Simen Owen Academic Proceedings Series

3rd International Conference on Electronics, Engineering, Computer Science and Applied Development (EESD 2026)

Article

Research on Python-Enabled Web Crawling and Data Visualization for Structured Data Analysis

Xizhou Deng [1,*]

[1] Qingdao Jinqiu International School, Qingdao, China
* Correspondence: Xizhou Deng, Qingdao Jinqiu International School, Qingdao, China

Abstract: This paper comprehensively analyzes the application of the Python programming language in the domains of web crawling and data visualization, specifically focusing on structured data analysis. With the unprecedented and rapid growth of Internet data across various sectors, traditional manual data collection methods can no longer meet the contemporary needs of efficient, large-scale data analysis. Consequently, automated extraction techniques have become indispensable. Python provides robust technical support and a highly versatile ecosystem for webpage data acquisition, data cleaning, structured processing, and visual presentation. This is achieved through the deployment of powerful libraries such as Requests, BeautifulSoup, Scrapy, and Selenium for extraction, alongside Pandas for data manipulation. Furthermore, Matplotlib, Seaborn, Plotly, and Pyecharts are utilized for advanced graphical representation. This study systematically discusses the fundamental processes of Python-based web crawling, detailing the methodologies of data cleaning, transformation, and formatting. Additionally, it evaluates the strategic selection of appropriate visualization tools tailored for diverse analytical scenarios and business intelligence requirements. The empirical results demonstrate that Python-driven frameworks can effectively and significantly improve data collection efficiency, enhance overall data quality, and facilitate deeper result interpretation. However, despite these advantages, several critical challenges remain. Issues such as sophisticated anti-crawling mechanisms, strict data privacy compliance, inherently unstable raw data quality, and the potential for subjective chart interpretation still require careful attention and ongoing methodological refinement.

Keywords: python; web crawling; data visualization; data analysis; structured data; data extraction

Received: 04 April 2026
Revised: 19 May 2026
Accepted: 01 June 2026
Published: 05 June 2026

1. Introduction

With the rapid development of the Internet and information technology, massive amounts of online data have become an important resource for academic research, business analysis, public opinion monitoring, and decision-making. Compared with traditional manual data collection methods, automated data acquisition based on programming languages can greatly improve the efficiency and scale of data collection. Among various programming languages, Python has been widely used in data-related fields because of its simple syntax, strong readability, rich third-party libraries, and flexible development environment. In particular, Python provides powerful support for data acquisition, data cleaning, statistical analysis, and visualization, making it suitable for building an integrated workflow from web crawling to data analysis [1].

In the field of data processing, Python has gradually become an important tool for handling structured and semi-structured data. Libraries such as Pandas provide efficient data structures and analytical functions, enabling researchers to clean, transform, organize, and analyze complex datasets in a convenient way. This technical advantage

allows Python to process large amounts of web data after collection and convert raw information into structured datasets that can be further analyzed. Therefore, Python is not only a programming language for data acquisition, but also an important platform for subsequent data processing and analysis [2].

Data visualization is another important component of Python-based data analysis. Raw data often contains complex patterns that are difficult to understand directly through numerical tables [3]. Visualization tools can transform abstract data into intuitive charts, helping users identify trends, distributions, correlations, and abnormal values. Matplotlib, as one of the most representative visualization libraries in Python, provides a flexible two-dimensional graphics environment and supports the generation of high-quality figures for research and practical applications. With the support of visualization libraries, Python can present data analysis results in a more intuitive and readable form.

In addition to data processing and visualization, Python is also widely used in web crawling. Web crawling technology can automatically collect target information from websites, extract useful content from web pages, and store the data for further processing. Existing studies have shown that Python-based web scraping can be effectively combined with visualization analysis. For example, research on movie website data has demonstrated how Python tools can be used to collect online data and visualize information such as movie types, scores, and trends [4]. This indicates that Python can support a complete technical process from online data acquisition to visual result presentation.

However, the application of web crawling also faces several problems. Since web data usually comes from public websites, researchers and developers must consider website access rules, data privacy, copyright, legality, and research ethics. Web scraping improves the efficiency of data collection, but improper crawling behavior may lead to legal disputes, privacy risks, or excessive pressure on website servers [5]. Therefore, the study of Python web crawling should not only focus on technical implementation, but also pay attention to standardized, ethical, and responsible data use.

