I.J. Information Technology and Computer Science, **2024, 3, 72-92**
Published Online on June 8, 2024 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijitcs.2024.03.07

Python Data Analysis and Visualization in Java GUI Applications Through TCP Socket Programming

Bala Dhandayuthapani V.
Department of IT, College of Computing and Information Sciences, University of Technology and Applied Sciences, Shinas campus, Oman
E-mail: bala.veerasamy@utas.edu.om, dhanssoft@gmail.com
ORCID iD: https://orcid.org/0000-0002-8310-0642

Received: 01 February 2024; Revised: 17 March 2024; Accepted: 20 April 2024; Published: 08 June 2024

Abstract: Python is popular in artificial intelligence (AI) and machine learning (ML) due to its versatility, adaptability, rich libraries, and active community. The existing Python interoperability in Java was investigated using socket programming on a non-graphical user interface (GUI). Python's data analysis library modules such as numpy, pandas, and scipy, together with visualization library modules such as Matplotlib and Seaborn, and Scikit-learn for machine-learning, aim to integrate into Java graphical user interface (GUI) applications such as Java applets, Java Swing, and Java FX. The substantial method used in the integration process is TCP socket programming, which makes instruction and data transfers to provide interoperability between Python and Java GUIs. This empirical research integrates Python data analysis and visualization graphs into Java applications and does not require any additional libraries or third-party libraries. The experimentation confirmed the advantages and challenges of this integration with a concrete solution. The intended audience for this research extends to software developers, data analysts, and scientists, recognizing Python's broad applicability to artificial intelligence (AI) and machine learning (ML). The integration of data analysis and visualization and machine-learning functionalities within the Java GUI. It emphasizes the self-sufficiency of the integration process and suggests future research directions, including comparative analysis with Java's native capabilities, interactive data visualization using libraries like Altair, Bokeh, Plotly, and Pygal, performance and security considerations, and no-code and low-code implementations.

Index Terms: Interoperability, Java, Matplotlib, Python, Seaborn, Socket Programming, Visualization.

1. Introduction

Artificial intelligence (AI) suggests that robots can replicate humans in talking, thinking, learning, planning, and understanding, which is known as machine intelligence or computer intelligence. Python is commonly used for AI by data scientists. Python contains built-in mathematical libraries and methods that make it easy to solve mathematical problems and analyze data. Data visualization is critical in current software systems because it allows users to obtain insights from enormous datasets. Data visualization is a method of displaying data graphically or in a pictorial format that aids in the comprehension of large volumes of data. This allows decision-makers to make better decisions and spot current trends and patterns more quickly. Python is a popular data science programming language that has established itself as a dominant language for data analysis and visualization [1,2], with libraries like Matplotlib, Seaborn, Ggplot, Plotly, Geoplotlib, Bokeh, Folium, Altair, and Pygal providing powerful tools for creating visually appealing charts and graphs. These aid in the creation of dynamic, highly changeable plots. Python offers more built-in libraries and a large user base. Python, when combined with add-ons, provides everything required for accurate, appealing, and intelligible visualizations.

Matplotlib [3] is a simple Python data visualization package based on NumPy, sklearn and pandas. It is a low-level module that offers flexibility at the expense of writing additional code. It would be the primary data visualization library to master while employed in data science with Python. Seaborn [4] is a high-level library built on top of Matplotlib, thus it can make use of Matplotlib functions and classes. This collection includes basic styles and color palettes for making a plot more visually appealing.

Ggplot [5] is a Python operation of the Grammar of Graphics that is based on the interface of the R tool ggplot2. It provides statistical graphics mapping data to the aesthetic properties of geometric objects. Faceting allows you to create

the same visualization for multiple subsets of the dataset. Plotly [6] hover feature allows us to find outliers or abnormalities in a huge number of data points. It enables us to endlessly customize our graphs, making our plots more meaningful and understandable by others. Geoplotlib [7] is a great Python data visualization library for plotting geographical data and making maps. It enables the creation of geographical maps, with more map formats accessible such as dot-density maps, choropleths, and symbol maps. It is a simple yet effective API for displaying OpenStreetMap tiles.

Bokeh [8] is a Python-based tool that creates interactive, web-ready plots, provides streaming and concurrent data and offers three interfaces: quick, middle, and low. It supports bar plots, box plots, and histograms, and needs developers to describe each component of the chart. Bokeh is native to Python and supports JSON objects, HTML documents, and streaming data. Folium [9] streamlines the time-consuming process of creating Google Maps, adding markers, and showing directions for developers. You can focus on entering and analyzing the data in Folium by simply loading libraries, designing a map, and using those tools. Altair [10] is a declarative visualization library based on Vega, like Seaborn but lacking plotly or bokeh appearance and assembly difficulties, allowing for quick and easy creation of interactive graphs and charts. Pygal [11] like Bokeh and Plotly, provides interactive web-based plots with the ability to output charts as SVGs. However, for large datasets, it may struggle. Respectively, each chart type is packed into a method, offering easy customization.

Java provides libraries for creating interactive and informative data visualization libraries like JavaFX, JFreeChart, and other third-party libraries. JavaFX graphical charts [12] are available in the javafx.scene.chart package of the JavaFX SDK and contain area charts, bar charts, bubble charts, line charts, pie charts, scatter charts, and styled charts with CSS. JavaFX includes built-in support for animations, enabling dynamic visualizations. JavaFX seamlessly integrates with the Java programming language, allowing for easy data manipulation and visualization. JFreeChart [13] supports image files in PNG and JPEG formats, as well as vector graphics files in PDF, EPS, and SVG formats, in Swing and JavaFX application components. It supports a variety of chart styles, such as pie charts, bar charts, time series charts, and others. It has a wide range of customization options for fine-tuning graphic aspects.

1.1. Statement of Problem

Interoperability is required in modern software development to mix the features of multiple programming languages into a single application. Java is a versatile cross-platform programming language, whereas Python is well renowned for its sophisticated data visualization features. The existing Python interoperability in Java was investigated using socket programming on a non-graphical user interface (GUI), which identified the gap to fill in the present research. It is challenging to incorporate Python's data visualization tools seamlessly into a Java GUI context. The study questions: How can Python's data visualization or charts (e.g., Matplotlib and Seaborn) be effortlessly incorporated into Java GUI applications? What are the essential factors to consider when picking the proper Python visualization or charts (e.g., Matplotlib and Seaborn) based on specific visualization requirements in a Java GUI application? What are the advantages and disadvantages of using Python's data visualization capabilities in Java GUI applications? Therefore, the primary objectives of this research are to discover Python's data visualization or charts and their capabilities. It aimed to investigate methods for incorporating Python data visualization into Java GUI applications and intended to evaluate the benefits and challenges of this integration. And lastly, it targeted to provide practical implementation to demonstrate the effectiveness of the integration.

1.2. Significance

This research aims to combine Python and Java's strengths in data visualization, resulting in more powerful and versatile applications. Python's data visualization libraries can be integrated into Java GUI applications, providing a vast ecosystem of tools and techniques for more sophisticated data representations. This integration is used in various fields like AI in data science, finance, engineering, and healthcare. Python libraries can also enhance the user experience, save development time and resources, and improve interoperability with existing systems. This integration can have a significant impact in academic and research settings, offering a competitive edge in industries where data presentation is critical. This research also fosters collaboration and knowledge sharing between Python and Java developers. Therefore, integrating Python's visualization or charts allows developers to take advantage of the rich ecosystem and expertise in data visualization. It enables the utilization of Python's data visualization strengths in Java GUI applications.

