, or market research purposes.

## 1.5 SCOPE OF PROJECT:

Data scraping has become an essential tool in market research and analysis, enabling businesses to gather large volumes of information from various online sources efficiently. This practice allows companies to gain insights into competitor strategies, customer preferences, and market trends, which are crucial for informed decision-making and strategic planning.

### *1.5.1. Competitive Analysis*

1. *Pricing Strategies:* By scraping product prices from competitors' websites, businesses can benchmark their prices, identify pricing trends, and adjust accordingly to stay competitive. Frequent updates in e-commerce prices make automated scraping especially valuable for real-time adjustments.
2. *Product Offerings and Features:* Data scraping reveals details about competitors' product features, specifications, and reviews. This information helps companies identify product gaps, plan product improvements, and discover emerging features that are popular in the market.

### *1.5.2. Customer Sentiment Analysis*

1. *Reviews and Ratings:* Scraping customer reviews and ratings allows businesses to gauge customer satisfaction and sentiment toward specific products or brands. Through sentiment analysis of this data, companies can identify common issues, strengths, and areas for improvement.
2. *Understanding Trends in Consumer Preferences*: Analyzing the frequency and nature of customer comments highlights trends in consumer preferences. These insights can inform decisions on new product development or updates to existing offerings based on what customers value or demand.

### *1.5.3. Market Trend Analysis*

1. *Tracking Changes in Demand and Popularity:* Scraping e-commerce platforms or social media sites for trending products, keywords, or hashtags helps businesses stay on top of demand shifts. Identifying trending products early enables a company to capitalize on new opportunities faster than competitors.
2. *Regional Preferences:* By segmenting data by region, companies can understand variations in product demand across different geographical markets. This allows for targeted marketing and customized inventory planning for specific regions.

### *1.5.4. Lead Generation and Target Audience Profiling*

1. *Prospecting and Customer Acquisition:* Data scraping can help identify potential customers, partners, or influencers. For example, scraping contact information or user profiles from public sources (in compliance with data privacy regulations) enables businesses to develop targeted outreach strategies.
2. *Behavioral Profiling and Audience Segmentation:* By scraping user engagement data, such as likes, shares, or comments on e-commerce platforms or social media, businesses can create detailed customer profiles. Understanding the audience's behavior and preferences allows for tailored marketing efforts that are more likely to convert.

### *1.5.5. Inventory and Supply Chain Optimization*

1. *Demand Forecasting:* Data scraping allows companies to monitor product availability, stock levels, and demand fluctuations in near real-time. This information can help optimize inventory levels, reduce overstock and stock out situations, and ensure a smoother supply chain.
2. *Supplier and Product Research:* For businesses that source products from multiple vendors, scraping can help compare suppliers based on product features, price, and quality, ensuring the best choices in the supply chain.

### *1.5.6. Advantages of Automated, Real-Time Data Collection*

1. *Efficiency and Scale:* Manually collecting large volumes of data is time-consuming and resource-intensive. Automated scraping enables faster and more frequent data collection, allowing businesses to analyze up-to-date information continuously.
2. *Data Accuracy and Consistency:* Automated scraping ensures consistency in data collection, minimizing the errors or biases that might arise from manual methods. It allows businesses to analyze data on a large scale, yielding more accurate insights.

### *1.5.7. Personal Shopping and Decision-Making:*

1. *Price Comparison:* Consumers can use the tool to compare prices of products across different sellers on Flipkart, ensuring they get the best deals.
2. *Product Research:* Before making a purchase, users can gather information on product features, ratings, and price history to make informed decisions.
3. *Tracking Price Drops:* By regularly scraping data, users can monitor price fluctuations for desired products and make purchases when prices are favorable.

### *1.5.8. Academic Research and Data Analysis Projects:*

1. *E-commerce Data Collection:* Researchers can utilize the scraper to collect data on product listings, pricing, and ratings for academic studies related to e-commerce, consumer behavior, or market analysis.
2. *Sentiment Analysis:* The scraped data, including product reviews (if implemented in the future), can be used for sentiment analysis to understand customer opinions and perceptions towards specific products or brands.
3. *Machine Learning Applications:* The structured data collected through scraping can serve as training data for machine learning models aimed at predicting product demand, price trends, or customer preferences.

In summary, data scraping is a powerful tool that fuels competitive analysis, consumer insight, trend forecasting, and strategic decision-making. For market research and analysis, data scraping allows businesses to leverage vast, real-time data sources, providing actionable insights that can drive growth and maintain a competitive edge.

# Chapter 2

# LITERATURE REVIEW

## 2.1 MODULES USED:

Table 1 Modules used in the project

| Module | Functionality | Used For |
|---|---|---|
| sys | System-specific parameters and functions | Accessing command-line arguments (sys.argv), controlling script execution (sys.exit()) |
| os | Operating system interface | Executing system commands (os.system()), clearing the console (os.system('cls')) |
| matplotlib.pyplot | Plotting library | Creating visualizations (line plots, bar charts) |
| pandas | Data analysis and manipulation library | Reading and manipulating data from CSV files, grouping and aggregating data |
| numpy | Numerical computing library | Working with arrays, generating numerical ranges for plot axes |
| PyQt5.QtCore | Core non-GUI functionality of PyQt5 | Providing signals and slots for communication between objects (pyqtSignal) |
| PyQt5.QtGui | GUI elements and related classes | Defining icons (QtIcon), handling graphical aspects of the application |
| PyQt5.QtWidgets | Core GUI elements (widgets) | Creating the application window (QWidget, QApplication), layouts (QVBoxLayout, QHBoxLayout), input fields (QLineEdit), buttons (QPushButton), labels (QLabel), message boxes (QMessageBox) |
| subprocess | Managing subprocesses | Running external commands (e.g., executing the Scrapy crawler) |

## 2.2 FRAMEWORK USED FOR SCRAPING:

***Scrapy*** is a powerful, open-source web scraping framework written in Python, widely used for extracting data from websites in a structured, automated way. Initially released in 2008, Scrapy is maintained by an active community and is designed to be flexible, fast, and scalable, making it suitable for complex scraping projects that require significant customization.

### *2.2.1. Key Features and Functionality*

Scrapy's architecture is built around "spiders," which are classes written by the user to define the logic for crawling a website and extracting data. Each spider can be programmed to follow links, parse HTML, and capture specific data points, like product names, prices, or reviews. Scrapy uses asynchronous network requests, allowing it to efficiently crawl multiple pages concurrently, which makes it faster than synchronous alternatives, especially for large-scale scraping projects.

### *2.2.2. Components of Scrapy*

The main components of Scrapy include spiders, selectors, and items:

1. *Spiders:* These are the core of Scrapy, controlling what data to extract and where to navigate on a site. Each spider is defined by rules on how to follow links and parse information on the target site.
2. *Selectors:* Scrapy's selectors, based on XPath and CSS expressions, help locate specific HTML elements within a page's structure. This makes it easier to isolate the data of interest.
3. *Items:* Items are containers used to store extracted data before it's saved to a desired format like JSON, CSV, or a database.

### *2.2.3 Workflow*

The typical workflow in Scrapy involves creating a spider that sends requests to a target website. Each request triggers a response, which is processed to extract data according to the spider's rules. The extracted data is temporarily stored in "items," then passed to a pipeline where it undergoes any necessary post-processing, like cleaning or validation, before being stored in the final output format.

### *2.2.4 Advantages of Scrapy*

1. *Efficiency*: Scrapy's asynchronous architecture allows it to handle multiple requests simultaneously, reducing the time required for scraping large websites.
2. *Built-in Data Pipelines:* These enable data to be cleaned, processed, and stored seamlessly after extraction, supporting output to various formats.
3. *Middleware:* Scrapy's middleware allows for customization, such as rotating proxies, handling redirects, and managing user agents, making it possible to bypass anti-scraping mechanisms.
4. *Extensibility*: The modular structure of Scrapy makes it easy to add new features or integrate it with other tools, like machine learning models for data analysis.

### *2.2.5 Scrapy commands*:

Here's a list of commonly used Scrapy commands, which are executed from the command line within a Scrapy project. These commands allow you to create projects, generate spiders, run crawls, and manage scraped data.

1. *Start a New Project*

```
scrapy startproject <project_name>
```

   - Creates a new Scrapy project with the specified name, generating the initial folder structure and files.

2. *Generate a Spider*

```
scrapy genspider <spider_name> <domain>
```

   - Generates a new spider with a specified name, setting up a basic template for scraping the given domain.

3. *Run a Spider*

```
scrapy crawl <spider_name>
```

   - Runs the specified spider to start scraping data according to the rules defined within the spider.

4. *List Available Spiders*

```
scrapy list
```

   - Displays a list of all spiders available in the project.

5. *Shell Command*

```
scrapy shell <url>
```

   - Opens the Scrapy shell to interactively test selectors, parse responses, and extract data from a specified URL.

6. *View Spider Output in Real-Time*

```
scrapy view <url>
```

   - Renders the HTML content of a given URL in the default web browser. This can be useful for checking page structure and content.

7. *Export Data to File*

```
scrapy crawl <spider_name> -o <filename>.<format>
```

   - Runs a spider and saves the output data directly to a specified file format (e.g., JSON, CSV, XML):

     ```
     scrapy crawl my_spider -o output.json
     ```

8. *Check Spider Logs*

```
scrapy logfile <filename.log>
```

   - Outputs the log data for a spider's run to a specified file, making it easier to analyze and debug.

9. *Run with Settings Override*

```
scrapy crawl <spider_name> -s <SETTING_NAME>=<value>
```

10. *Fetch a URL*

```
scrapy fetch <url>
```

- Downloads the HTML of a URL and displays it in the command line. This is helpful for debugging responses.

11. *Check Project Benchmarks*

```
scrapy bench
```

- Runs a benchmarking test for the project, giving insights into Scrapy's performance for your setup.

12. *Run with a Specific Job ID*

```
scrapy crawl <spider_name> -s JOBDIR=crawls/<job_name>
```

- Runs the spider with a specific job ID to enable pausing and resuming of crawls.

13. *Clear Project Cache*

```
scrapy clean
```

- Cleans up cache files generated by the job if caching is enabled.