Recent studies further show that Python-based crawling tools such as BeautifulSoup and Selenium have practical value in extracting online reviews and supporting review analytics. These applications demonstrate that Python can be used not only for simple webpage data extraction, but also for deeper analysis of user opinions, market information, and social trends. By combining web crawling, data cleaning, statistical processing, and visualization, Python provides a complete and efficient solution for transforming online data into useful analytical results.

Based on the above background, this paper focuses on the application of Python in web crawling and data visualization [3]. It first reviews the existing research on Python data processing, web crawling, and visualization technologies. Then, it analyzes the application process of Python in web data acquisition, data cleaning, structured processing, visualization tool selection, and result analysis. Finally, it discusses the practical value and limitations of Python-based web crawling and data visualization, aiming to provide a reference for future data analysis research and application practice.

2. Literature Review

Existing studies have provided a theoretical and technical foundation for the application of Python in web crawling and data visualization [6]. As a general-purpose programming language, Python has been widely adopted in scientific computing and data analysis because of its concise syntax, strong extensibility, and rich ecosystem of open-source libraries. It has been noted that Python is suitable for scientific computing because it combines readability, interactive development, cross-platform support, and the ability to integrate with high-performance numerical computing tools. This indicates that Python is not only a basic programming language but also an effective technical environment for data-oriented research.

With the continuous development of the Python ecosystem, scientific computing libraries have further improved Python's ability to process and analyze complex data.

SciPy provides a large number of fundamental algorithms for scientific computing, including optimization, statistics, interpolation, signal processing, and linear algebra, which expands Python's application value in data analysis and computational research. At the same time, NumPy offers efficient array programming support and has become a basic component of the scientific Python ecosystem, providing the underlying data structure and numerical computing capability for many data analysis tasks. These studies show that Python has formed a relatively mature technical foundation for data processing, which is necessary for the later stages of web data cleaning, transformation, and analysis [4].

In terms of data visualization, Python has also developed from basic static chart generation to more advanced statistical and interactive visualization. Seaborn has been introduced as a statistical data visualization library that provides high-level interfaces for exploring data distribution, relationships, and categorical patterns. This type of visualization tool helps users better understand hidden patterns in datasets and improves the interpretability of analysis results [2]. In addition, Altair has been introduced as an interactive statistical visualization library for Python, which supports declarative visualization design and helps users create flexible visual representations for data exploration and analysis. Therefore, Python visualization tools can meet different levels of analytical needs, from simple data presentation to statistical exploration and interactive visual analysis.

Research on web data extraction has also developed over a long period. Different web data extraction tools have been reviewed, and a classification framework for understanding various extraction methods has been proposed [5]. This early research laid a foundation for later studies on web crawling and web scraping by clarifying how structured information can be extracted from semi-structured or unstructured web pages. With the expansion of online information, web scraping has gradually become an important approach for collecting Internet data. The state of the art of web scraping has been summarized, discussing its methods, tools, and application areas, showing that web scraping has become a systematic technical field rather than a simple webpage copying process.

Further studies have examined the technical processes and application scenarios of web crawling and web scraping. The working mechanisms, techniques, approaches, and applications of web scraping and web crawling have been discussed, pointing out their relevance to business intelligence, artificial intelligence, data science, and big data analysis. This suggests that web crawling is not only used for simple information collection but also serves as an important data acquisition method for more complex analytical systems [7]. In large-scale data environments, traditional local scraping methods may face limitations in computing power, storage, and scalability. Cloud-based web scraping for big data applications has been proposed, indicating that web scraping can be expanded through cloud platforms to support large-scale online data collection. This reflects the development trend of web crawling from small-scale data extraction to scalable and distributed data acquisition.

Application-oriented studies have also demonstrated the practical value of web scraping in different domains [8]. A web scraping tool has been developed for extracting newspaper and image data, showing that automated web data collection can be applied to diverse web resources such as news websites, blogs, and image platforms. These studies indicate that web crawling technology has broad application potential in information extraction, public opinion analysis, market research, and multimedia data collection.