1.3. Advantages and Challenges

The research concentrated on incorporating Python data analysis and visualization, or charts, into Java GUI applications. The target audience for the Java GUI application may include developers, analysts, or end-users. The advantages include that there are no compatibility issues or version dependencies since there is no third-party library or any extra library needed when integrating Python data visualization with Java. It is a potential benefit for Java developer to incorporate Python features without any extra cost, time, or effort. It will identify industries where integration can provide value, assess its impact on the user experience, and provide support resources. Java GUI applications enhance the presentation and interpretation of complex datasets, leading to more effective decision-making and improved user experiences compared to applications solely reliant on Java-based visualization tools. Integrating Python data visualization into Java may lead to improvements in usability and the user experience. It has broad practical

applications in various industries. It fosters collaboration and knowledge sharing between the Python and Java developer communities. The integrated solution is platform-independent, ensuring compatibility across different operating systems, and the Python integration does not significantly alter the existing Java codebase. The research challenges focused on specific Python data visualization libraries like Matplotlib and Seaborn, potentially excluding other options and introducing performance overheads and security concerns. It may not cover all scenarios or edge cases, assume a specific platform, and not delve into industry specific requirements.

Section 2 identifies and covers the related literature review works. In Section 3, the methodology is introduced: the research design and implementation code for the Python server and Java client with the common code used in all Java applets, Java Swing, and Java FX, which is suitable for different execution environment paradigms, are demonstrated on a local server with a local client. Section 4 examines, evaluates, and visualizes the interoperability of the Python server and Java client with Java applets, Java Swing, and Java FX. Section 5 provides a simple conclusion. This is empirical research that demonstrates Python code execution on Java clients using Java applets, Java Swing, and Java FX. However, Python should include all the required libraries before utilizing them in Java. In particular, the Matplotlib and Seaborn libraries were utilized to create data visualizations in this study. It also specified the scope of the completed research as well as future directions.

2. Related Works

The R package jsr223 [14] is a complex combination of five Java programming languages: Groovy, JavaScript, JRuby, Jython, and Kotlin. It allows Java inside R by embedding codes. It relies on the Java Scripting API and features data exchange capabilities. Overall, jsr223 simplifies Java or jsr223-supported language solutions. Doing better data visualization [15] is a tutorial that aims to help social scientists communicate data patterns effectively to others, assuming basic statistical and visualization knowledge. It discusses design philosophies, color decisions, and code examples in R for visualizing trends, scopes, and relations between variables. The tutorial is manageable to put together for R and ggplot2 users. The data science toolkit (DST) [16] is a Python library designed to enhance code abstraction and productivity. It is used in research into artificial intelligence for agricultural management practices. DST uses an object-oriented approach, including Data Frame and Model classes. The overview of PM4Py [17], is a Python library offering numerous tools for the mining process, its integration with other libraries, and its significant impact on the academy, industry, and the open-source community.

Big Data has transformed healthcare by enhancing data visualization techniques, enhancing understanding, and reducing treatment time. The COVID-19 pandemic underscored the importance of data visualization in identifying patterns and presenting data effectively [18]. Python's popularity in data science has led to an increase in free libraries for data mining and big data analysis. A comparison of free Python libraries [19] over twenty libraries, divided them into six groups: core libraries, data preparation, visualization, machine learning, deep learning, and big data. It suggested Pandas for data preparation, Matplotlib for visualization, scikit-learn for machine learning, TensorFlow for deep learning, and Hadoop Streaming for big data. A new system [20] allowed users to achieve interactive questions, making real-time evidence in multiple formats instantaneously. The system used a map-reduced paradigm model for exploratory instruction and diagnostic tasks for deep learning processes, attaining simplification and extensibility of primitives with a different approach than current systems.

Programming languages have unusually difficult syntaxes, which make learning difficult and affect the testability of the program. The literature offered numerical proof of the impact of sophisticated programming language syntax and looked at how programming grammar affected it and how difficult the source code was. A total of 298 algorithms were chosen [21], and the cyclamate complexity matrix was used to examine how they were implemented in Java and Python. The findings indicated that Python syntax is a little more difficult than Java, making Python code more systematic than Java writing. Unsupervised translation [22] of programming languages proposed a fully unsupervised neural trans compiler to interpret functions between C++, Java, and Python with more correctness. The model, trained on open-source GitHub project code, requires no expertise in the source. It overtakes rule-based profitable standards by an important margin. The method requires no expertise in the source and can be widespread in other programming. The model also outperforms rule-based commercial baselines.

Visualization is the graphical representation of data using pictorial design [23], aiming to make it easy to understand and present. It can be univariate or multivariate and is commonly used in management research. As data science evolves, new techniques for visualizing quantitative and qualitative data are sought. Data analysis using Python discussed data analysis processes like cleaning, transforming, and modelling, focusing on exploratory data analysis [24] using the "World Happiness Report 2021" dataset and illustrating graphical analysis using Python libraries and functions. Researchers have gained experience in data visualization technology [25], playing a crucial role in technical findings, medical diagnosis, business decision-making, and engineering applications. Matplotlib, a Python library, is used for data visualization, and the article discusses its use and its impact on various fields. The article [26] offers a variety of visual presentation techniques, such as text data visualization, network visualization, and spatial data visualization which include common data visualization techniques. It discusses data visualization tools, such as SmartBI, D3, Google Chart API, and RapidMiner. A comparative analysis of the data visualization [27] libraries Matplotlib and Seaborn in Python examined and evaluated Python Programming Languages' data science libraries, Matplotlib and

Seaborn, to identify strengths and weaknesses in data visualization and computational modelling techniques, focusing on univariate and multivariate plotting patterns.

Python data visualization libraries [28], including Matplotlib, Seaborn, Plotly, Bokeh, Altair, and Ggplot, evaluated the performance, ease of use, flexibility, and speed of library functionality separately. The findings indicate that each library has distinct strengths and weaknesses, making it challenging to select the best one for all visualization purposes. The most popular and extensively used libraries are Matplotlib, Seaborn, and Plotly, each having its unique set of strengths. Bokeh, Altair, and Ggplot are less popular, but they each have their own set of features and usefulness. The findings can assist data analysts and scientists in selecting the best library for their specific visualization requirements. Data visualization is crucial for understanding and interacting with data, aiding in conclusions, and pre-processing machine learning and artificial intelligence algorithms. However, most users struggle with complex interfaces or a lack of programming knowledge. It presented [29] an easy-to-use website, developed using Python, Plotly, Flask, HTML, CSS, and JavaScript, aimed at beginners with limited programming knowledge and difficulty dealing with datasets. Companies need to track sales and advancement, but estimating sales is challenging due to substantial amounts of data. To address this [30], a web application has been proposed to analyze and visualize data using graphs, tabulations, and charts and allow for a more accurate understanding of company sales and sales by different salespeople.

Python-based teaching system for data visualization [31] focused on hybrid teaching of data. It introduced numerical, textual, and hybrid data visualization methods and explored their effectiveness in interactive presentations. It confirmed the success of data visualization in blended teaching and suggested optimizing its application strategy for data-driven precision teaching. Engine [32] data analysis and visualization aimed to create a Python script for data analysis and visualization tasks, including generating visualizations, extracting labels, managing incompatible files, and computing performance metrics like pressure-volume diagram area, engine power output, and fuel consumption, and to create an automated tool for data analysis. Food systems are essential for feeding the global population, but there is an absence of interoperability through database experiments. USDA Food Data Central assessed databases for interoperability, identifying existing crosswalks and visualizations [33]. This benefits the databases in terms of extension, integration, coordination, joining data, and linking administrative walls.