- *Help Command*

```
scrapy -h
```

- Displays a list of all Scrapy commands and options.

```
Scrapy 1.6.0 - no active project

Usage:
  scrapy <command> [options] [args]

Available commands:
  bench         Run quick benchmark test
  fetch         Fetch a URL using the Scrapy downloader
  genspider     Generate new spider using pre-defined templates
  runspider     Run a self-contained spider (without creating a project)
  settings      Get settings values
  shell         Interactive scraping console
  startproject  Create new project
  version       Print Scrapy version
  view          Open URL in browser, as seen by Scrapy

  [ more ]      More commands available when run from project directory

Use "scrapy <command> -h" to see more info about a command
```

*Fig 1: Scrapy Commands*

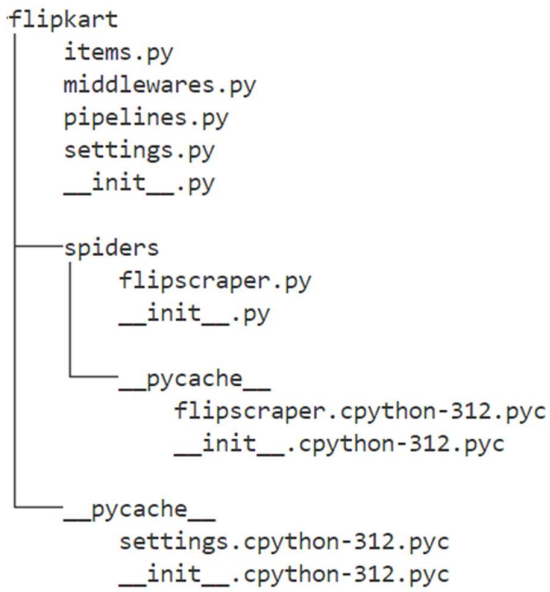


*Fig 2 Heirarchy tree of scrapy project*

## 2.3 FRAMEWORK USED FOR DESIGNING GUI:

PyQt is a set of Python bindings for the Qt application framework, which is a comprehensive toolkit for creating cross-platform applications with graphical user interfaces (GUIs). Developed by Riverbank Computing, PyQt allows developers to use the Qt library with Python, providing all the tools necessary for building sophisticated desktop applications for Windows, macOS, and Linux. With PyQt, developers can create interfaces ranging from simple forms to highly complex, feature-rich applications.

### *2.3.1 Key Features of PyQt*

PyQt includes modules and tools for managing every aspect of a GUI, such as widgets, layouts, dialogs, event handling, and advanced graphics. Some of its notable features include:

1. *Rich Widget Set*: PyQt offers a wide variety of built-in widgets, like buttons, labels, text fields, and sliders, as well as complex components like tables, trees, and multimedia viewers. This extensive library allows developers to

create detailed and interactive interfaces without building components from scratch.

2. *Layouts and Customization:* PyQt provides flexible layout management, allowing for complex arrangements of widgets within windows. Developers can organize content using vertical, horizontal, or grid layouts, ensuring that the interface remains responsive and user-friendly.
3. *Event Handling:* PyQt's event-driven architecture enables precise control over user interactions. Events, such as clicks, key presses, or mouse movements, trigger functions or changes in the application's state, allowing for dynamic, responsive applications.
4. *Cross-Platform Compatibility:* PyQt applications run on multiple operating systems without code modification, making it an excellent choice for projects that need to support diverse platforms.
5. *Graphics and Multimedia:* PyQt supports 2D and 3D graphics, animations, and multimedia content, making it suitable for applications that require rich visual elements, like games or design software.

### *2.3.2 PyQt Structure and Components*

PyQt is composed of several modules, each offering functionality for different areas of GUI development:

1. *QtWidgets:* Contains the core components for creating windows, dialogs, and interactive elements.
2. *QtCore:* Provides essential non-GUI functionalities like file I/O, threading, and data handling.
3. *QtGui:* Includes tools for drawing, handling images, and working with fonts and colors.
4. *QtMultimedia* and *QtMultimediaWidgets:* These modules allow developers to add multimedia features, like audio and video playback, to applications.
5. *QtWebEngine:* Integrates web content, enabling developers to embed web pages or use HTML for parts of the interface.

### *2.3.3. Building a PyQt Application*

Creating an application in PyQt typically follows these steps:

1. Setup**:** Import the necessary PyQt modules and define the main application instance.
2. Window and Layout**:** Design the main window, using widgets and layouts to arrange the interface elements.
3. Event Handling**:** Connect widgets to functions using PyQt's signal and slot mechanism, which allows specific actions (signals) to trigger functions (slots).
4. Execution**:** Launch the application with app.exec_(), which starts PyQt's event loop and keeps the window open until the user exits.

### *2.3.4. Advantages of Using PyQt*

1. Python-Friendly**:** PyQt integrates well with Python's syntax and libraries, making it accessible to Python developers without extensive GUI experience.
2. High-Level Abstraction**:** PyQt abstracts complex elements, such as window handling and event processing, allowing developers to focus on application logic rather than platform-specific details.
3. Designer Tool Support**:** PyQt works with Qt Designer, a visual tool that allows developers to design GUIs by dragging and dropping widgets, which can then be imported into PyQt code. This accelerates development by reducing the need for manual layout coding.
4. Extensibility**:** PyQt applications can be integrated with a variety of Python libraries for additional functionality, such as NumPy for data processing or Matplotlib for visualizations.

### *2.3.5. Use Cases and Applications*

PyQt is widely used in various fields for applications like:

1. Data Analysis and Visualization Tools**:** Scientific and financial applications use PyQt for creating analysis tools that require extensive visualizations.
2. Desktop Applications: PyQt supports building robust, user-friendly applications like text editors, database managers, and image viewers.
3. Educational and Interactive Tools: Its flexibility and multimedia support make PyQt a popular choice for educational software, interactive simulations, and games.

## 2.4 MODULES USED FOR DATA ANALYSIS AND DATA VISUALIZATION:

### *2.4.1 Pandas:*

Pandas is an open-source data manipulation and analysis library for Python, widely used in data science, finance, economics, and machine learning. Built on top of the NumPy library, Pandas provides fast, flexible, and expressive data structures designed to make working with structured data easy and intuitive. It is particularly useful for handling data in the form of tables or data frames, making it a go-to tool for data wrangling and analysis.

***Key Features of Pandas***

1. *Data Structures:*
    - Series**:** A one-dimensional labeled array capable of holding any data type (integers, strings, floats, etc.). A Pandas Series is similar to a column in a spreadsheet and can hold indexed data.
    - DataFrame**:** A two-dimensional labeled data structure with columns of potentially different types, similar to a table in SQL or an Excel spreadsheet. DataFrames are the primary data structure in Pandas and offer extensive functionality for data analysis.
2. *Data Manipulation:*
    - Pandas provides powerful tools for reading, writing, merging, reshaping, filtering, and transforming data. Functions like merge(), concat(), and join() allow seamless integration of multiple data sources, while groupby() enables complex aggregations and analysis.
    - Reshaping data**:** With functions like pivot_table(), melt(), and stack(), users can reshape data into different formats for more accessible analysis.
3. *Data Cleaning:*
    - Pandas has robust functions for handling missing data, removing duplicates, converting data types, and applying transformations. This is crucial in real-world scenarios where data is often incomplete or inconsistent.

- Functions like fillna() and dropna() allow users to handle missing values, while replace() can substitute incorrect or placeholder values.

4. *Data Import and Export:*
    - Pandas supports various data formats, including CSV, Excel, SQL, JSON, and even HTML tables. The read_csv(), read_excel(), and to_csv() functions, for example, enable easy loading and saving of data, making Pandas versatile for handling diverse file types.
5. *Time Series Analysis:*
    - Pandas has strong support for working with time series data. It can parse dates, generate date ranges, and allow resampling and rolling calculations, which are essential for tasks in finance, climate science, and other fields that rely on temporal data.
6. *Data Aggregation:*
    - The groupby() function allows grouping data by one or more columns, making it easy to calculate aggregated statistics, such as sums, means, or counts, for each group. This is particularly useful for summarizing large datasets quickly.
7. *Data Visualization:*
    - Although not a dedicated visualization library, Pandas integrates with Matplotlib and has built-in plotting functions, enabling quick visualization of data through line charts, histograms, scatter plots, and more.

***Why Pandas is Essential***

- Ease of Use**:** Pandas' intuitive syntax and data-centric functions allow users to perform complex data operations with simple commands, speeding up the data analysis process.
- Flexibility**:** Pandas can handle different types of data, from numeric to text to time-series data, making it suitable for a wide range of data-centric tasks.
- Performance**:** Built on NumPy, Pandas is optimized for performance, handling large datasets efficiently. It supports vectorized operations, which are faster than iterative approaches.
- Integration with Other Libraries**:** Pandas integrates smoothly with other popular Python libraries, such as Matplotlib for visualization, SciPy for

scientific computation, and scikit-learn for machine learning. This makes it central in the Python data science ecosystem.

**matplotlib.pyplot** is a module within the Matplotlib library, widely used in Python for data visualization. It provides an interface for creating static, animated, and interactive plots, enabling users to transform raw data into visual insights. pyplot is modeled after MATLAB's plotting commands, making it intuitive and easy to use, especially for creating common chart types quickly.

***Key Features of matplotlib.pyplot***

1. *Versatile Plotting Options:*
    - pyplot supports a wide range of plots, such as line plots, scatter plots, bar charts, histograms, pie charts, and more. This versatility allows users to visualize data in ways that best represent the underlying patterns or distributions.
2. *High Customizability:*
    - Users can customize almost every aspect of a plot, including colors, line styles, markers, fonts, grid lines, and titles. This customization enables fine control over how information is presented, making it easy to create publication-quality visuals.
3. *Layered Plotting:*
    - With pyplot, multiple plot elements can be layered on top of one another within the same figure. For instance, a line plot and scatter plot can be combined in one visualization, allowing for comparisons and deeper insights.
4. *Subplot and Grid Support:*
    - pyplot includes methods to create multiple plots within the same figure using subplot() and subplots(). This is especially useful for comparative analysis, where multiple data series are shown side-by-side.
5. *Interactive Features:*
    - pyplot offers support for interactive elements like zooming, panning, and hover information, especially when used within Jupyter

Notebooks or IPython environments. This interaction helps in exploring data points more closely without additional configuration.

6. *Seamless Integration with Data Science Libraries:*
   - matplotlib.pyplot integrates smoothly with libraries like Pandas, NumPy, and Seaborn. Pandas data frames can be passed directly to pyplot functions, simplifying data visualization within data analysis workflows.

## 2.5 OTHER MODULES USED:

### 1. sys:

The **sys** module in Python provides access to system-specific parameters and functions. It allows interaction with the Python runtime environment, such as handling command-line arguments (sys.argv), controlling standard I/O streams (sys.stdin, sys.stdout), and managing the interpreter's behavior with functions like sys.exit() for exiting a program.

### 2. os:

The os module in Python provides a way to interact with the operating system. It enables access to file and directory operations, environment variables, and process management. Key features include:

1. *File and Directory Management:* Functions like os.mkdir(), os.rmdir(), os.remove(), and os.rename() allow creating, deleting, and renaming files and directories.
2. *Path Manipulation:* os.path functions (e.g., os.path.join(), os.path.exists(), os.path.abspath()) make it easy to work with file paths across different operating systems.
3. *Environment Variables:* os.environ provides access to environment variables, enabling programs to interact with the system's configuration.
4. *Process Management:* Functions like os.system() allow running system commands, while os.getpid() and os.getlogin() retrieve the process ID and current user, respectively.

The os module is essential for creating platform-independent Python programs that interact directly with the operating system.