Overall, previous studies have mainly focused on three aspects: Python-based scientific computing and data processing, Python visualization tools, and web scraping technologies. These studies provide strong support for the application of Python in data acquisition and analysis [9]. However, some existing research focuses more on single technical tools or specific application cases, while fewer studies systematically discuss the complete process from web crawling, data cleaning, structured processing, visualization design, to result analysis. Therefore, this paper takes Python as the core technical

environment and discusses its application in web crawling and data visualization from an integrated workflow perspective, aiming to provide a more systematic reference for online data collection and visual analysis.

3. Application of Python in Web Crawling and Data Processing

3.1. Web Data Acquisition Based on Python Crawling Technology

Web crawling is an important technical method for obtaining data from the Internet automatically. Compared with manual data collection, Python-based web crawling can collect large amounts of webpage information more efficiently and can support subsequent data cleaning, statistical analysis, and visualization. In general, a web crawler simulates the process of human access to webpages. It sends requests to target websites, receives webpage responses, parses webpage content, extracts useful information, and stores the collected data in a structured format. Therefore, Python web crawling is not only a data acquisition technique but also the starting point of an integrated data analysis process [10].

The basic working mechanism of a web crawler includes several steps [11]. First, the crawler sends a request to the target URL through the HTTP or HTTPS protocol. After receiving the request, the server returns a response, which may include HTML documents, JSON data, images, or other resources. Second, the crawler parses the returned webpage content and locates the target data according to HTML tags, CSS selectors, XPath paths, or regular expressions. Third, the extracted data are temporarily stored and further processed according to the research purpose. Finally, the crawler may continue to access the next page by identifying pagination links or dynamically generated URLs. This process allows Python programs to obtain data continuously from multiple webpages.

In practical applications, webpage data acquisition can be divided into static webpage crawling and dynamic webpage crawling. Static webpages usually return complete HTML content after a request is sent [3]. For this type of webpage, Python libraries such as Requests and BeautifulSoup are commonly used. Requests is mainly responsible for sending URL requests and obtaining webpage responses, while BeautifulSoup is used to parse HTML documents and extract specific text, links, tables, and other webpage elements. This method is simple, efficient, and suitable for collecting data from news websites, information portals, academic pages, and ordinary HTML-based websites.

However, many modern websites use JavaScript to load data dynamically. In such cases, the required data may not appear directly in the original HTML source code. For dynamic webpages, Selenium is often used to simulate browser behavior, such as clicking buttons, scrolling pages, entering keywords, and waiting for page elements to load. Selenium is especially useful for websites that require interaction before data can be displayed. In addition, some dynamic webpages transmit data through API interfaces. If the API request rules can be identified, Python can directly obtain JSON-format data through Requests, which is usually more efficient than browser simulation [2].

For larger and more complex crawling tasks, Scrapy provides a more systematic crawling framework. Compared with simple Requests-based crawling, Scrapy supports request scheduling, asynchronous processing, automatic link following, item pipelines, middleware, and data storage management [12]. It is suitable for large-scale data collection tasks that involve multiple pages, multiple categories, and continuous crawling. Through Scrapy, developers can define crawling rules, extraction logic, and storage methods in a more standardized way, which improves both efficiency and maintainability.

A complete Python web crawling process usually includes URL construction, request sending, response decoding, HTML parsing, target data extraction, pagination processing, and data storage. For example, when collecting product review data from an e-commerce platform, the program first constructs the URL of each product page or review page. Then it sends requests to obtain webpage content [13]. After that, it extracts information such as product name, price, rating, comment text, comment time, and user information. If the

comments are distributed across multiple pages, the crawler needs to identify the page-turning rule and collect data page by page. Finally, the extracted data can be saved as CSV, Excel, JSON, or database records for further analysis.

In actual crawling tasks, anti-crawling mechanisms are also an important issue. Many websites use technical measures to prevent frequent automated access, such as user-agent detection, IP restriction, login verification, CAPTCHA, request frequency limitation, and dynamic token verification. To improve crawling stability, Python crawlers often need to set request headers, control access frequency, use session management, handle cookies, and design exception-handling mechanisms. However, crawler development should also follow legal and ethical principles. Researchers and developers should respect website terms of service, avoid excessive server pressure, and ensure that the collected data are used within a reasonable and compliant scope.