Interoperability of programming languages [34,35] allows two or more languages to interact in a system, often involving passing messages and data between different languages. This is crucial for client-server architectures and distributed computing systems. Tools like virtual machines and mark-up languages address cross-language communication, addressing different problem domains and their strengths in various systems. Python Interoperability in Java [34] examined the significance of communication interoperability between Java and Python via socket programming. It demonstrated how to integrate Python libraries with Java without the use of any additional libraries or third-party libraries. This technique saves money by reducing development time, maintenance, usability, and system integration expenses. We looked at Python modules for data exploration, statistical analysis, and machine learning with linear regression. This research paper improves and incorporates data analysis and visualization capabilities, as well as an artificial intelligence (AI) and machine learning (ML) approach within Java's graphical user interface (GUI), by emphasizing Python's data visualization capabilities to strengthen Java GUI applications. Matplotlib is efficient at generating charts, and it is well-suited to a variety of data visualization tasks. Seaborn, based on Matplotlib, provides a high-level interface for generating interesting visualizations. Both the library charts and plots are examined in Java GUI applications.

3. Methodology

Python interoperability in Java can be achieved by TCP socket communication in several execution environments [34]. We can implement any of the execution environments indicated in the use cases. This study was done and evaluated on a local server and client. Python and Java are two of the most popular programming languages, and there are several scenarios in which merging Python and Java code may be appropriate. TCP socket programming, which allows network connections between applications operating on different machines, is one method of accomplishing this with Python as a server and Java as a client. TCP socket programming is lightweight and adaptable, making it simple to create and manage small applications.

3.1. Research Design

Python data visualization in Java can be achieved through Python interoperability in Java through TCP socket programming [34], establishing a socket connection between Python and Java. This model enhanced the data analysis with multiple lines of commands and integrated the visualization interchange allowed between Python and Java. Python functions as a server, performing data analysis and visualization and returning results to Java clients. During this process, Java initially sends the request to the Python server, where the Python server executes the commands to prepare the chart, which is then stored in the “plot.jpg” file on the server side. This “plot.jpg” file was transferred to Java, where Java received it and stored it in the local file system. The plot.jpg file shows using the Java swing frame. The plot.jpg file can load in different Java GUI applications, such as Java applet, Java swing, and JavaFX. The applet can load either in the applet viewer or a web browser using the <applet> html tag. However, recent browsers have

stopped working with web apps. Despite that, CheerpJ applet runner is used to run web apps. The following Figure 1 shows a conceptual view of the execution model for Python data visualization:

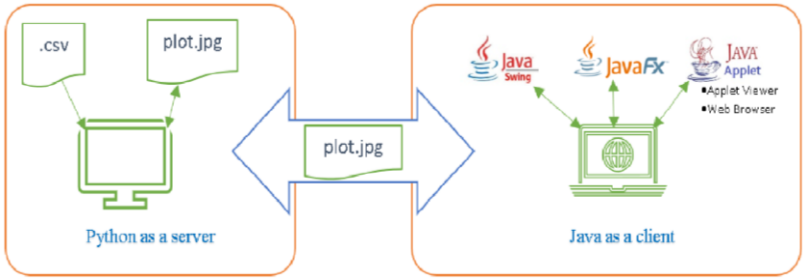
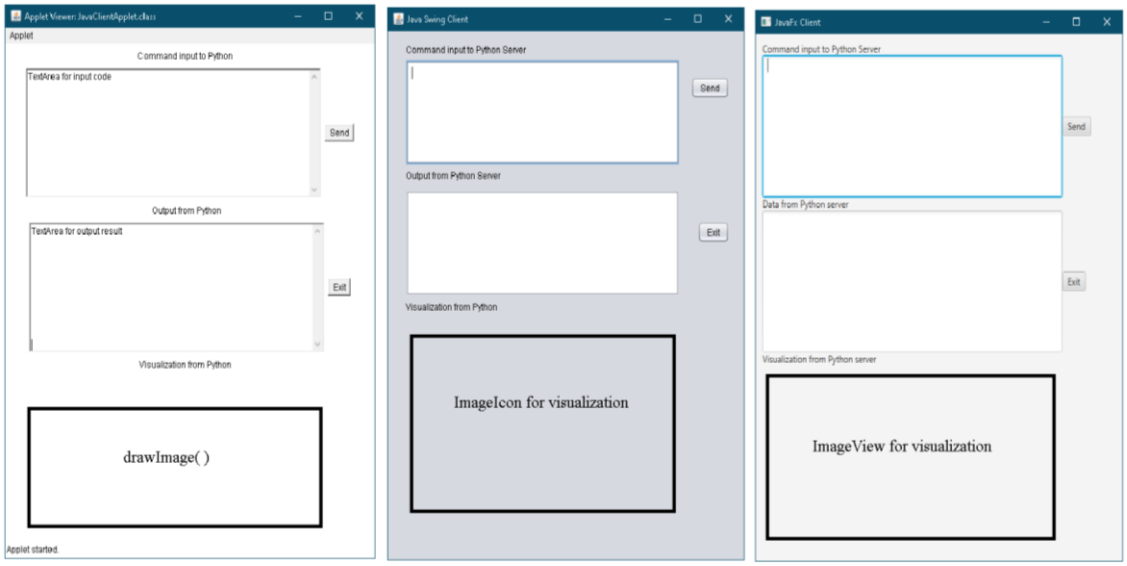


Fig.1. Conceptual view of the execution model



2a. Java Applet 2b. Java Swing 2c. Java FX

Fig.2. Research design of a java client

Figure 2 shows the research design of a Java client for Java applet, Java swing, and Java FX. The design focused on three parts: input command, output result, and visualization. The input command and the output command are used in the Text Area component, and the visualization of applets is done by drawing image graphics; swing is used by the ImageIcon component; and Java FX is used by the ImageView component. The button labelled send is used to collect a single or list of instructions or commands from the input Text Area component, then send the commands to the Python server and receive outputs placed on the output result Text Area.

3.2. Algorithm

The Python server initiates and communicates over a TCP connection using IPv4 address. The server is bound to the host's name and any unreserved port number, and it listens for incoming connections. Once a connection is established, the server enters a loop to manage client requests. It receives messages from the client, interprets them as commands, and processes them accordingly. If the command is an exit request or a quit request, the server responds appropriately and terminates the connection. If the command indicates a CSV file, the server attempts to read it, creating a data frame and notifying the client. If the command pertains to generating a chart, the server creates an image, encodes it, and sends it back to the client. For other commands, they are interpreted as Python code and executed, and the output is sent to the client. If any unexpected errors occur during this process, the server notifies the client of the error, closes the connection, and halts the program. This program effectively serves as a versatile interface for clients to interact with a Python server, accommodating distinct types of requests and providing appropriate responses. The algorithm for the provided Python program is here:

Algorithm 1: Python server

- Initialize Server:
    - Create a socket named s using IPv4 addressing and TCP connection.
    - Bind the socket to the current machine's hostname and port number.

- Start listening for incoming connections, allowing any number of connections.

- Main Server Loop:
  - Continuously run the following steps:
  - Accept an incoming connection and get the address of the client.
  - Receive a message from the client with a minimum size of 24576 bytes.
  - Decode the message from bytes to a string, assuming UTF-8 encoding, and store it as code.

- Command Processing:
  - Check if the received command code is an exit request or quit request:
    - If true, print a message indicating the request, exit the program with a message, close the client connection, and end this iteration.
  - Check if the command ends with ".csv":
    - If true, attempt to read a CSV file with the same name as the command, create a Data Frame (df), print a loading message, and send a confirmation message back to the client.
  - Check if the command starts with "chart":
    - Check if a file named "plot.jpg" exists, and if it does, it is to be deleted.
    - If true, generate a plot and save it as "plot.jpg". Open the file, read the image data, and send it back to the client.
  - If none of the above conditions are met, assume it is a Python command and execute it:
    - Use exec to run the command and capture the output in out.
    - Convert the output to a string and store it in the output.