## 2.6 METHODS FROM EACH MODULE

### 1. PyQt5 Module

- ***QtCore:***
  - pyqtSignal(): Used in DataWindow to create a custom signal (close_signal) that's emitted when the window is closed. This allows for custom actions before the window closes completely.
- ***QtWidgets:***
  - QApplication(sys.argv): Initializes the Qt application, which is essential for any PyQt5 GUI.
  - QWidget: The base class for all user interface objects in PyQt5. Both ScraperGUI and DataWindow inherit from it.
  - QLabel('text'): Creates a label widget to display text.
  - QLineEdit(): Creates a single-line text input field.
  - QPushButton('text'): Creates a clickable button.
  - QHBoxLayout(), QVBoxLayout(): Layout managers to arrange widgets horizontally and vertically, respectively.
  - QMessageBox.warning(), QMessageBox.information(), QMessageBox.critical(), QMessageBox.question(): Display dialog boxes for warnings, information, errors, and questions to the user.
- ***QtGui:***
  - QIcon('icon.png'): Loads an image file to be used as an icon.

### 2. subprocess Module

- subprocess.run(["command", "arg1", "arg2"], check=True): Executes an external command (in this case, your scraping script) with arguments. The check=True argument raises an exception if the command fails.

### 3. os Module

- os.system(f'cmd /c "start {file_name}.csv"'): Opens a file using the default system application.
- os.system(f'cmd /c del /Q {file_name}.csv'): Deletes a file.
- os.system('cmd /c cls'): Clears the console screen.

**4. matplotlib.pyplot Module (plt)**

- plt.figure(figsize=(width, height)): Creates a new figure for plotting.
- plt.plot(x, y, color='color', linestyle='style', label='label'): Plots a line graph.
- plt.bar(x, height, color='color'): Plots a bar chart.
- plt.grid(axis='axis', color='color', linestyle='style', linewidth=width, alpha=transparency): Adds grid lines to the plot.
- plt.xlabel('label'), plt.ylabel('label'): Sets labels for the x and y axes.
- plt.xticks(ticks, labels, rotation=angle, fontsize=size), plt.yticks(ticks, labels, fontsize=size): Sets the ticks and labels for the x and y axes.
- plt.legend(): Displays the legend on the plot.
- plt.show(): Shows the plot.

**5. pandas Module (pd)**

- pd.read_csv(filename): Reads data from a CSV file into a pandas DataFrame.

**6. numpy Module (np)**

- np.arange(start, stop, step): Creates an array of evenly spaced values within a given range.

# Chapter 3
# PROJECT

## 3.1 PROJECT STRUCTURE:

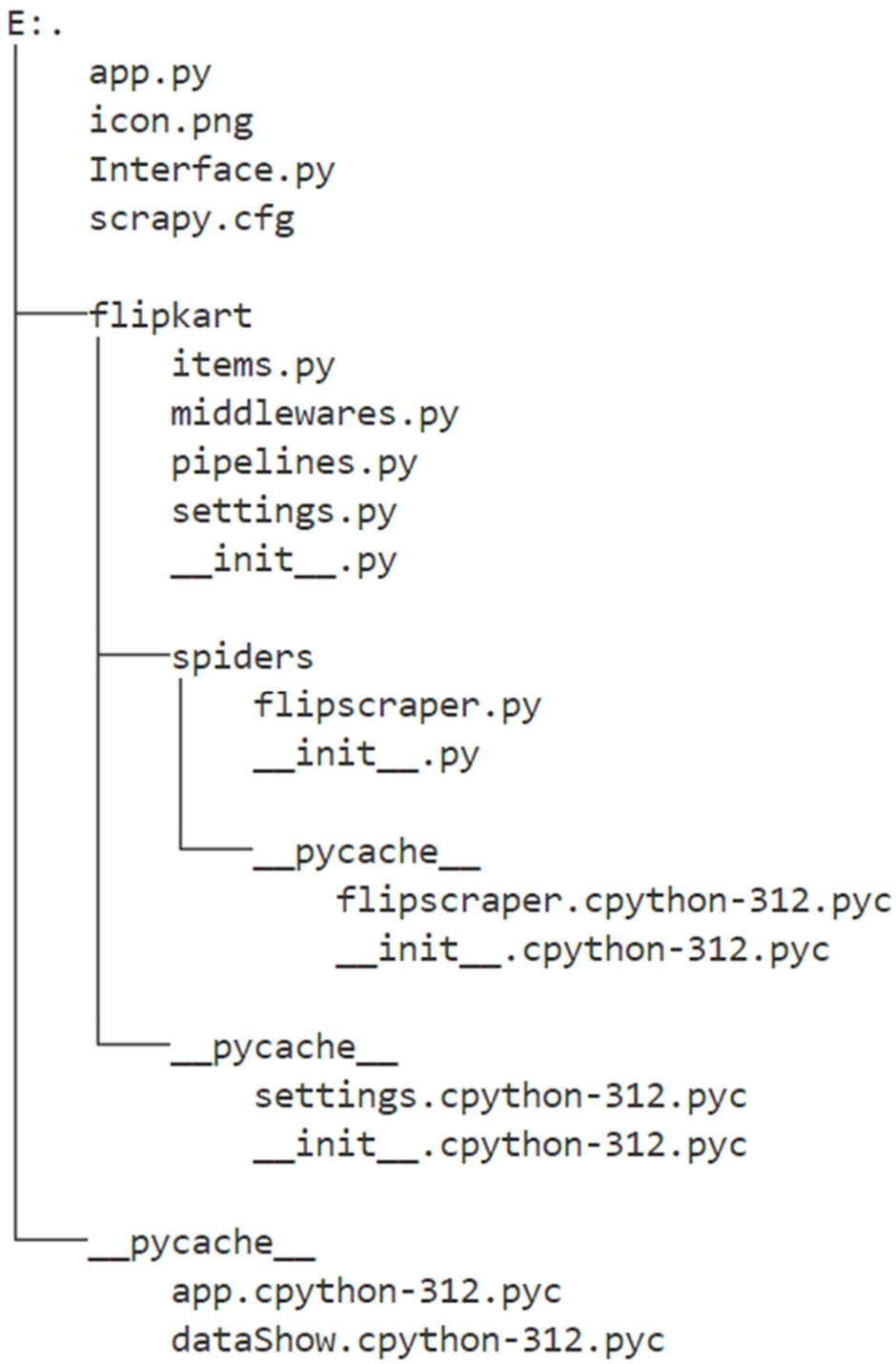


*Fig 3: The Whole Project Structure*

These files represent a standard Scrapy project setup for web scraping.

- **flipscraper.py**

  The Python script flipscraper.py is a Scrapy spider designed to extract product information from Flipkart.

  The script's main purpose is to scrape data from Flipkart product pages based on a given search term. It extracts details such as product name, price, discount, rating, reviews, and URL. It handles pagination to scrape multiple pages of search results.

  The spider is typically run from the command line using scrapy crawl flipscraper -a search_term=<the_search_term> -a num=<number_of_items>. The scraped data is then outputted, usually to a file (e.g., CSV, JSON) as defined in the Scrapy project's settings.

- **middlewares.py**: This file defines custom middleware components for your Scrapy spider. Middleware acts as an intermediary between the spider and the Scrapy engine, allowing you to modify requests and responses. This particular file has two classes:
  - **FlipkartSpiderMiddleware**: This class provides hooks to process requests before they reach the spider and responses before they are processed. It can be used for things like adding headers, handling redirects, or logging. The provided code includes default implementations that don't modify anything.
  - **FlipkartDownloaderMiddleware**: This class provides hooks to process requests before they are sent by the downloader and responses received from the website. This can be used for handling cookies, proxies, retrying failed requests, and other downloader-related tasks. The provided code includes default implementations that don't modify anything.
- **pipelines.py**: This file defines pipelines for processing items scraped by your spider. Items are the data extracted from web pages. Pipelines are used for cleaning, validating, and storing the scraped data (e.g., saving to a database, writing to a file). The provided FlipkartPipeline class has a

placeholder process_item method that currently does nothing. You would add your data processing logic here.

- **items.py**: This file defines the structure of the items that your spider will extract. It uses Scrapy's Item class to define fields for the data you want to scrape. Currently, the FlipkartItem class is empty, meaning no specific data fields are defined. You'd add fields like product_name, price, description, etc., corresponding to the data you intend to extract from Flipkart.
- **settings.py**: This file contains the configuration settings for your Scrapy project. It controls various aspects of the scraping process, such as:
    - BOT_NAME: The name of the scraping bot.
    - SPIDER_MODULES and NEWSPIDER_MODULE: These settings tell Scrapy where to find our spiders.
    - ROBOTSTXT_OBEY: A setting to respect robots.txt rules on websites, crucial for ethical scraping.
    - Several commented-out settings relate to request throttling, concurrency, cookies, headers, middleware, pipelines, and caching. These offer fine-grained control over the scraping process and can be uncommented and customized as needed.

In essence, these files work together to form a complete web scraping pipeline: the spider requests pages, middleware processes requests and responses, the spider extracts data into items, pipelines process those items, and the settings file configures the entire process. The project is currently a basic template; you would need to add your specific scraping logic to the spider, define the data structure in items.py, and configure the pipelines and settings to perform a meaningful scrape of Flipkart.

***Files that provide Interface:***

- ***Interface.py (User Interface):***

  This file defines the graphical user interface (GUI) for the scraper using PyQt5.

    - **ScraperGUI Class:** This class creates the main window, including input fields for the filename, search term, and item count. The "Scrape"

button triggers the scraping process. Error handling and informational messages are displayed using QMessageBox.

- **DataWindow Class:** This class creates a window for data analysis after scraping is complete. It displays buttons to visualize sales data, ratings, and open the CSV file. It also handles the deletion of the CSV file upon closing if the user chooses not to save it.

- **app.py (Scraping and Data Visualization):**

  This file contains the core scraping logic and data visualization functions.

  - **DataShow Class:** This class handles data processing and visualization. It reads the scraped data from the CSV file and provides methods to generate plots for sales and ratings.
  - **__main__ Block:** This block executes when the script is run directly. It takes command-line arguments for the output filename, search term, and count, then uses os.system to call the scrapy command-line tool. This executes the Flipkart scraper (presumably defined in a separate flipscraper.py file, not provided here) to fetch data. Finally, it clears the console.

### 3.2 ALGORITHMS INCLUDED IN THE PROJECT:

1. **Web Scraping with Scrapy:** The project utilizes the Scrapy framework, which employs a breadth-first search (BFS) algorithm to crawl web pages. It starts with a seed URL (start_urls) and then follows links found on those pages to discover new URLs, continuing this process until a certain depth or criteria is met. Scrapy itself doesn't inherently use any specific sorting or ranking algorithms during the crawling process; it simply follows links as it encounters them, organized by BFS.
2. **Data Processing with Pandas:** The project class utilizes the Pandas library for data analysis. Pandas uses various algorithms for data manipulation, including:
   - **Sorting:** For operations like sorting dataframes by columns.
   - **Grouping and Aggregation:** For functions like groupby() and aggregation functions (e.g., mean, max, min), which typically involve hash-based algorithms for efficient grouping.

- **Data Cleaning:** Pandas employs algorithms for handling missing data and data type conversions.