Python has several advantages in web data acquisition. First, its syntax is simple and readable, which lowers the difficulty of crawler development. Second, Python has a rich ecosystem of third-party libraries, including Requests, BeautifulSoup, Scrapy, Selenium, lxml, and pandas, which can support the whole process from data acquisition to data processing. Third, Python has strong extensibility and can be combined with databases, visualization tools, machine learning models, and big data platforms [14]. Therefore, Python is suitable not only for small-scale webpage data collection but also for more complex data mining and analysis tasks.

Overall, Python crawling technology provides an efficient technical foundation for Internet data acquisition. Through different tools and methods, Python can deal with both static and dynamic webpages and can extract useful information from complex webpage structures. However, successful web crawling does not only depend on technical implementation. It also requires stable crawling logic, reasonable data extraction rules, effective anti-error mechanisms, and awareness of data compliance. Only when web data acquisition is combined with subsequent data cleaning and structured processing can it provide reliable data support for visualization and analysis.

3.2. Data Cleaning and Structured Processing

After web data are collected through Python crawling technology, the next key task is data cleaning and structured processing. Raw webpage data usually cannot be used directly for analysis because they often contain repeated records, missing values, inconsistent formats, irrelevant symbols, HTML tags, abnormal values, and other noisy information. If these problems are not processed properly, they may affect the accuracy of statistical analysis and reduce the reliability of visualization results. Therefore, data cleaning is an essential step between web crawling and data analysis.

The first common problem in crawled data is duplication. When a crawler collects data from multiple pages or repeatedly accesses the same webpage, duplicate records may appear. For example, the same product comment, news title, job posting, or user information may be collected more than once. Python can use Pandas to identify and remove duplicate data through functions such as drop_duplicates(). By comparing key fields such as title, URL, time, user ID, or product ID, repeated records can be deleted effectively. This process helps improve the uniqueness and validity of the dataset.

The second problem is missing data. In webpage crawling, some fields may be empty because of incomplete webpage structures, loading failure, access restrictions, or irregular HTML formats. For instance, some recruitment pages may lack salary information, some product reviews may not contain ratings, and some news pages may not provide publication dates. Pandas provides flexible methods for handling missing values, such as deleting incomplete records, filling missing values with default values, or replacing them with statistical values such as mean, median, or mode. The specific method should depend on the research purpose and the importance of the missing field.

Format inconsistency is another frequent issue in crawled data. Webpage information is usually presented in different textual forms. Dates may appear as "2026-04-28," "April 28, 2026," or "2 days ago"; prices may contain currency symbols; numerical

values may include commas, spaces, or units; and text fields may include unnecessary line breaks or special characters [15]. Python can standardize these formats through string processing, regular expressions, and Pandas transformation functions. For example, date data can be converted into a unified datetime format, price data can be converted into numerical values, and text data can be cleaned by removing extra spaces, punctuation, HTML entities, and invalid characters.

In addition to basic cleaning, abnormal value detection is also necessary. Abnormal values may come from webpage parsing errors, incorrect data extraction rules, or unusual original data. For example, a product price may be extracted as zero, a comment length may be extremely abnormal, or a numerical field may contain irrelevant text. NumPy and Pandas can be used to detect abnormal values through statistical methods, such as maximum and minimum checking, standard deviation analysis, quantile analysis, or logical condition filtering. Removing or correcting abnormal values can make the dataset more stable and suitable for further analysis.

After cleaning, crawled webpage data need to be transformed into structured data. Webpage content is usually semi-structured or unstructured. It may be hidden in HTML tags, tables, paragraphs, links, scripts, or nested JSON objects. Python can extract useful information from these sources and organize it into rows and columns. For example, in an e-commerce review dataset, each row may represent one review, while columns may include product name, price, rating, review text, review time, and user location. In a recruitment dataset, each row may represent one job post, while columns may include job title, salary, company name, city, education requirement, and job description. This transformation makes the data easier to search, compare, calculate, and visualize.