- Sending Output Back to Client:
  - Check if the output is numeric:
    - If true, encode it into bytes and send it back to the client.
  - Check if the output is a floating-point number:
    - If true, encode it into bytes and send it back to the client.
  - If none of the above conditions are met, assume it is a string and send it back to the client.

- Exception Handling:
  - In case of any unexpected errors:
    - Print an error message with the details of the exception.
    - Send the error message back to the client.
    - Close the client connection.
    - Raise the exception to halt the program.

The Java client retrieves user input from a "command" element, initializes input and output streams, and splits multi-line commands into an array. It sends the command to the server, checks if it is an exit or quit request, closes the connection, and updates the UI element "result." The program then processes server responses, converts data to strings and numeric data, and displays the output. If a runtime error message is detected, the user is notified and advised to reconnect. If the input is empty, the program prompts the user to enter a command. The following algorithm two outlines the steps performed by a Java program to manage user commands, communicate with a Python server, process data, and display the results to the user. It provides a versatile interface for interacting with Python code and managing distinct types of responses and errors.

Algorithm 2: Java Clients

- User Input:
  - Retrieve user input from a UI element named command. Trim any leading or trailing spaces.
  - Reset the command field to an empty string.
- Split Multi-line Commands:
  - Split the user input into an array of individual commands based on newline characters.
- Check Input Length:
  - If the user input is not empty:
    - Establish a socket connection to the server using the IP address and the server's port number.
    - Initialize input and output streams for communication.
- Iterate Over Commands:
  - For each code in the array of commands:
    - Build an output string output with a prompt ">>>" and the command.
    - Send the command to the server by writing its bytes to the output stream.
    - Check if the command is an exit request or quit request:
    - If true, close the connection, append a disconnection message to the output, and update the UI element result.
- Process Server Responses:
  - For non-exit commands:
    - Receive data from the server as bytes.
    - Convert this data to strings and numeric.
- Handle "chart" Command:
  - If the command is "chart":
    - Check if a file named "plot.jpg" exists, and if it does, it deletes.
    - Save the received bytes as an image named "plot.jpg".
    - Open the image file, create an image object, and show the image.
- Update UI with Output:
  - If the data is numeric, append it to the output. Otherwise, append the text data.
  - If there is a runtime error message, notify the user and advise reconnection.
- Manage Empty Input:
  - If the user input is empty, prompt them to enter a command.
- Exception Handling:
  - In case of any unexpected errors:
    - Print an error message with details.
    - Append the error message to the output.
    - Update the UI element results with the updated output.

3.3. Implementations

The Java client side requires the importation of necessary packages, including the socket package for TCP communication and the sys and os packages for program execution. The server establishes a connection using the socket() function and binds up the host using any unreserved port number. The program reads data from the socket and receives input code from the Java client using msg=clt.recv(24576). The input code is decoded in UTF-8 format. Before execution, different conditions are applied, such as exit() or quit(), os.system('rm -rf *') or loading a data frame object in Pandas. If the output is numerical, bytes, floating value, or encoded, the output is sent to the Java client. All program code must be written inside the try block to catch runtime exceptions.

Python's server-side code uses the server's host and port number, while Java clients should use the same. The server listens on a TCP socket, with client connections adjustable based on requirements.

Program code 1: Python as a server

```
import socket,math,datetime,random,sys,os
import numpy as np
import pandas as pd
from scipy import constants
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
def is_float(a_string):
    try:
        float(a_string)
        return True
    except ValueError:
        return False
try:
    s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((socket.gethostname() ,1234))
    print(socket.gethostname())
    s.listen(5)
    while True:
        clt,adr=s.accept()
        msg=clt.recv(24576)
        xcode=msg.decode("utf-8")
        print(xcode)
        if(xcode=="exit()" or xcode=="quit()"):
            print("output-0: ",xcode)
            sys.exit("You have stopped")
            clt.close()
        elif(xcode.endswith(".csv")):
            df = pd.read_csv(xcode)
            print("output-00: ",xcode," loaded")
            clt.send("pandas data frame object with 'df' loaded.".encode())
        elif(xcode.startswith("chart")):
            if os.path.exists("plot.jpg"):
                os.remove("plot.jpg")
            else:
                print("The file does not exist")
            plt.savefig("plot.jpg")

            with open('plot.jpg', 'rb') as file:
                image_data = file.read()
                clt.sendall(image_data)
        else:
            exec(f"out= "+xcode) #, globlsparam,  localsparam)
            output=str(out)

            if(output.isnumeric()):
                clt.send(bytes(out))
                print("output-1: ",out)
            elif (is_float(output)):
```

```
            clt.send(output.encode())
            print("output-1: ",output)
        else:
            clt.send(output.encode())
            print("output-2: ",output)
except Exception as err:
    print(f"Unexpected {err=}")
    error=str('Runtime Error: ') + str(err)
    clt.send(error.encode())
    clt.close()
    raise
```

The provided Python program 1 sets up a server using sockets to listen on port 1234. It binds the server to the local hostname. Upon receiving a connection request, it accepts it and waits for a message. The message is decoded from bytes to a string, representing a command from a client. The program then processes the command based on certain conditions. If the command is "exit()" or "quit()", the program prints a message indicating that the server is stopped and exits the program. If the command ends with ".csv", it attempts to read a CSV file with the same name as the command and sends a confirmation message back to the client. If the command starts with "chart", the program generates a plot, saves it as "plot.jpg", and sends the image data back to the client. Otherwise, the program executes the command using exec, capturing the output. It then checks if the output is numerical or a floating-point number and sends it back to the client accordingly. In case of any unexpected errors, the program prints an error message and sends it back to the client prefixed with "Runtime Error:". The connection was then closed, and the error was raised to halt the program. Overall, this program serves as a server that interprets commands from a client processes them, and returns results or images based on the nature of the command. Program code 2 shows that the Java implementation connects to the server's host and port number.

Program code 2: Common code for Java as a client using Java Applet, Java Swing, and Java Fx

```
String output="";
      try{
         String sendData = command.getText().trim();
         command.setText("");
         String[] code = sendData.split("\n");
         if (sendData.length() > 0) {
            Socket s=null;
            DataInputStream in=null;
            DataOutputStream out=null;
          for (String xcode : code ) {
            s=new Socket("10.153.70.91",1234);
            in=new DataInputStream(s.getInputStream());
            out=new DataOutputStream(s.getOutputStream());
            output= ">>> "+ xcode + "\n"; //+ result.getText();
              out.write(xcode.trim().getBytes());
              if(xcode.equals("exit()") || xcode.equals("quit()") ) {
                  s.close();
                  output += "Disconnected"+ "\n" + result.getText();
                  result.setText(output);
               }
               else{ //receive from python
                  byte[] bdata=new byte[24576];
                  String sdata="",edata="";
                  if(xcode.equals("chart")){
                       File f=new File("plot.jpg");
                       if(f.exists()) f.delete();
                       FileOutputStream fileOutputStream = new FileOutputStream("plot.jpg");
                       fileOutputStream.write(bdata, 0, in.read(bdata,0,bdata.length));
                       fileOutputStream.close();
                       fileOutputStream.flush();
                    //….
                          // Follow the program codes 3, 4, and 5.
                   //….
                  }
                  else{
```

```
                sdata=String.valueOf(in.read(bdata,0,bdata.length)); //numbers
                edata = new String(bdata);                    //text
                }
            if(edata.trim().equals("")){
                output += ">>> "+ sdata + "\n" + result.getText();
                result.setText(output); // number data
             }
             else{
                output += ">>> "+ edata + "\n" + result.getText() ;
                result.setText(output);  //text data
                if(edata.contains("Runtime Error: ")) {
                output = "Connection Terminated, reconnect again...\n" + output;
                result.setText(output); }
             }
             }
          }
        }
        else {
           output += "Type the command to send to the Python server..." + "\n" + result.getText();
           result.setText(output);
     }
  }catch(Exception e1){ output += e1.toString() + "\n" + result.getText(); result.setText(output);}
```

The given Java program 2 establishes a connection with a Python server and allows the user to interact with it. It starts by retrieving a command string from a GUI element called command, which is a text area input field. The program then splits this string into separate commands based on newline characters. For each command, the program creates a socket connection to the server at the IP address and port number. Next, it sets up input and output streams to communicate with the server. If the command is not empty, it sends the command to the server and appends it to the output string along with the command prompt (">>>"). If the command is either "exit()" or "quit()", it closes the socket, indicating a disconnect, and updates the output string accordingly.