3. **Data Visualization with Matplotlib:** Matplotlib uses algorithms for plotting and rendering visualizations:
    - **Line Plotting:** Algorithms for generating line plots based on data points and interpolation.
    - **Bar Charting:** Algorithms to create bar charts, handling bar positioning, width, and height based on provided data.

## How Scraping is done?

Consider scraping the data of shoes from the flipkart site, and the webpage is specified by the URL `https://www.flipkart.com/search?q=shoes&as=on&as-show=on&otracker=AS_Query_TrendingAutoSuggest_2_0_na_na_na&otracker1=AS_Query_TrendingAutoSuggest_2_0_na_na_na&as-pos=2&as-type=TRENDING&&requestId=8f0b3ee0-959c-4c33-872a-b620540cb6b5.`

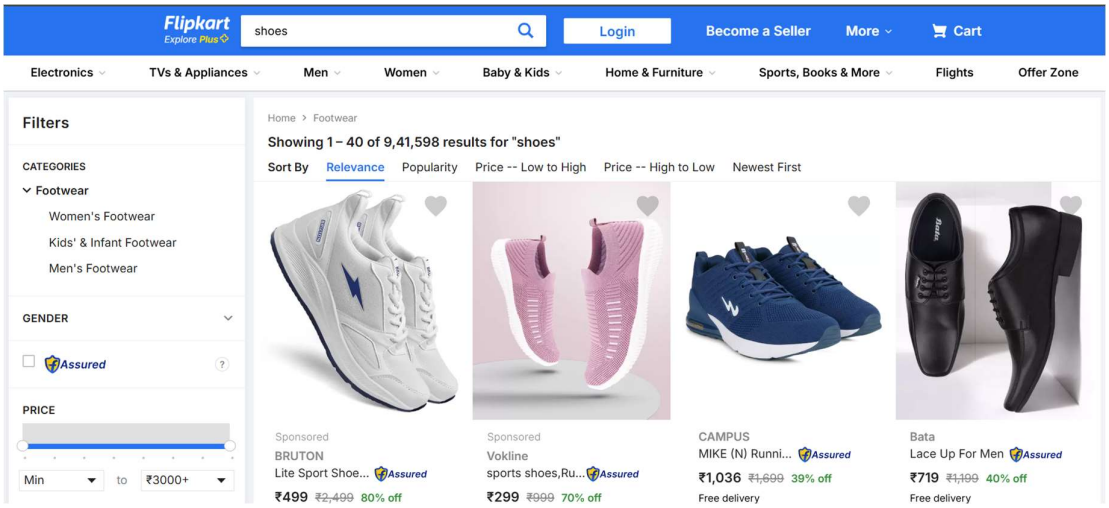


*Fig 4: FlipKart WebPage*

The background structure of the whole webpage is its HTML code. It consists of tags like div, img, p, etc… The information and text that appears on the webpage is that text which is enclosed between the HTML tags. Thus we obtain the text from the HTML of the webpage.

The HTML code can be obtained by doing inspect. And it appears as follows:

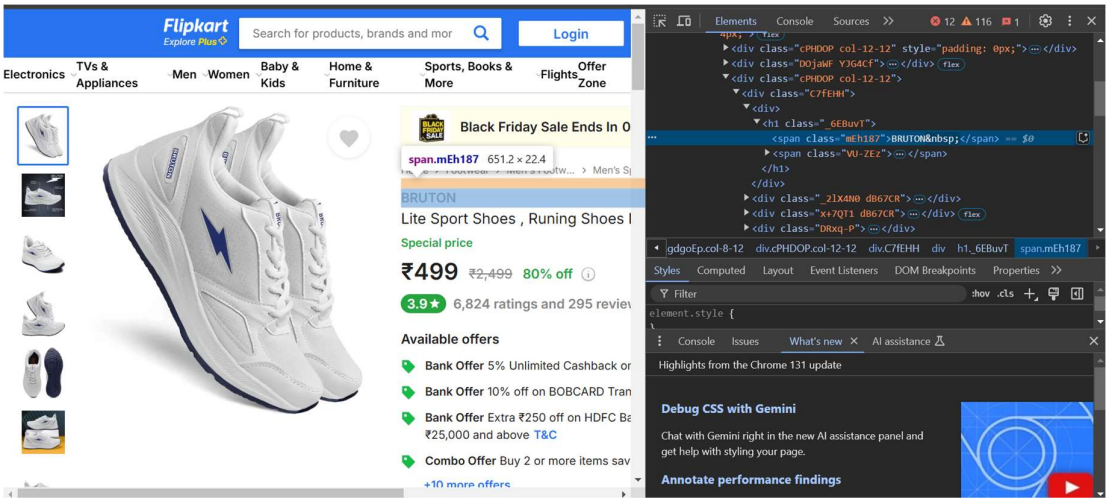


*Fig 5: HTML Structure of FlipKart Webpage*

Note how the name BRUTON is enclosed within the span tag. Thus by somehow navigating through the HTML code to the required part of code and obtaining the text within the tags is the primary job. This can be done using xpath() or css() methods.

Here in the project we made use of xpath() method as it proides more efficient way to navigate to tags which aren't belongs to specified class.

The code for obtaining that company name (enclosed between span tags in previous image)

```
item.xpath("//div/h1/span[@class='mEh187']/text()").get()
```

where "item" navigates to a div class as follow:

```
item = response.xpath("//div[@class='C7fEHH']")
```

Thus in total, the above two statements contribute in navigating to following part of HTML structure and obtaining the text:

```
<div class="C7fEHH">
      <div>
            <h1 class="_6EBuvT">
            <span class="mEh187">BRUTON </span>
            </h1>
      </div>
</div>
```

Similarly, the description is obtained in following way:

```
item.xpath("///div/h1/span[@class='VU-ZEz']/text()").extract())
```

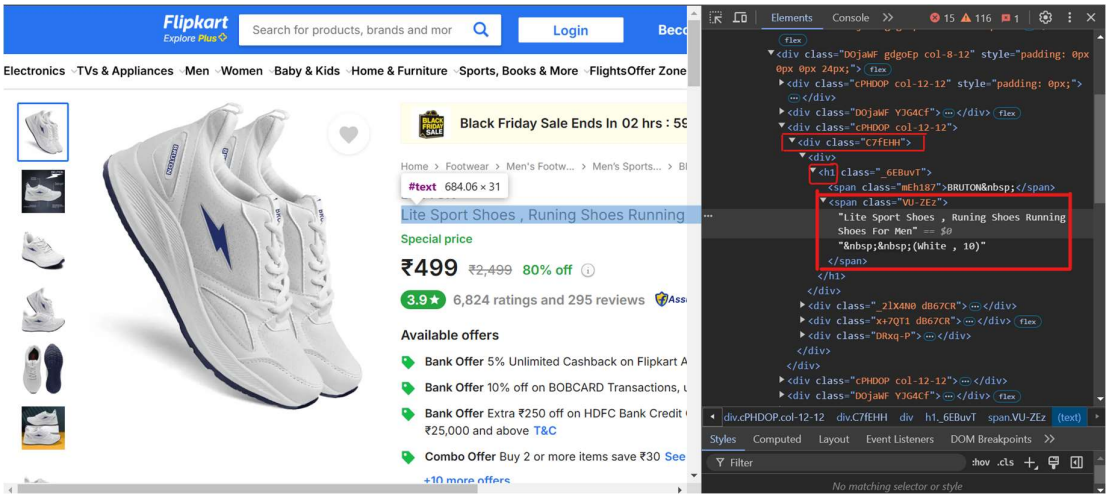


*Fig 6: Obtaining the text using xpath()*

## 3.3 PROJECT CODE:

### 3.3.1 Code that specifies the work of spider (flipscraper.py):

```
import scrapy
class FlipscraperSpider(scrapy.Spider):
    name = "flipscraper"
    #Setting the allowed domains to restrict our spider from crawling into
unnecessary domains
    allowed_domains = ["flipkart.com"]
    #Name of spider
    i = 0
    def __init__(self, search_term=None, num='all', *args, **kwargs):
        super(FlipscraperSpider, self).__init__(*args, **kwargs)
        self.search_term = search_term.replace(' ','+')
        if num!= 'all':
            self.num = int(num)
        else:
            self.num = num
        #The wepage from where the spider starts crawling
        self.start_urls =
[f"https://www.flipkart.com/search?q={self.search_term}&as=on&as-
show=on&otracker=AS_Query_TrendingAutoSuggest_2_0_na_na_na&otracker1=AS_Quer
y_TrendingAutoSuggest_2_0_na_na_na&as-pos=2&as-type=TRENDING&
&requestId=8f0b3ee0-959c-4c33-872a-b620540cb6b5"]
    #Method to generate a dictionary of details related to the item
    def parse_item(self, response):
        if response.status > 200:
            response = response.replace(status=201)
        link = response.url
        item = reponse.xpath("//div[@class='C7fEHH']")
```

```
        sale_price =
float((item.xpath("//div[@class='x+7QT1dB67CR']/div[@class='UOCQB1']/div/div
[@class='Nx9bqj CxhGGd']/text()").get()).strip('₹').replace(",",''))
        total =
item.xpath("//div[@class='x+7QTdB67CR']/div[@class='UOCQB1']/div/div[@class=
'yRaY8j A6+E6v']/text()").extract()
        if total is not None:
            total = float(total[-1].replace(",",''))
        rating =
item.xpath("//div[@class='DRxqP']/div/div/span[@class='Y1HWO0']/div/text()")
.get()
        if rating is not None:
            rating = float(rating)
        yield{
            'Company_name':
item.xpath("//div/h1/span[@class='mEh187']/text()").get().replace("\xa0","")
.upper(),
            'Descrption': (" ".join(item.xpath("///div/h1/span[@class='VU-
ZEz']/text()").extract())).replace("\xa0",''),
            'Sale Price (₹)': sale_price,
            'Total Price (₹)': total,
            'Discount': item.xpath("//div[@class='x+7QT1
dB67CR']/div[@class='UOCQB1']/div/div[@class='UkUFwK WW8yVX
dB67CR']/span/text()").get(),
            'Rating': item.xpath("//div[@class='DRxq-
P']/div/div/span[@class='Y1HWO0']/div/text()").get(),
            'Reviews & Ratings': item.xpath("//div[@class='DRxq-
P']/div/div/span[@class='Wphh3N']/span/text()").get(),
            'url': link
        }

    #Method to parse the webpage and follow the links to the next page
    def parse(self, response):
        if response.status > 200:
            response = response.replace(status=201)

        items = response.xpath("//div[@class='_1sdMkc
LFEi7Z']/a[@class='rPDeLR']/@href").extract()
        for item in items:
            if self.num != 'all':
                if self.i < int(self.num):
                    item_url = 'https://www.flipkart.com' + item
                    yield response.follow(item_url,
callback=self.parse_item)
                    self.i += 1
                else:
                    return
            else:
                item_url = 'https://www.flipkart.com' + item
                yield response.follow(item_url, callback=self.parse_item)
```

```
        next_page =
response.xpath("//div[@class='_1G0WLw']/nav[@class='WSL9JP']/a/@href").extra
ct()[-1]
        print(next_page)
        url = 'https://www.flipkart.com' + next_page
        yield response.follow(url, callback=self.parse)
```

The Python script flipscraper.py is a Scrapy spider designed to scrape product data from Flipkart.