Pandas plays a central role in structured processing. It provides the DataFrame structure, which is suitable for storing and processing tabular data. Through DataFrame operations, researchers can filter records, select columns, rename variables, merge datasets, group data, calculate statistical indicators, and generate new fields. NumPy is often used together with Pandas to support numerical computation, array operations, and mathematical processing. The combination of Pandas and NumPy greatly improves the efficiency of data organization and provides technical support for later visualization and modeling.

Data storage is also an important part of structured processing. After cleaning and transformation, data should be saved in an appropriate format according to the application scenario. CSV files are simple and widely supported, making them suitable for general tabular data storage. Excel files are convenient for manual inspection and report preparation [16]. JSON files are suitable for storing nested or hierarchical data, especially when the original webpage data come from API responses. For larger datasets or long-term data management, relational databases such as MySQL and SQLite can be used to improve data query efficiency and storage reliability. In some large-scale applications, data may also be stored in distributed databases or big data platforms.

Data cleaning and structured processing are not isolated technical steps, but a bridge between web crawling and data visualization [15]. Web crawling solves the problem of data acquisition, but only cleaned and structured data can support reliable analysis. If the collected data remain messy, incomplete, or inconsistent, visualization results may be misleading. For example, repeated records may exaggerate data frequency, missing values may distort distribution patterns, and inconsistent time formats may affect trend analysis. Therefore, before generating charts, it is necessary to ensure that the dataset has clear fields, unified formats, reasonable values, and reliable structure.

Overall, data cleaning and structured processing determine the quality of the entire Python-based data analysis workflow. Through Pandas, NumPy, regular expressions, and appropriate storage methods, Python can transform complex webpage content into standardized datasets [17]. This process improves data usability and provides a solid foundation for visualization and result interpretation. Only after effective cleaning and structured processing can crawled data be further converted into meaningful charts, analytical conclusions, and practical decision-making information.

4. Application of Python in Data Visualization and Result Analysis

4.1. Visualization Methods and Tool Selection

Data visualization is a crucial stage in the Python-based data analysis process. After data are collected, cleaned, and structured, visualization transforms abstract numerical information into intuitive graphical forms. Compared with textual descriptions or raw tables, visual charts present data distribution, variation trends, internal relationships, and abnormal phenomena more clearly. Therefore, data visualization serves not only as a method of result presentation but also as an analytical tool for discovering patterns and supporting decision-making.

In practical data analysis, different chart types are suited to different analytical purposes. Line charts are commonly used to show time-series changes, such as stock price trends, website traffic fluctuations, or public opinion dynamics. Bar charts are ideal for comparing values across categories, such as sales volumes of various products, keyword frequencies, or the number of job postings in different cities. Pie charts effectively display the proportion of different components, although they are better suited for simple composition analysis with a limited number of categories. Scatter plots are often employed to observe relationships between two numerical variables, such as price and sales volume or salary and work experience. Heatmaps are valuable for visualizing correlation matrices, density distributions, or high-dimensional data patterns. In text analysis scenarios, word clouds visually present high-frequency words, enabling users to quickly grasp the main topics within large volumes of textual data.

Python offers a rich array of visualization libraries, among which Matplotlib, Seaborn, Plotly, and Pyecharts are widely utilized. Matplotlib is one of the most fundamental visualization libraries in Python, supporting various basic chart types, including line charts, bar charts, scatter plots, histograms, and pie charts. Due to its high flexibility, Matplotlib allows users to adjust chart size, axes, labels, legends, colors, fonts, and layout details. It is suitable for basic scientific plotting, exploratory data analysis, and academic paper chart generation. However, Matplotlib may require more code to achieve highly polished visual effects.

Seaborn, built on Matplotlib, is more suited for statistical visualization. It provides simpler interfaces and more visually appealing default styles for charts such as distribution plots, box plots, violin plots, regression plots, pair plots, and heatmaps. Compared with Matplotlib, Seaborn is more convenient for analyzing relationships among variables and displaying statistical patterns. For instance, when analyzing review scores, salary distributions, or user behavior data, Seaborn can quickly reveal distribution differences among categories and correlations between variables. Consequently, Seaborn is often employed in exploratory data analysis and statistical result presentation [2].