For the "chart" command, the program receives an image from the server, saves it as "plot.jpg", and displays it using a GUI element of Java applets, Java swing and Java FX. It also ensures that the image maintains its original aspect ratio. For other commands, the program receives data from the server, differentiating between numerical data stored in sdata and text data stored in edata. It then updates the output string with the received data. If an exception occurs during the process, the program catches it and appends an error message along with the exception details to the output string. Finally, the program updates the GUI element results with the modified output string. If the command input is empty, the program prompts the user to enter a command. This program effectively serves as a Java client for interacting with a Python server, providing a user-friendly interface for sending commands and receiving responses.

Program code 3: Show Plot on the Java Applet Client

```
if(xcode.equals("chart")){
//….
    // Follow the program code 2 to store plot.jpg file using FileOutputStream
//….
//Setting the image view parameters
String dir = "file:" + System.getProperty("user.dir") + "/plot.jpg";  //prepare URL
URL url=new URL(dir);                                                  //create the URL object
picture=getImage(url);                                                 //get the image using URL
repaint();                                                  // invoke the repaint() method to draw the image.
}
public void paint(Graphics g) {
    g.drawImage(picture, 30,420, this);                    // draw an image on the given x and y axes.
}
```

The program code 3 segment works for a Java applet to show the plots. It constructs a URL for the saved image file, retrieves the image content from the URL, and updates the display in a graphical user interface. This process processes and displays an image received from the server when the command "chart" is issued. The Java program defines a method called "paint" that overrides the standard "paint" method. It uses the Graphics object "g" as a parameter to draw an image, represented by the variable "picture", onto a graphical component. The drawImage() method is called on the Graphics object, specifying the image's position and coordinates (30, 420). The component itself acts as an image observer, monitoring image loading.

Program code 4: Show Plot on the Java Swing Client

```
if(xcode.equals("chart")){
//….
    // Follow the program code 2 to store plot.jpg file using FileOutputStream
//….
ImageIcon icon = new ImageIcon("plot.jpg");      //image icon open the plot.jpg
imageShow.setIcon(icon);                         // Label component: set the icon as a picture.
}
```

The program code 4 segment works for Java Swing to show plots. The code sets image view parameters and creates an ImageIcon component for easy display in a Swing GUI. The imageShow object is a label, which will set the image icon for the plot.jpg picture.

Program code 5: Show Plot on the Java FX Client

```
if(xcode.equals("chart")){
//….
    // Follow the program code 2 to store plot.jpg file using FileOutputStream
//….
InputStream stream = new FileInputStream("plot.jpg"); //creating the image object
Image image = new Image(stream);                      //creating the image view
imageView.setImage(image);                            //setting image to the image view.
imageView.setPreserveRatio(true);                     //setting the image view parameters.
}
```

The program code 5 segment works for Java FX; a new input stream is created to read image data from the file. An Image object is created, and the image is displayed in the ImageView component. The setPreserveRatio method preserves the image's aspect ratio.

4. Result and Discussion

The technique of presenting data in the form of graphs or charts is known as data visualization. It facilitates the understanding of large and complex amounts of data. It enables decision-makers to make more efficient decisions while also simply discovering current trends and patterns. It is used for high-level data analysis, such as machine learning and exploratory data analysis (EDA). Matplotlib is a Python low-level library for data visualization, based on NumPy arrays. It offers various plot types, like line charts, bar charts, and histograms, and is user-friendly. Pyplot is a MATLAB-style module with a MATLAB-style interface that supports various plot types. Seaborn is a statistical graphics toolkit for data visualization, integrating with Panda's data structures and offering dataset-oriented APIs for a better understanding of the dataset. Both libraries offer flexibility and ease of use. This research used the sample dataset "data.csv" file, which consists of forty-seven rows of records with three columns such as area, rooms, and price, in which each column consists of 1 Not a Number (NaN) value that is used to represent undefined, unpresentable, or empty values.

Table 1 shows the program code that loads the "data.csv" file into the Panda data frame, which is used to create a data visualization of all the plots or figures given in this section. The title of the plot is "Linear graph," and the plot's x value as df['rooms'] and y value as df['price'] are assigned to prepare the visualization. The Python server produces output based on the different instructions executed. They are output-0, output-00, output-1, and output-2. The output-0 is produced when an exit() or quit() instruction is executed. The output-00 is produced when the ".csv" file is loaded into Panda's data frame. Output-1 is produced when a numerical output is generated. Output-2 is produced when a string output is generated. The received output from Python is shown in the text area provided for output from Python.

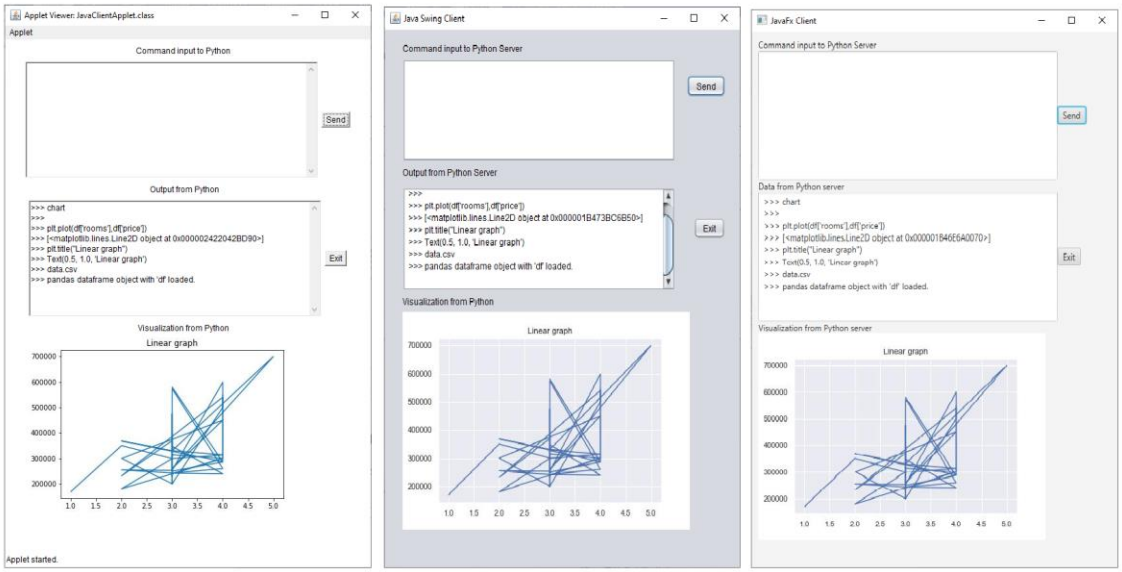
Table 1. Program code for line chart with matplotlib

Java client (Applet, Swing, Java FX)	Python Server
data.csv plt.title("Linear graph") plt.plot(df['rooms'],df['price']) chart	output-00: data.csv loaded output-2: Text(0.5, 1.0, 'Linear graph') output-2: [<matplotlib.lines.Line2D object at 0x000001B470EA1CA0>]

Once the "chart" instruction is entered, the visualization plot name "plot.jpg" is saved on the Python server and sent to the client as a byte stream, which is stored on the client side. Further, "plot.jpg" is loaded into the Java applet (Fig. 3a. Java Applet), Java Swing (Fig. 3b. Java Swing), and Java FX (Fig. 3c. Java FX), where it is shown in the visualization from the Python area in the user interfaces, which is shown in Figure 3 line chart with Matplotlib. It shows the rooms on the x-axis and the price of the rooms on the y-axis.