***1. Class Definition and Initialization (FlipscraperSpider):***

- name = "flipscraper": This defines the spider's name, used to run it from the command line.
- allowed_domains = ["flipkart.com"]: Restricts the spider to only crawl within the flipkart.com domain, preventing accidental crawls on other websites.
- __init__(self, search_term=None, num='all', *args, **kwargs): The constructor initializes the spider with a search_term (the product to search for), and num (the number of items to scrape). The search_term is formatted for URL encoding by replacing spaces with '+'. num defaults to 'all,' indicating scraping all items. start_urls are initialized dynamically using the search query by formatting an f-string into the URL.

***2. parse_item(self, response):***

This method is called for each individual product page.

1. **HTTP Error Handling:** The first conditional checks for HTTP error status codes (greater than 200) and attempts to convert them to a 201 status. This is unusual error handling as 201 signifies "Created" rather than an error.
2. **XPath Extraction:** The core of this method is using XPath expressions to extract data from the HTML structure of the product page. It targets various elements like price, discount, rating, reviews, etc.
3. **Data Cleaning:** Some extracted data (price, total price) are cleaned by removing currency symbols (₹) and commas before conversion to float.
4. **Yielding Data:** The extracted data is organized into a dictionary and yielded as a Scrapy item. This dictionary contains key-value pairs representing the scraped information.

***3. parse(self, response):***

This method is the main entry point for the spider. It parses the search results page.

1. **HTTP Error Handling:** Similar to parse_item, it includes the unconventional error status modification.
2. **Extracting Product Links:** It uses XPath to find the URLs of individual product pages from the search results.
3. **Iteration and Callback:** It iterates through the extracted product links, and for each link, it calls response.follow(item_url, callback=self.parse_item). This schedules the parse_item method to be called with the response from the individual product page.
4. **Pagination:** The code extracts the link to the next page of search results and calls response.follow() with the parse method as the callback, creating a recursive crawling process to scrape multiple pages of results.

### 3.3.2 Code for visualiualization of data and running the spider from script (app.py):

```
import sys
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

class DataShow():
    def __init__(self, file_name):
        self.df = pd.read_csv(f'{file_name}.csv')
    def visualise_sale(self):
              df_sale_max   =   self.df.groupby('Company_name')['Sale   Price
(₹)'].max().to_dict()
              df_sale_min   =   self.df.groupby('Company_name')['Sale   Price
(₹)'].min().to_dict()
        plt.figure(figsize = (50,50))
         plt.plot(df_sale_max.keys(),df_sale_max.values(), color = 'r',label
= 'Maximum Sale Price')
               plt.plot(df_sale_min.keys(),df_sale_min.values(),    color   =
'g',linestyle = '--', alpha = 0.5, label = 'Minimum Sale Price')
         plt.grid(axis= 'y', color=  'b', linestyle= '-.', linewidth = 0.5,
alpha = 0.5)
        plt.yticks(fontsize = 10)
        plt.xticks(fontsize = 5, rotation=90)
        plt.xlabel('Company name')
        plt.ylabel('Price in Rs')
```

```
        plt.legend()
        plt.show()

    def visualise_rating(self):
        plt.ylabel('Rating')
        plt.xlabel('Company name')
                                                    df_rating            =
self.df.groupby('Company_name')['Rating'].mean().to_dict()
        plt.grid(axis = 'y', color = 'b', linestyle = '--', alpha = 0.5 )
        plt.bar(df_rating.keys(),df_rating.values(), color = 'g')
        plt.yticks(np.arange(0, 5.1, 0.5), fontsize = 10)
        plt.xticks(fontsize = 5, rotation=90)
        plt.show()

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python app.py <output_file> <search_term> <count>")
        sys.exit(1)

    name = sys.argv[1]
    search_term = sys.argv[2]
    count = sys.argv[3]

    os.system(f'cmd /c "scrapy crawl flipscraper -a search_term={search_term}
-a num={count} -O {name}.csv"')

    path = f'flipkart\{name}.csv'

    os.system('cmd /c cls')
```

app.py serves two primary functions: data visualization and acting as an entry point for running the Scrapy spider. Let's break down the code:

1. Data Visualization (DataShow Class):

1. __init__(self, file_name): Initializes the DataShow object by reading the CSV file specified by file_name into a Pandas DataFrame.
2. visualise_sale(self): Generates a line plot visualizing the maximum and minimum sale prices for each company. It uses groupby() to aggregate data by company name and extracts maximum and minimum sale prices. The plot includes labels, a legend, and gridlines.
3. visualise_rating(self): Creates a bar chart visualizing the average rating for each company. It uses groupby() to calculate average ratings and then plots the data using Matplotlib. The chart includes labels, gridlines, and custom y-tick spacing.

2. Scrapy Execution (Main Block):

- Command-line Argument Handling: Checks if the correct number of command-line arguments (output file name, search term, and count) is provided. If not, it prints usage instructions and exits.
- Scrapy Execution: This is the core function of the main block. It constructs a command string to execute the flipscraper Scrapy spider. The command includes arguments for the search term, count, and output filename. It uses os.system to execute this command in the system's shell.
- os.system('cls'): This command clears the console after the scraping process is completed. This is done to prevent Scrapy's output from cluttering the screen after execution finishes.

```
scrapy crawl flipscraper -a search_term={search_term} -a num={count} -O {name}.csv
```

- **scrapy crawl flipscraper**: This is the core command to run a Scrapy spider. scrapy is the command-line tool for the Scrapy framework, crawl is the subcommand to execute a spider, and flipscraper is the name of the spider defined in your flipscraper.py file.
- **-a search_term={search_term}**: This passes a custom argument named search_term to the spider. The value of {search_term} will be substituted with the actual search term provided by the user. This argument is then used by the flipscraper spider to construct the URL to scrape.
- **-a num={count}**: This passes another custom argument named num to the spider. The value of {count} is replaced with the number of items the user wants to scrape. The spider can use this argument to limit the number of pages or products it processes.
- **-O {name}.csv**: This directs Scrapy to output the scraped data to a CSV file. The -O flag overwrites the file if it already exists. {name} is replaced with the filename provided by the user, and .csv specifies the file format.

Example:

Let's say the user provides the following inputs:

- name: "mobiles"

- search_term: "samsung phones"
- count: "50"

The command would become:

```
scrapy crawl flipscraper -a search_term=samsung+phones -a num=50 -O mobiles.csv
```

This would inovke the flipscraper spider to crawl, which would search for "samsung phones" on Flipkart, scrape data for up to 50 items, and save the output to a file named mobiles.csv. The spaces in the search term are typically converted to '+' characters for URL encoding in the actual execution.

### 3.3.3 Code for Graphical User Interface (Interface.py):

```
import sys
import os
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
import subprocess  # For running external scripts
from app import DataShow

class ScraperGUI(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('FlipKart Scraper')
        self.setWindowIcon(QIcon('icon.png'))
        self.setFixedSize(800, 200)
        self.setGeometry(500, 100, 800, 200)
        self.initUI()

    def initUI(self):
        layout1 = QHBoxLayout()
        layout2 = QHBoxLayout()
        layout3 = QHBoxLayout()
        vLayout = QVBoxLayout()

        # File Name Input
        file_label = QLabel('File name:')
        self.file_input = QLineEdit()
        layout1.addWidget(file_label)
        layout1.addWidget(self.file_input)

        # Search Term Input
        search_label = QLabel('Search term:')
        self.search_input = QLineEdit()
        layout2.addWidget(search_label)
```

```
        layout2.addWidget(self.search_input)

        # Scrape Button
        scrape_button = QPushButton('Scrape')
        scrape_button.clicked.connect(self.start_scraping)
        layout3.addWidget(scrape_button)

        #Count input:
        count_label = QLabel('Count: ')
        self.count_input = QLineEdit()
        layout2.addWidget(count_label)
        layout2.addWidget(self.count_input)

        vLayout.addLayout(layout1)
        vLayout.addLayout(layout2)
        vLayout.addLayout(layout3)
        self.setLayout(vLayout)

    def start_scraping(self):
        file_name = self.file_input.text().replace(' ', '_')
        search_term = self.search_input.text().replace(' ', '+')
        count = self.count_input.text()

        if not file_name or not search_term or not count:
            QMessageBox.warning(self, "Missing Input", "Please enter both a
file name and a search term.")
            return

        try:
            # Execute your scraping script (modify the command as needed)
                subprocess.run(["python", "app.py", file_name, search_term,
count], check=True)
            QMessageBox.information(self, "Scraping Complete", f"Data saved
to {file_name}.csv")

            self.close()
        except subprocess.CalledProcessError as e:
            QMessageBox.critical(self, "Scraping Error", f"An error occurred
during scraping:\n{e}")

    def file_name(self):
        return self.file_input.text()

class DataWindow(QWidget):
    close_signal = pyqtSignal()
    def __init__(self, file_name):
        super().__init__()
        self.setWindowTitle('Analyze Data')
        self.setGeometry(500,200, 250, 300)
        self.setWindowIcon(QIcon('icon.png'))
        self.function(file_name)
```

```
        self.initUI()

    def initUI(self):
        layout = QVBoxLayout()
        #Creating buttons
        sale = QPushButton('Visualise Sales')
        sale.clicked.connect(self.df.visualise_sale)
        rating = QPushButton('Visualize Ratings')
        rating.clicked.connect(self.df.visualise_rating)
        file = QPushButton('Open File')
        file.clicked.connect(self.open_file)
        #Adding buttons to layout
        layout.addWidget(sale)
        layout.addWidget(rating)
        layout.addWidget(file)
        #Setting the layout
        self.setLayout(layout)
    def function(self, file_name):
        self.file = file_name
        self.df = DataShow(file_name)

    def open_file(self):
        try:
             os.system(f'cmd /c "start {self.file}.csv"')
        except subprocess.CalledProcessError as e:
                  QMessageBox.critical(self,  "File  Error",  f"Cannot  open
{self.file}.csv")

    def closeEvent(self, event):
        self.close_signal.emit()
        super().closeEvent(event)
        reply = QMessageBox.question(self, "Save File..?",
                                     f"Do you want to save {self.file}.csv?",
                                           QMessageBox.Yes | QMessageBox.No,
QMessageBox.Yes)
        if reply == QMessageBox.No:
            os.system(f'cmd /c del /Q {self.file}.csv')
            event.accept()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    main = ScraperGUI()
    main.show()
    app.exec_()
    file_name = main.file_name()
    data = DataWindow(file_name)
    data.show()
    app.exec_()
```

Interface.py creates the graphical user interface (GUI) for your Flipkart scraping application using PyQt5. It defines two main classes: ScraperGUI and DataWindow.

**1. ScraperGUI Class:**

This class creates the main window where the user inputs the filename, search term, and count.