Plotly is a powerful tool for interactive visualization. Unlike static charts generated by Matplotlib and Seaborn, Plotly enables users to zoom, drag, hover, select, and interact with chart elements. It is particularly suitable for dashboards, web reports, and data analysis scenarios requiring user interaction. For example, when analyzing regional sales data, financial data, or website traffic data, Plotly allows users to explore details by hovering over data points or filtering specific categories [6, 17]. Its interactivity enhances the flexibility of data interpretation, making it more suitable for practical business analysis and online presentations.

Pyecharts is another useful Python visualization library, particularly for webpage-based visualization. Based on ECharts, it supports various chart types, including line charts, bar charts, pie charts, maps, heatmaps, word clouds, and dashboards. One of its key advantages is its ability to generate HTML files directly, facilitating the display of visualization results in browsers or embedding them into web pages. Additionally, Pyecharts performs well in Chinese-language visualization scenarios, as it supports Chinese labels, maps, and interactive webpage charts. As a result, it is commonly used in public opinion analysis, regional data display, web-based reports, and teaching demonstrations.

Different visualization tools have distinct advantages and limitations. Matplotlib offers strong flexibility and is suitable for basic and academic charts, but its default visual style is relatively simple. Seaborn enhances the visual quality and statistical expression of charts but is primarily used for static statistical visualization. Plotly provides robust interactivity, making it ideal for dynamic exploration and dashboard design, though its output files may be larger and require additional configuration. Pyecharts is convenient for generating webpage visualizations and interactive HTML charts, but it may lack the fine-grained academic figure control of Matplotlib. Therefore, tool selection should depend on the data type, analytical purpose, target audience, and presentation environment [17].

Overall, Python visualization methods support both exploratory analysis and final result presentation. For academic research, Matplotlib and Seaborn are often more appropriate due to their ability to generate clear and standardized figures. For business analysis and online reporting, Plotly and Pyecharts may be more suitable, as they offer greater interactivity and webpage display capabilities. In practical projects, these tools can also be used in combination. For instance, Matplotlib can be employed to create basic statistical charts, Seaborn to explore variable relationships, and Plotly or Pyecharts to develop interactive visual reports. By selecting appropriate tools, Python can transform cleaned data into clear, intuitive, and meaningful visual results.

4.2. Visualization-Based Data Analysis and Application Value

Visualization-based data analysis refers to the process of using graphical methods to interpret cleaned and structured data. In Python-based data analysis, visualization is not only the final presentation of results but also an important method for discovering hidden patterns [8]. Through charts, researchers can observe data distribution, compare category differences, identify changing trends, and detect abnormal phenomena more efficiently. Compared with raw data tables, visualization reduces the difficulty of information understanding and improves the efficiency of data interpretation.

One important function of visualization is to reveal the distribution characteristics of data. Histograms, box plots, and density plots can show whether numerical data are concentrated, dispersed, skewed, or affected by outliers. In product review analysis, score distribution charts can indicate whether users generally hold positive, neutral, or negative attitudes toward a product. In recruitment data analysis, salary distribution charts can reveal the overall salary level and differences among industries, cities, or job types [13]. These visual results allow researchers to understand the basic characteristics of the dataset before conducting deeper analysis.

Visualization also plays an important role in presenting changing trends. Line charts and time-series charts are commonly used to analyze data changes over time. In network public opinion analysis, Python can be used to visualize the number of comments, reposts, or keyword occurrences at different time points, thereby showing the rise, peak, and decline of a public topic. In financial data analysis, stock prices, trading volume, and index fluctuations can be displayed through line charts or candlestick charts, helping analysts observe market trends and volatility. In education data analysis, changes in students' scores, learning time, or online platform activity can be visualized to evaluate learning progress and teaching effectiveness.

Another analytical value of visualization lies in its ability to identify abnormal phenomena. Scatter plots, box plots, and heatmaps can help detect unusual values, abnormal relationships, or unexpected patterns in data. In commodity price analysis, an extremely high or low price may indicate a data extraction error or a special market condition. In website traffic analysis, a sudden increase in visits may reflect a marketing event or abnormal automated access. In public opinion monitoring, a sharp rise in negative keywords may indicate a potential crisis. Therefore, visualization supports early warning, problem detection, and decision-making.