Since the "data.csv" file is already loaded into the Panda data frame, there is no need to repeat and enter the data.csv as an instruction. And it can continue with the other instructions. Table 2 shows the program code used to enter the bar chart. The bar chart x-axis represents df['rooms'] and the y-axis represents df['price'] respectively. The title of the plot is "House Price", the x-axis label is "Rooms" and the y-axis label is "Price". The output-2 was produced on the

Python server since a string output was generated.



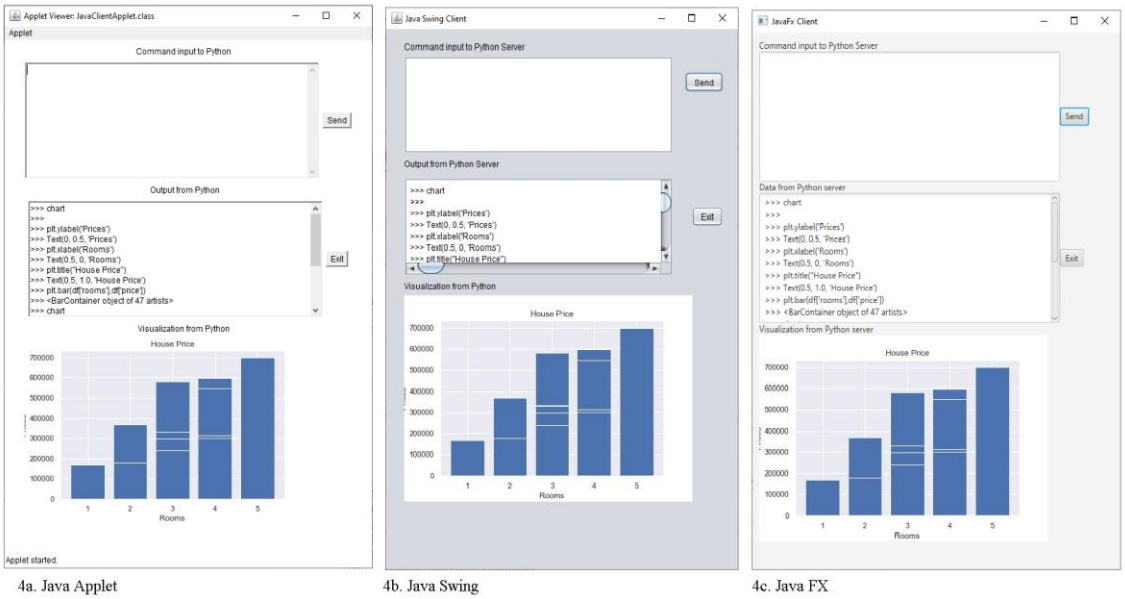
3a. Java Applet 3b. Java Swing 3c. Java FX

Fig.3. Line chart with matplotlib

Table 2. Program code for bar chart with matplotlib

Java client (Applet, Swing, Java FX)	Python Server
plt.bar(df['rooms'],df['price']) plt.title("House Price") plt.xlabel('Rooms') plt.ylabel('Prices') chart	output-2: <BarContainer object of 47 artists> output-2: Text(0.5, 1.0, 'House Price') output-2: Text(0.5, 0, 'Rooms') output-2: Text(0, 0.5, 'Prices')

As earlier, once the “chart” instruction is entered, the visualization plot name "plot.jpg" is saved on the Python server and sent to the client as a byte stream, which is stored on the client side. Further, "plot.jpg" is loaded into the Java applet (Fig. 4a. Java Applet), Java Swing (Fig. 4b. Java Swing), and Java FX (Fig. 4c. Java FX), where it is shown in the visualization from the Python area in the user interface, which is shown in Figure 4 bar chart with Matplotlib. It shows the rooms on the x-axis and the price of the rooms on the y-axis. The chart clearly shows that the highest price falls on a 5-room house, then 4 rooms, and later 3 rooms. We can understand that as rooms increase areas are also increased and prices are proportional.



4a. Java Applet 4b. Java Swing 4c. Java FX

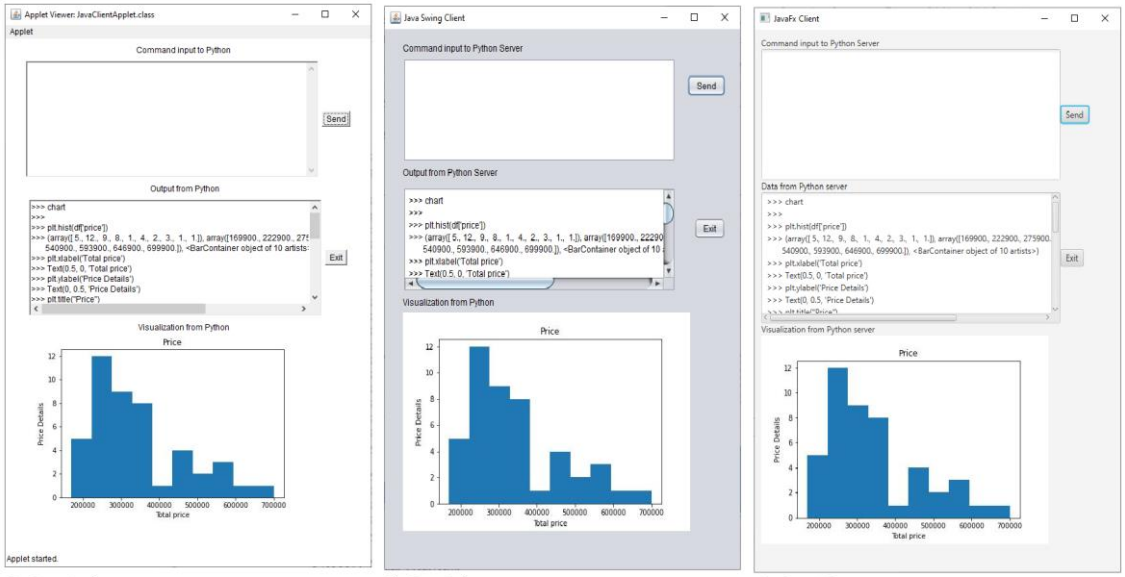
Fig.4. Bar chart with matplotlib

Table 3 shows the program code used to enter the histogram chart. The histogram chart x-axis label represents the total prices, and the y-axis label represents price details with the number of occurrences of each price, respectively. The title of the plot is “Price,” the histogram generated for the df[‘price’] variable. The output-2 was produced on the Python server since a string output was generated.