1. **__init__(self):** Initializes the window, sets its title, icon, fixed size, and position. Calls initUI() to create the user interface elements.
2. **initUI():** Creates the layout and widgets.
    - Three QHBoxLayout objects are used for horizontal arrangement of widgets within the window. These layouts hold labels and input fields.
    - A QVBoxLayout arranges these horizontal layouts vertically.
    - Labels and QLineEdit widgets are created for file name, search term, and count input.
    - A "Scrape" button (QPushButton) is created and connected to the start_scraping method.
3. **start_scraping():** This is the core function triggered when the "Scrape" button is clicked.
    - It retrieves the values entered by the user for file name, search term, and count. Spaces in the file name are replaced with underscores, and spaces in the search term are replaced with plus signs.
    - It checks if all input fields are filled. If not, a warning message is displayed.
    - It uses subprocess.run to execute the app.py script (which handles the actual scraping). The file name, search term, and count are passed as arguments.
    - Upon successful completion, an information message is shown, and the window closes using self.close().
    - If an error occurs during scraping, a critical error message is displayed.
4. **file_name():** A simple getter method that returns the file name entered by the user.

**2. DataWindow Class:**

This class creates a separate window for data analysis after scraping is complete.

1. **close_signal = pyqtSignal():** This creates a custom signal that is emitted when the window is closed. It's used for communication between the DataWindow and the main application.
2. **__init__(self, file_name):** Initializes the DataWindow with the chosen file_name. It sets the window's properties (title, geometry, icon) and calls function(file_name) and initUI() for setup.
3. **initUI():** Creates the layout and widgets for the data analysis window. It includes buttons for visualizing sales, ratings, and opening the CSV file. These buttons are connected to respective methods (not fully shown in the given code).
4. **function(self, file_name):** Initializes self.file (stores the filename) and an instance of DataShow for data visualization, passing the file_name.
5. **open_file():** Opens the scraped CSV file using the default system application associated with .csv files. It uses os.system and start to achieve this.
6. **closeEvent(self, event):** This method is called when the window is closed.
    - self.close_signal.emit() emits the custom close signal.
    - A message box asks the user if they want to save the CSV file.
    - If the user clicks "No," the CSV file is deleted using os.system(f'cmd /c del /Q {self.file}.csv').

**3. Main Application Execution:**

- The if __name__ == '__main__' block creates the main application instance and shows the ScraperGUI window.
- After the scraping is done and the ScraperGUI window closes, a DataWindow is created and shown.

# Chapter 4
# RESULTS

## 4.1 OUTPUT:

The project creates a PyQt GUI application that allows users to scrape data from Flipkart and then visualize it.

1. **Data Scraping:** The user enters a search term and a count (number of items) into the GUI. This information is then passed to a Scrapy spider (flipscraper) which scrapes data from Flipkart based on the search term and count. The scraped data is saved into a CSV file with the user-specified name.
2. **Data Visualization:** After scraping, a new window opens offering options to visualize the scraped data. The user can visualize the sales data (maximum and minimum sale prices for each company) as a line graph and the average ratings for each company as a bar chart. There's also an option to directly open the CSV file.
3. **File Handling:** The application attempts to handle file operations gracefully. It replaces spaces in the filename with underscores to prevent potential issues. Before closing the data visualization window, it asks the user if they want to save the CSV file. If the user chooses "No," the file is deleted.
4. **Graphical User Interface:**

   **i)** Window 1: ScraperGUI

   - Purpose: Takes user input for the filename, search term, and the number of items to scrape.
   - Functionality:
     - Uses QLineEdit widgets for filename, search term, and count.
     - A "Scrape" button triggers the scraping process.
     - Input validation: Checks if all fields (filename, search term, count) are filled. Displays a warning message if any are empty.
     - Error Handling: Uses try-except blocks to catch errors during the scraping process. Displays error messages if the scraping script fails.
     - File Handling: Replaces spaces in the filename with underscores.

- Scraping Execution: Calls app.py with the user-provided arguments using the subprocess module.
- Closes after successful scraping, passing the filename to the second window.

i) Window 2: DataWindow

- Purpose: Displays the scraped data and offers visualization options.
- Functionality:
  - Receives the filename from the first window.
  - Reads the CSV data using pandas.
  - Provides buttons for:
    - Visualizing sales data (max and min prices) as a line graph using matplotlib.
    - Visualizing average ratings as a bar chart using matplotlib.
    - Opening the CSV file directly.
  - File Handling: Prompts the user whether to save the CSV file upon closing. If the user chooses "No," the file is deleted.

### *4.1.1 The GUI of FlipKartScraper:*

The FlipKart Scarper GUI provides the fields File name, Search term, Count that are provided to accept the user input and the scrape button to start the scraping process.

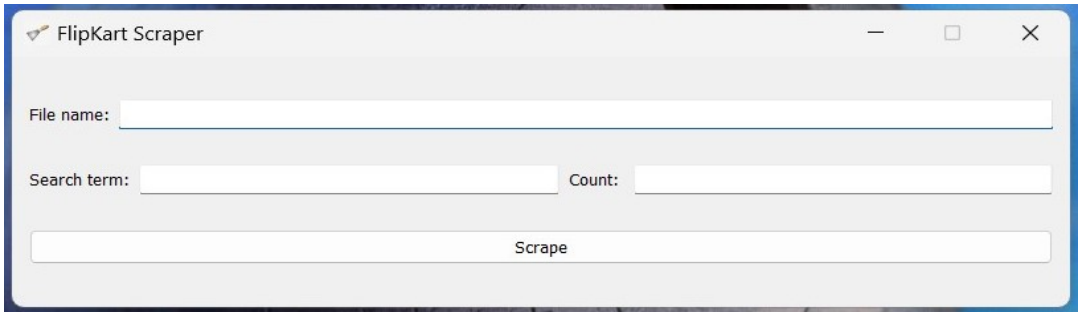


*Fig 7: FlipKart Scraper - takes required input from user*

If the user input is missing in any of the fields and the user wanted to scrape, then it pops up a warning message box prompting the user to enter all the required inputs.

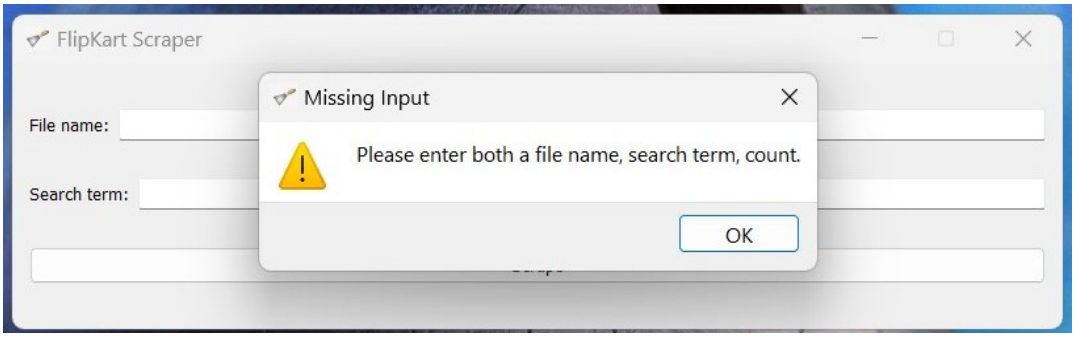


*Fig 8: Warning message Box*

If any error raises during the execution or the spider crawling, an error message box is popped up. The possible reason for the error could be invalid search term, Invalid count, poor network connection etc.

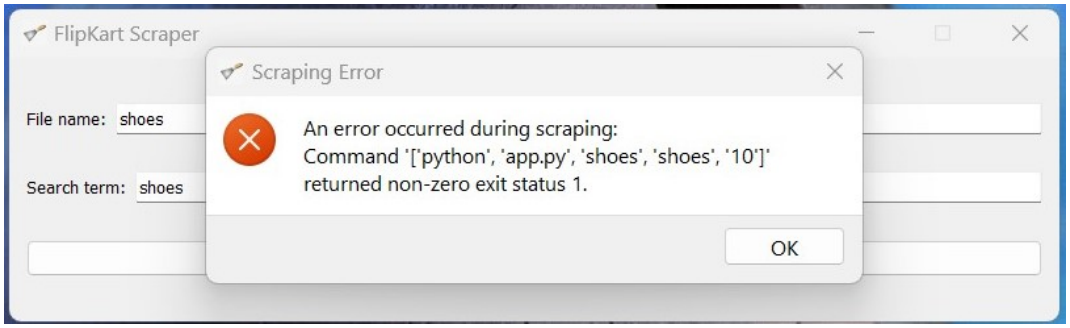


*Fig 9: Error message box*

If the data is successfully scraped and saved into a csv file without encountering any errors, then a message box is popped up indicating that the data is saved

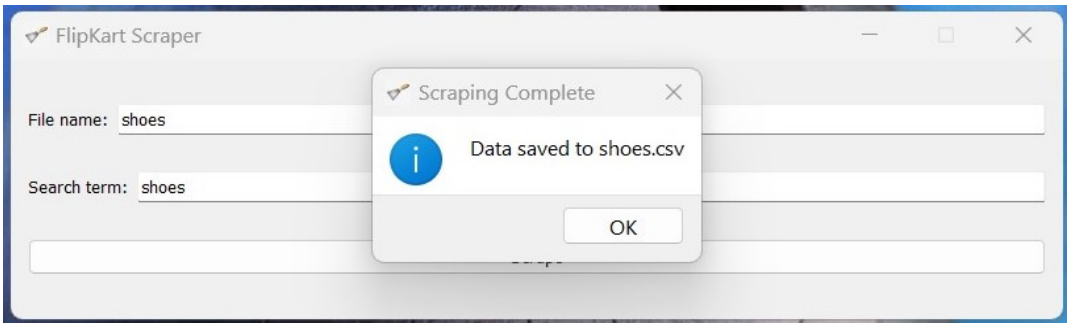


*Fig 10: Message box indicating that data is saved into a csv file*

After scraping, as the data is saved into a csv file, a new window “Analyze data” opens that provide some data insights about the scraped data like “Price graph”, “Ratings graph”

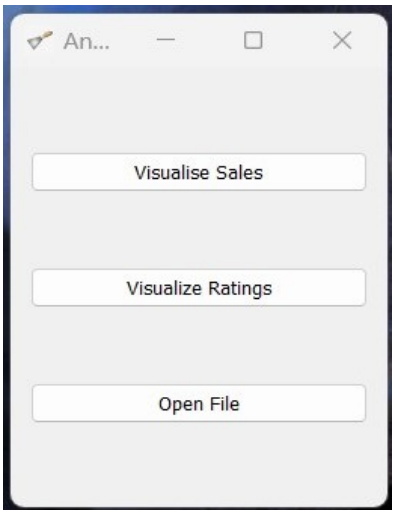


*Fig 11:' Analyse Data' window*

On clicking “Visualise Sales” button, a line plot showing price details appears as below:

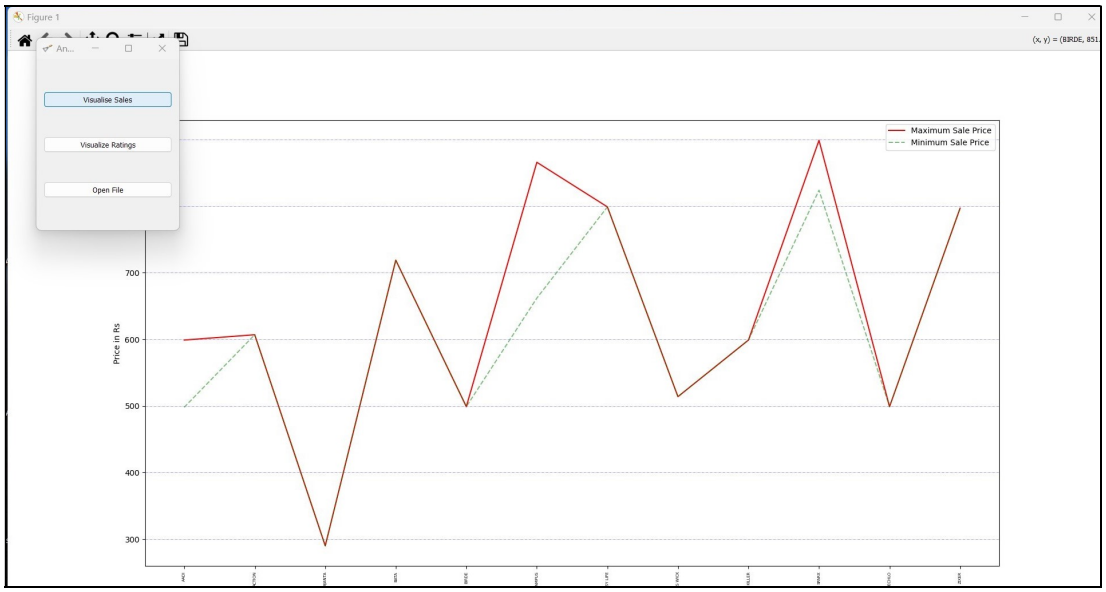


*Fig 12: Visualizing the price details as line plot*

On clicking “Visualise Ratings” button, a bar graph showing ratings details appears as below:

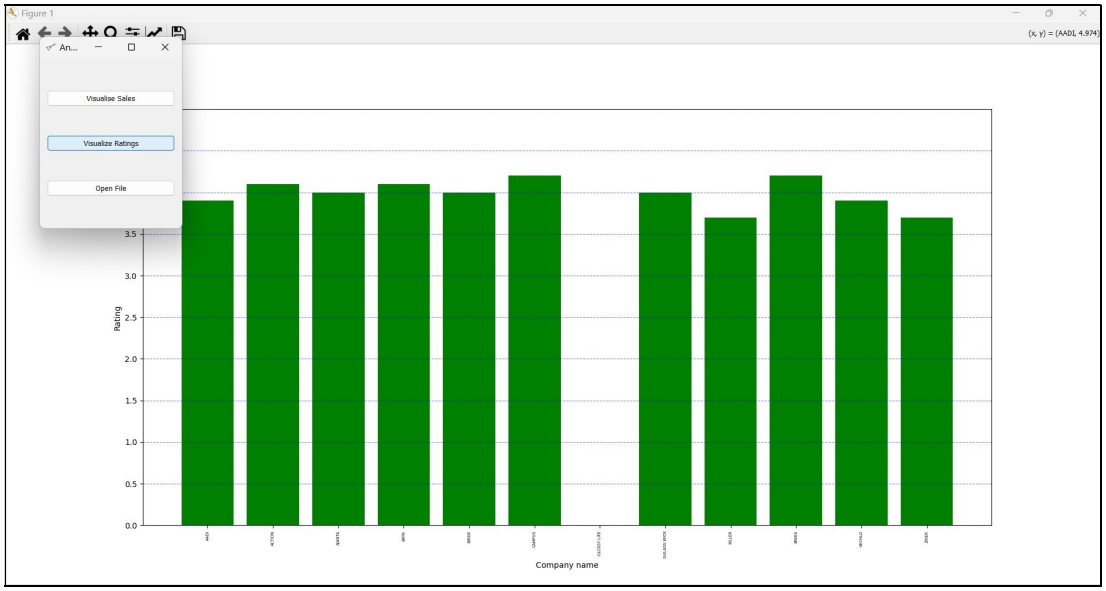


*Fig 13:Visualizing the ratings details as bar plot*

On clicking “Open File” button, the file in which scraped data is saved opens that appears as shown below:

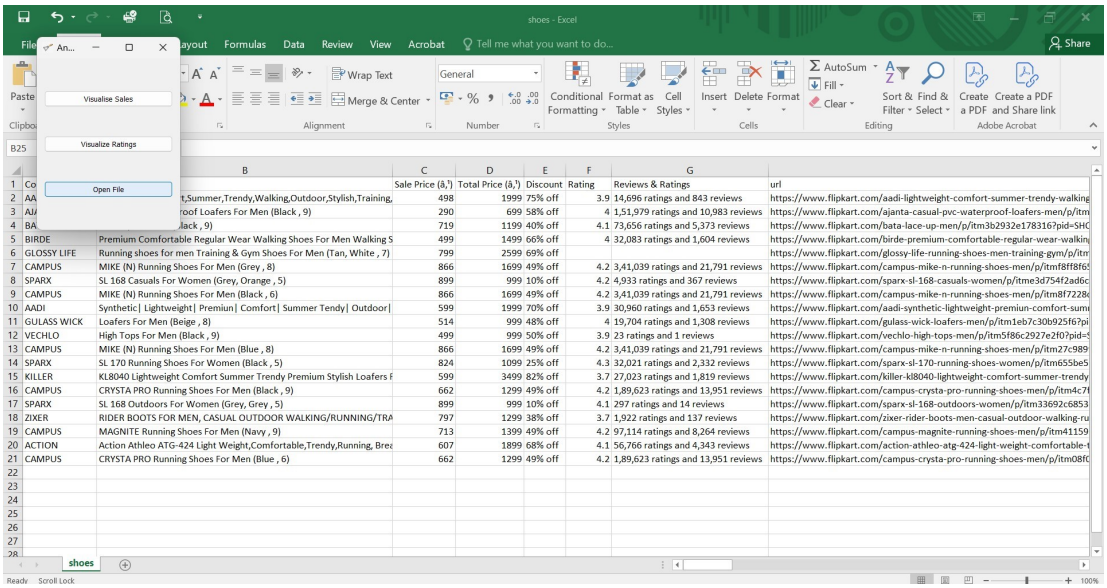


*Fig 14: CSV file of scraped data*

When closing the “Analyse Data” window, a message box pops up asking the user whether to save the file or not. If the user selects “No”, the file is deleted

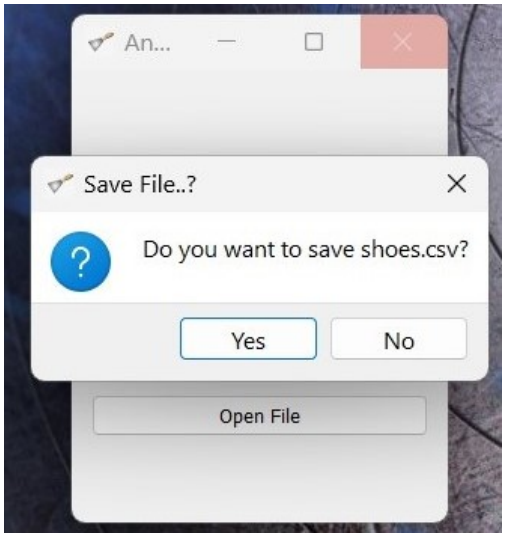


*Fig 15: Save file*

# Chapter 5
# CONCLUSIONS

## 5.1 SUMMARY:

This project is a Flipkart web scraper with a graphical user interface. It uses Scrapy to extract product data and PyQt5 for user interaction and data visualization.

### 5.1.1 Key Features:

1. **User-Friendly Interface:** The PyQt5 GUI simplifies the scraping process by allowing users to specify the output filename, search term, and the number of products to scrape. It also handles potential errors gracefully, informing the user about missing inputs or scraping issues.
2. **Data Extraction:** The Scrapy spider (flipscraper.py) extracts detailed product information from Flipkart search results, including company name, description, sale price, original price, discount, rating, and the number of reviews. The spider handles pagination to scrape multiple pages of results.
3. **Data Storage and Visualization:** Scraped data is stored in a user-specified CSV file. The GUI then provides options to visualize the sales prices and ratings using Matplotlib, directly from the CSV data. It also allows the user to directly open the CSV file.
4. **Data Management:** Upon closing the visualization window, the user is prompted whether to save or delete the generated CSV file, providing a convenient way to manage the scraped data.

### 5.1.2 Technical Components:

1. **Scrapy:** Handles the core web scraping functionality, including request handling, data extraction, and pagination.
2. **PyQt5:** Provides the GUI elements for user interaction, data visualization, and file management.
3. **Matplotlib:** Used for creating visualizations of the scraped data.
4. **Subprocess:** Facilitates the execution of the Scrapy spider from within the Python GUI application.
5. **Pandas:** Used for reading and processing data from the CSV file for visualization.

This project effectively combines web scraping, data processing, and visualization into a user-friendly application, making it easy to collect and analyze product data from Flipkart.

## 5.2 CONCLUSIONS

This project demonstrates a functional yet basic web scraper for Flipkart product data. Here's a breakdown of its strengths and weaknesses, along with potential improvements:

### 5.2.1 Strengths:

1. **Targeted Data Extraction:** The scraper successfully extracts specific product details from Flipkart, addressing the core project goal.
2. **GUI Enhancement:** The PyQt5 GUI provides a user-friendly interface for inputting search parameters and visualizing results, making the tool more accessible than a command-line-only solution.
3. **Data Visualization:** Integrating Matplotlib allows for basic visualization of price ranges and ratings, offering immediate insights into the scraped data.
4. **Data Persistence:** The CSV output provides a persistent record of the scraped data, allowing for further analysis or use in other applications.

### 5.2.2 Weaknesses:

1. **Flipkart's Anti-Scraping Measures:** The scraper is vulnerable to Flipkart's anti-scraping mechanisms. Implementing techniques like rotating user agents, using proxies, or adding delays could improve its resilience.
2. **Lack of Modularity and Maintainability:** The spider's code for data extraction is tightly coupled with the website's structure. Changes in Flipkart's HTML could break the scraper. A more modular and adaptable approach, possibly using XPath expressions or CSS selectors more strategically, would be desirable.
3. **Performance:** The scraper's performance might be limited, especially when scraping many products. Optimizations like asynchronous requests or concurrent processing could significantly speed up the process.

### 5.3 POTENTIAL IMPROVEMENTS:

1. **Data Cleaning and Preprocessing:** Implementing data cleaning steps to handle missing values, format inconsistencies, and normalize text.
2. **Circumvent Anti-Scraping:** Using techniques like rotating user agents, proxies, and randomized delays to avoid being blocked by Flipkart.
3. **Performance Optimization:** Explore asynchronous requests, concurrent processing, or other optimization techniques to improve scraping speed.
4. **Utilize Scrapy Features:** Define items in items.py and implement middleware and pipeline components for enhanced data handling, processing, and storage.