Python visualization has broad application value in many fields. In network public opinion analysis, crawler technology can collect comments, news, and social media texts,

while visualization can present keyword frequency, sentiment tendencies, topic changes, and regional distribution. In commodity review analysis, Python can collect user comments from e-commerce platforms and visualize rating distribution, high-frequency words, user satisfaction, and product defects. In recruitment information analysis, crawled job data can be visualized to compare salary levels, skill requirements, education requirements, and regional demand. In financial data analysis, visualization can help display price trends, transaction changes, risk fluctuations, and correlation among indicators. In education data analysis, charts can show learning behavior, performance differences, resource usage, and teaching feedback [8].

The combination of web crawling and data visualization forms a relatively complete data analysis workflow. Web crawling provides the original data source, data cleaning improves data quality, structured processing organizes data into analyzable forms, and visualization transforms analytical results into intuitive charts [3]. This process can be described as "data acquisition---data processing---data analysis---visual expression." In this workflow, each stage supports the next stage. Incomplete data acquisition may weaken the representativeness of analysis, insufficient data cleaning may reduce the accuracy of visualization, and unreasonable visualization design may lead to misleading interpretation. Therefore, visualization should be based on reliable data and appropriate analytical logic.

Python visualization also improves the communication value of data analysis results. For non-technical users, complex numerical tables are often difficult to understand directly. Visual charts can simplify the expression of complex data and help readers quickly grasp key conclusions. A bar chart can directly show which product has the highest sales volume, a heatmap can reveal which variables are strongly correlated, and a word cloud can present the main topics in text data. In research reports, business analysis, teaching, and public decision-making, visualization enhances the readability and persuasiveness of data results.

However, Python-based web crawling and visualization still face several problems in practical applications. Data legality and compliance are major concerns because not all webpage data can be freely collected and used. Some websites restrict automated access through terms of service, robots protocols, login systems, or copyright protection. Therefore, data collection should follow relevant laws, platform rules, and ethical requirements. Anti-crawling restrictions also affect the stability of data acquisition. Websites may change page structures, limit request frequency, block IP addresses, or use verification mechanisms, which increases the difficulty of continuous data collection.

Data quality instability is another important limitation. Webpage data are often affected by inconsistent formats, missing fields, repeated content, and changes in webpage layout. If the crawler fails to update its extraction rules in time, the collected data may contain errors. In addition, chart interpretation may involve subjectivity. The same dataset may lead to different conclusions if different chart types, scales, colors, or grouping methods are used. An inappropriate axis range may exaggerate differences, while excessive chart elements may make the result confusing. Therefore, visualization should emphasize accuracy, clarity, and consistency rather than only visual attractiveness.

Overall, visualization-based data analysis is a key part of Python application in web crawling and data analysis. It helps transform cleaned data into understandable information and supports pattern discovery, trend judgment, anomaly detection, and decision-making. By combining web crawling, data cleaning, structured processing, and visualization, Python provides a complete technical path for data-driven research and practical applications. At the same time, researchers should pay attention to data legality, crawling stability, data quality, and chart interpretation accuracy to ensure that visualization results are both intuitive and reliable [7].

5. Conclusion

This study analyzes the application of Python in web crawling and data visualization. The discussion shows that Python has become an effective technical tool for data

acquisition, data processing, and visual analysis because of its concise syntax, rich third-party libraries, and high development efficiency. In the process of web data collection, Python can use tools such as Requests, BeautifulSoup, Scrapy, and Selenium to obtain data from static and dynamic webpages. These tools improve the efficiency of webpage access, content parsing, data extraction, and large-scale information collection.

After data are collected, data cleaning and structured processing become necessary steps. Raw webpage data usually contain duplicate records, missing values, inconsistent formats, abnormal values, and other noisy information. By using Pandas, NumPy, regular expressions, and related storage tools, Python can transform unstructured or semi-structured webpage content into standardized datasets. This process improves data quality and provides a reliable foundation for subsequent analysis.