Table 3. Program code for Histogram Chart with Matplotlib

Java client (Applet, Swing, Java FX)	Python Server
plt.title("Price") plt.ylabel('Price Details') plt.xlabel('Total price') plt.hist(df['price']) chart	output-2: Text(0.5, 1.0, 'Price') output-2: Text(0, 0.5, 'Price Details') output-2: Text(0.5, 0, 'Total price') output-2: (array([5., 12., 9., 8., 1., 4., 2., 3., 1., 1.]), array([169900., 222900., 275900., 328900., 381900., 434900., 487900.,540900., 593900., 646900., 699900.]), <BarContainer object of 10 artists>)

As before, once the “chart” instruction was entered, the visualization plot name "plot.jpg" was saved on the Python server and sent to the client as a byte stream, which was stored on the client side. Further, "plot.jpg" was loaded in the Java applet (Fig. 5a. Java Applet), Java Swing (Fig. 5b. Java Swing), and Java FX (Fig. 5c. Java FX), which is shown in the visualization from the Python area in the user interface, which is shown in Figure 5 histogram chart with Matplotlib. It shows the total price on the x-axis and the price details on the y-axis. The chart clearly shows that most of the prices fall within the range of 20,000 to 30,000.



5a. Java Applet 5b. Java Swing 5c. Java FX

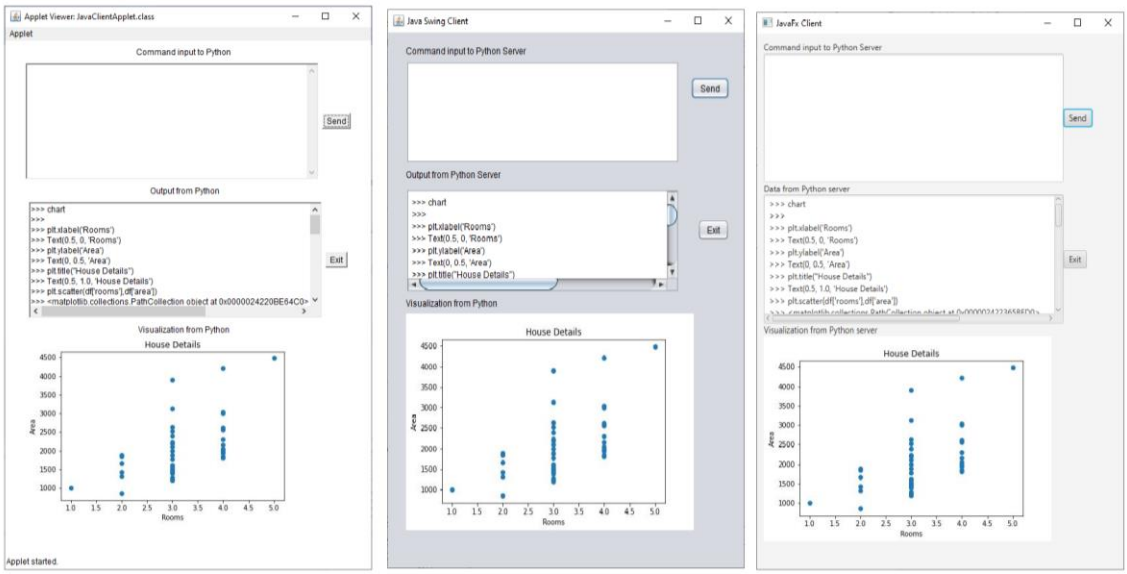
Fig.5. Histogram chart with matplotlib

Table 4 shows the program code used to enter the scatter chart. The scatter chart x-axis label represents the rooms, and the y-axis label represents the area, respectively. The title of the plot is “House Details" and the scatter generated for df['price’] and df['area'] variables is represented by the x and y axes, respectively. The output-2 was produced on the Python server since a string output was generated.

Table 4. Program code for scatter plot with matplotlib

Java client (Applet, Swing, Java FX)	Python Server
plt.scatter(df['rooms'],df['area']) plt.title("House Details") plt.ylabel('Area') plt.xlabel('Rooms') chart	output-2: <matplotlib.collections.PathCollection object at 0x000002422365BFD0> output-2: Text(0.5, 1.0, 'House Details') output-2: Text(0, 0.5, 'Area') output-2: Text(0.5, 0, 'Rooms')

The “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface as said earlier. The Java applet (Fig. 6a. Java Applet), Java Swing (Fig. 6b. Java Swing), and Java FX (Fig. 6c. Java FX) are shown in the visualization from the Python area in the user interface, which is shown in Figure 6 scatter chart with Matplotlib. It shows the rooms on the x-axis and the area on the y-axis. The chart clearly shows that 1 and 5 rooms have only one, and other rooms have more.



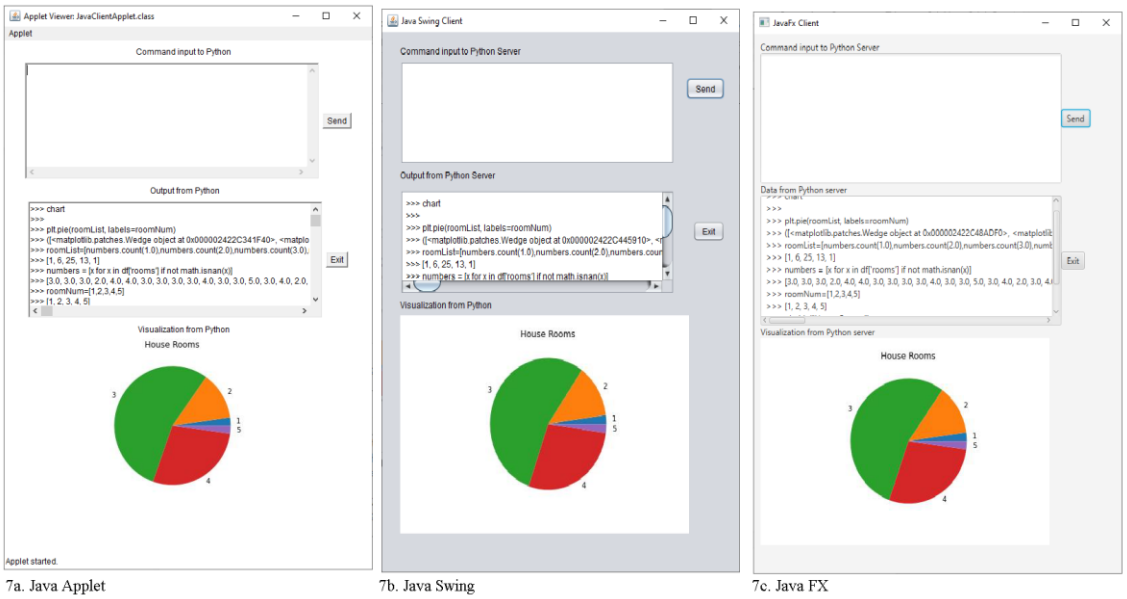
6a. Java Applet 6b. Java Swing 6c. Java FX

Fig.6. Scatter plot with matplotlib

Table 5 shows the program code used to enter the pie chart. The title of the plot is “House Rooms" and the pie chart is generated by counting the number of rooms after removing null or empty values from df['rooms’]. The roomNum array was created to align with the count values to prepare the chart. The output-2 was produced on the Python server since a string output was generated.

Table 5. Program code for pie chart with matplotlib

Java client (Applet, Swing, Java FX)	Python Server
plt.title("House Rooms") roomNum=[1,2,3,4,5] numbers = [x for x in df['rooms'] if not math.isnan(x)] roomList=[numbers.count(1.0),numbers.count(2.0), numbers.count(3.0),numbers.count(4.0),numbers.count(5.0)] plt.pie(roomList, labels=roomNum) chart	output-2: Text(0.5, 1.0, 'House Rooms') output-2: [1, 2, 3, 4, 5] output-2: [3.0, 3.0, 3.0, 2.0, 4.0, 4.0, 3.0, 3.0, 3.0, 3.0, 4.0, 3.0, 3.0, 5.0, 3.0, 4.0, 2.0, 3.0, 4.0, 4.0, 3.0, 2.0, 3.0, 4.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 2.0, 1.0, 4.0, 3.0, 4.0, 3.0, 3.0, 4.0, 4.0, 2.0, 3.0, 4.0, 3.0, 2.0, 4.0, 3.0] output-2: [1, 6, 25, 13, 1] output-2:([<matplotlib.patches.Wedge object at 0x000002422C341F40>, …



7a. Java Applet 7b. Java Swing 7c. Java FX

Fig.7. Pie chart with matplotlib

The “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface as said earlier. The Java applet (Fig. 7a. Java Applet), Java Swing (Fig. 7b. Java Swing), and Java FX (Fig. 7c. Java FX) are shown in the visualization from the Python area in the user

interface, which is shown in Figure 7 pie chart with Matplotlib. It shows more than half of the pie chart, with 3 rooms, followed by 4 rooms.