By addressing these weaknesses and incorporating the suggested improvements, the project can evolve into a more robust, efficient, and maintainable web scraping solution.

**REFERENCES:**


1. **Python Libraries**:
    1. **Pandas**: For data manipulation and analysis. Documentation available at: https://pandas.pydata.org.
    2. **Matplotlib**: For data visualization. Documentation available at: https://matplotlib.org.
    3. **NumPy**: For numerical operations. Documentation available at: https://numpy.org.
    4. **PyQt5**: For building the graphical user interface. Documentation available at: https://riverbankcomputing.com/software/pyqt/intro.
    5. **Scrapy**: For web scraping. Documentation available at: https://scrapy.org.
2. **Development Tools**:
    1. **Python**: Programming language used for the project. Official documentation available at: https://python.org.
    2. **Integrated Development Environment (IDE)**: Visual Studio Code
3. **Web Sources**:
    1. Flipkart Website: Source of data for scraping. https://www.flipkart.com.
4. **Operating System Utilities**:
    1. `os` and `subprocess` modules for file and process management.
    2. System command-line interface for executing commands (`cmd` in Windows).
5. **Miscellaneous**:
    1. **CSV Format**: For storing scraped data. Specifications available at: https://tools.ietf.org/html/rfc4180.

# APPENDIX

**flipscraper.py**

```
import scrapy

class FlipscraperSpider(scrapy.Spider):

    name = "flipscraper"
    #Setting the allowed domains to restrict our spider from crawling into
unnecessary domains
    allowed_domains = ["flipkart.com"]
    #Name of spider
    i = 0

    def __init__(self, search_term=None, num='all', *args, **kwargs):
        super(FlipscraperSpider, self).__init__(*args, **kwargs)
        self.search_term = search_term.replace(' ','+')
        if num!= 'all':
            self.num = int(num)
        else:
            self.num = num
        #The wepage from where the spider starts crawling
        self.start_urls =
[f"https://www.flipkart.com/search?q={self.search_term}&as=on&as-
show=on&otracker=AS_Query_TrendingAutoSuggest_2_0_na_na_na&otracker1=AS_Quer
y_TrendingAutoSuggest_2_0_na_na_na&as-pos=2&as-
type=TRENDING&suggestionId=shoes&requestId=8f0b3ee0-959c-4c33-872a-
b620540cb6b5"]


    #Method to generate a dictionary of details related to the item
    def parse_item(self, response):
        if response.status > 200:
            response = response.replace(status=201)
        link = response.url
        item = response.xpath("//div[@class='C7fEHH']")

        sale_price = float((item.xpath("//div[@class='x+7QT1
dB67CR']/div[@class='UOCQB1']/div/div[@class='Nx9bqj
CxhGGd']/text()").get()).strip('₹').replace(",",''))

        total = item.xpath("//div[@class='x+7QT1
dB67CR']/div[@class='UOCQB1']/div/div[@class='yRaY8j
A6+E6v']/text()").extract()
        if total is not None:
            total = float(total[-1].replace(",",''))

        rating = item.xpath("//div[@class='DRxq-
P']/div/div/span[@class='Y1HWO0']/div/text()").get()
```

```
        if rating is not None:
            rating = float(rating)

        yield{
            'Company_name':
item.xpath("//div/h1/span[@class='mEh187']/text()").get().replace("\xa0","")
.upper(),
            'Descrption': (" ".join(item.xpath("///div/h1/span[@class='VU-
ZEz']/text()").extract())).replace("\xa0",''),
            'Sale Price (₹)': sale_price,
            'Total Price (₹)': total,
            'Discount': item.xpath("//div[@class='x+7QT1
dB67CR']/div[@class='UOCQB1']/div/div[@class='UkUFwK WW8yVX
dB67CR']/span/text()").get(),
            'Rating': item.xpath("//div[@class='DRxq-
P']/div/div/span[@class='Y1HWO0']/div/text()").get(),
            'Reviews & Ratings': item.xpath("//div[@class='DRxq-
P']/div/div/span[@class='Wphh3N']/span/text()").get(),
            'url': link
        }

    #Method to parse the webpage and follow the links to the next page
    def parse(self, response):
        if response.status > 200:
            response = response.replace(status=201)

        items = response.xpath("//div[@class='_1sdMkc
LFEi7Z']/a[@class='rPDeLR']/@href").extract()
        for item in items:
            if self.num != 'all':
                if self.i < int(self.num):
                    item_url = 'https://www.flipkart.com' + item
                    yield response.follow(item_url,
callback=self.parse_item)
                    self.i += 1
                else:
                    return
            else:
                item_url = 'https://www.flipkart.com' + item
                yield response.follow(item_url, callback=self.parse_item)

        next_page =
response.xpath("//div[@class='_1G0WLw']/nav[@class='WSL9JP']/a/@href").extra
ct()[-1]
        print(next_page)
        url = 'https://www.flipkart.com' + next_page
        yield response.follow(url, callback=self.parse)
```

**App.py**

```
import sys
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import pandas.errors


class DataShow():
    def __init__(self, file_name):
        try:
            self.df = pd.read_csv(f'{file_name}.csv')
        except:
            sys.stderr.write("Error Reading CSV file")
    def visualise_sale(self):
        df_sale_max = self.df.groupby('Company_name')['Sale Price
(₹)'].max().to_dict()
        df_sale_min = self.df.groupby('Company_name')['Sale Price
(₹)'].min().to_dict()
        plt.figure(figsize = (50,50))
        plt.plot(df_sale_max.keys(),df_sale_max.values(), color = 'r',label
= 'Maximum Sale Price')
        plt.plot(df_sale_min.keys(),df_sale_min.values(),  color =
'g',linestyle = '--', alpha = 0.5, label = 'Minimum Sale Price')
        plt.grid(axis= 'y', color=  'b', linestyle= '-.', linewidth = 0.5,
alpha = 0.5)
        plt.yticks(fontsize = 10)
        plt.xticks(fontsize = 5, rotation=90)
        plt.xlabel('Company name')
        plt.ylabel('Price in Rs')
        plt.legend()
        plt.show()

    def visualise_rating(self):
        plt.ylabel('Rating')
        plt.xlabel('Company name')
        df_rating =
self.df.groupby('Company_name')['Rating'].mean().to_dict()
        plt.grid(axis = 'y', color = 'b', linestyle = '--', alpha = 0.5 )
        plt.bar(df_rating.keys(),df_rating.values(), color = 'g')
        plt.yticks(np.arange(0, 5.1, 0.5), fontsize = 10)
        plt.xticks(fontsize = 5, rotation=90)
        plt.show()

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python app.py <output_file> <search_term> <count>")
        sys.exit(1)
```

```
    name = sys.argv[1]
    search_term = sys.argv[2]
    count = sys.argv[3]

    os.system(f'cmd /c "scrapy crawl flipscraper -a
search_term={search_term} -a num={count} -O {name}.csv"')

    path = f'flipkart\{name}.csv'

    os.system('cmd /c cls')
```

**Interface.py**

```
import sys
import os
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
import subprocess  # For running external scripts
from app import DataShow
import pandas as pd


class ScraperGUI(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('FlipKart Scraper')
        self.setWindowIcon(QIcon('icon.png'))
        self.setFixedSize(800, 200)
        self.setGeometry(500, 100, 800, 200)
        self.initUI()

    def initUI(self):
        layout1 = QHBoxLayout()
        layout2 = QHBoxLayout()
        layout3 = QHBoxLayout()
        vLayout = QVBoxLayout()

        # File Name Input
        file_label = QLabel('File name:')
        self.file_input = QLineEdit()
        layout1.addWidget(file_label)
        layout1.addWidget(self.file_input)

        # Search Term Input
        search_label = QLabel('Search term:')
        self.search_input = QLineEdit()
```

```
        layout2.addWidget(search_label)
        layout2.addWidget(self.search_input)

        # Scrape Button
        scrape_button = QPushButton('Scrape')
        scrape_button.clicked.connect(self.start_scraping)
        layout3.addWidget(scrape_button)

        #Count input:
        count_label = QLabel('Count: ')
        self.count_input = QLineEdit()
        layout2.addWidget(count_label)
        layout2.addWidget(self.count_input)

        vLayout.addLayout(layout1)
        vLayout.addLayout(layout2)
        vLayout.addLayout(layout3)
        self.setLayout(vLayout)

    def start_scraping(self):
        file_name = self.file_input.text().replace(' ', '_')
        search_term = self.search_input.text().replace(' ', '+')
        count = self.count_input.text()

        if not file_name or not search_term or not count:
            QMessageBox.warning(self, "Missing Input", "Please enter both a
file name, search term, count.")
            return

        try:
            # Execute your scraping script (modify the command as needed)
            subprocess.run(["python", "app.py", file_name, search_term,
count], check=True)
            QMessageBox.information(self, "Scraping Complete", f"Data saved
to {file_name}.csv")

            self.close()
        except subprocess.CalledProcessError as e:
            QMessageBox.critical(self, "Scraping Error", f"An error occurred
during scraping:\n{e}")

    def file_name(self):
        return self.file_input.text()

class DataWindow(QWidget):
    close_signal = pyqtSignal()
    def __init__(self, file_name):
        super().__init__()
        self.setWindowTitle('Analyze Data')
        self.setGeometry(500,200, 250, 300)
```

```
        self.setWindowIcon(QIcon('icon.png'))
        self.function(file_name)
        self.initUI()

    def initUI(self):
        layout = QVBoxLayout()
        #Creating buttons
        sale = QPushButton('Visualise Sales')
        sale.clicked.connect(self.df.visualise_sale)
        rating = QPushButton('Visualize Ratings')
        rating.clicked.connect(self.df.visualise_rating)
        file = QPushButton('Open File')
        file.clicked.connect(self.open_file)
        #Adding buttons to layout
        layout.addWidget(sale)
        layout.addWidget(rating)
        layout.addWidget(file)
        #Setting the layout
        self.setLayout(layout)
    def function(self, file_name):
        self.file = file_name
        try:
            self.df = DataShow(file_name)
        except:
            QMessageBox.critical(self, "File Error", "No data found in
file")
            sys.exit(0)

    def open_file(self):
        try:
             os.system(f'cmd /c "start {self.file}.csv"')
        except subprocess.CalledProcessError as e:
            QMessageBox.critical(self, "File Error", f"Cannot open
{self.file}.csv")

    def closeEvent(self, event):
        self.close_signal.emit()
        super().closeEvent(event)
        reply = QMessageBox.question(self, "Save File..?",
                                     f"Do you want to save
{self.file}.csv?",
                                     QMessageBox.Yes | QMessageBox.No,
QMessageBox.Yes)
        if reply == QMessageBox.No:
            os.system(f'cmd /c del /Q {self.file}.csv')
            event.accept()


if __name__ == '__main__':
    app = QApplication(sys.argv)
```

```
    main = ScraperGUI()
    main.show()
    app.exec_()
    file_name = main.file_name()
    data = DataWindow(file_name)
    data.show()
    app.exec_()
```