Data visualization further enhances the value of Python-based data analysis. Through Matplotlib, Seaborn, Plotly, Pyecharts, and other visualization libraries, Python can present complex data in the form of line charts, bar charts, scatter plots, heatmaps, word clouds, and interactive charts. Visualization helps researchers discover data distribution characteristics, changing trends, correlations, and abnormal phenomena. It also improves the readability and explanatory power of analytical results, making data conclusions easier to understand and apply.

Overall, web crawling solves the problem of data acquisition, data cleaning improves the reliability of data, and data visualization strengthens the intuitive expression of analysis results. The combination of these three stages forms a complete workflow of "data collection---data processing---visual analysis." This workflow has practical value in network public opinion analysis, commodity review analysis, recruitment information analysis, financial data analysis, education data analysis, and other fields.

However, Python applications in web crawling and data visualization still face several limitations. Website structures may change frequently, which can reduce the stability of crawling programs. Anti-crawling mechanisms may restrict automated access and increase the difficulty of data collection. In addition, data compliance, privacy protection, copyright issues, and website usage rules must be considered during data acquisition. At the data processing stage, unstable data quality may affect the accuracy of analysis. At the visualization stage, inappropriate chart selection or subjective interpretation may lead to misleading conclusions.

Future research can further improve Python-based data analysis from several aspects. First, automated data cleaning and intelligent error detection can be introduced to improve data processing efficiency. Second, interactive visualization and dashboard technologies can be used to enhance data exploration and result presentation. Third, machine learning methods can be combined with crawled data to support prediction, classification, clustering, and sentiment analysis. Finally, Python can be integrated with databases, cloud computing, and big data platforms to support larger-scale and more complex data analysis tasks.

In conclusion, Python provides strong technical support for web crawling and data visualization. Its flexibility, extensibility, and rich library ecosystem make it suitable for building an integrated data analysis process. With the continuous development of data acquisition technologies, visualization tools, and intelligent analysis methods, Python will continue to play an important role in data-driven research and practical applications.

References

1. V. Krotov, L. Johnson, and L. Silva, "Tutorial: Legality and ethics of web scraping."
2. A. M. Tanasă, S. V. Oprea, and A. Bâra, "Web scraping and review analytics. extracting insights from commercial data," *International Journal of Data Science Applications*, vol. 5, no. 1, pp. 44-58, 2023.
3. J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, 2007.
4. A. H. Laender, B. A. Ribeiro-Neto, A. S. Da Silva, and J. S. Teixeira, "A brief survey of web data extraction tools," *ACM Sigmod Record*, vol. 31, no. 2, pp. 84-93, 2002.
5. J. VanderPlas, B. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, ... and S. Sievert, "Altair: Interactive statistical visualizations for Python," *Journal of Open Source Software*, vol. 3, no. 32, p. 1057, 2018.

6. T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10-20, 2007.
7. W. McKinney, "Data structures for statistical computing in Python," *Scipy*, vol. 445, no. 1, pp. 51-56, 2010.
8. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, ... and P. Van Mulbregt, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261-272, 2020.
9. M. L. Waskom, "Seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021.
10. R. Diouf, E. N. Sarr, O. Sall, B. Birregah, M. Bousso, and S. N. Mbaye, "Web scraping: state-of-the-art and areas of application," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 6040-6042, Dec. 2019.
11. C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, ... and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357-362, 2020.
12. M. A. Khder, "Web scraping or web crawling: State of art, techniques, approaches and application," *International Journal of Advances in Soft Computing & Its Applications*, vol. 13, no. 3, 2021.
13. R. S. Chaulagain, S. Pandey, S. R. Basnet, and S. Shakya, "Cloud based web scraping for big data applications," in *2017 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 138-143, Nov. 2017.
14. H. Nigam and P. Biswas, "From web scraping to web crawling," in *Applications of Artificial Intelligence and Machine Learning: Select Proceedings of ICAAAIML 2020*, Singapore: Springer Singapore, pp. 97-112, 2021.
15. D. M. Thomas and S. Mathur, "Data analysis by web scraping using Python," in *2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 450-454, Jun. 2019.
16. S. Goel, M. Bansal, A. K. Srivastava, and N. Arora, "Web crawling-based search engine using Python," in *2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pp. 436-438, Jun. 2019.
17. R. Lawson, *Web scraping with Python*. Packt Publishing Ltd, 2015.