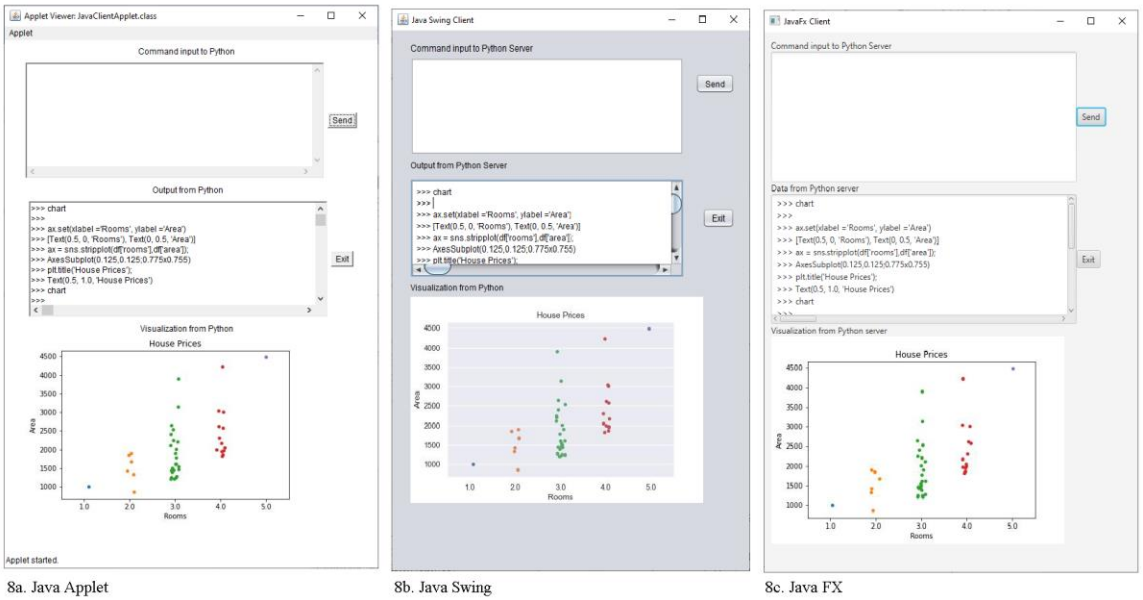
Table 6 shows the program code used to enter the strip plot. The strip plot x-axis label represents the rooms, and the y-axis label represents the area, respectively. The title of the plot is “House Price" and the strip plot generated for df['rooms’] and df['area'] variables are represented by the x and y-axes, respectively. The output-2 was produced on the Python server since a string output was generated.

Table 6. Program code for strip plot with seaborn

Java client (Applet, Swing, Java FX)	Python Server

plt.title('House Prices');
ax = sns.stripplot(df['rooms'],df['area']);
ax.set(xlabel ='Rooms', ylabel ='Area')
chart
	output-2: Text(0.5, 1.0, 'House Prices')
	output-2: AxesSubplot(0.125,0.125;0.775x0.755)
	output-2: [Text(0.5, 0, 'Rooms'), Text(0, 0.5, 'Area')]

As said earlier, the “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface. The Java applet (Fig. 8a. Java Applet), Java Swing (Fig. 8b. Java Swing), and Java FX (Fig. 8c. Java FX) are shown in the visualization from the Python area in the user interface, which is shown in Figure 8 strip plot with Matplotlib. The chart clearly shows that 1 and 5 rooms have only one, and other rooms have more.



8a. Java Applet 8b. Java Swing 8c. Java FX

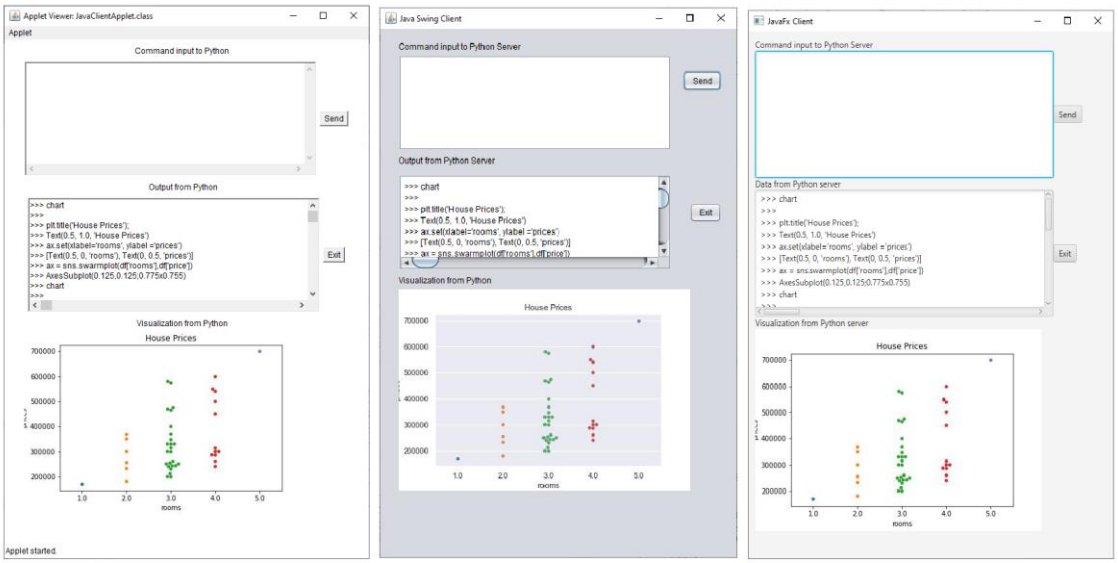
Fig.8. Strip plot with seaborn

Table 7 shows the program code used to enter the swarm plot. The swarm plot x-axis label represents the rooms, and the y-axis label represents the price, respectively. The title of the plot is “House Prices" and the strip plot generated for df['rooms’] and df['price'] variables is represented by the x and y axes, respectively. The output-2 was produced on the Python server since a string output was generated.

Table 7. Program code for swarm plot with seaborn

Java client (Applet, Swing, Java FX)	Python Server
ax = sns.swarmplot(df['rooms'],df['price']) ax.set(xlabel='rooms', ylabel ='prices') plt.title('House Prices'); chart	output-2: AxesSubplot(0.125,0.125;0.775x0.755) output-2: [Text(0.5, 0, 'rooms'), Text(0, 0.5, 'prices')] output-2: Text(0.5, 1.0, 'House Prices')

The “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface as said earlier. The Java applet (Fig. 9a. Java Applet), Java Swing (Fig. 9b. Java Swing), and Java FX (Fig. 9c. Java FX) are shown in the visualization from the Python area in the user interface, which is shown in Figure 9 pie chart with Matplotlib. It is like the previous strip plot, which clearly shows that 1 and 5 rooms have only one, and other rooms have more.



9a. Java Applet 9b. Java Swing 9c. Java FX

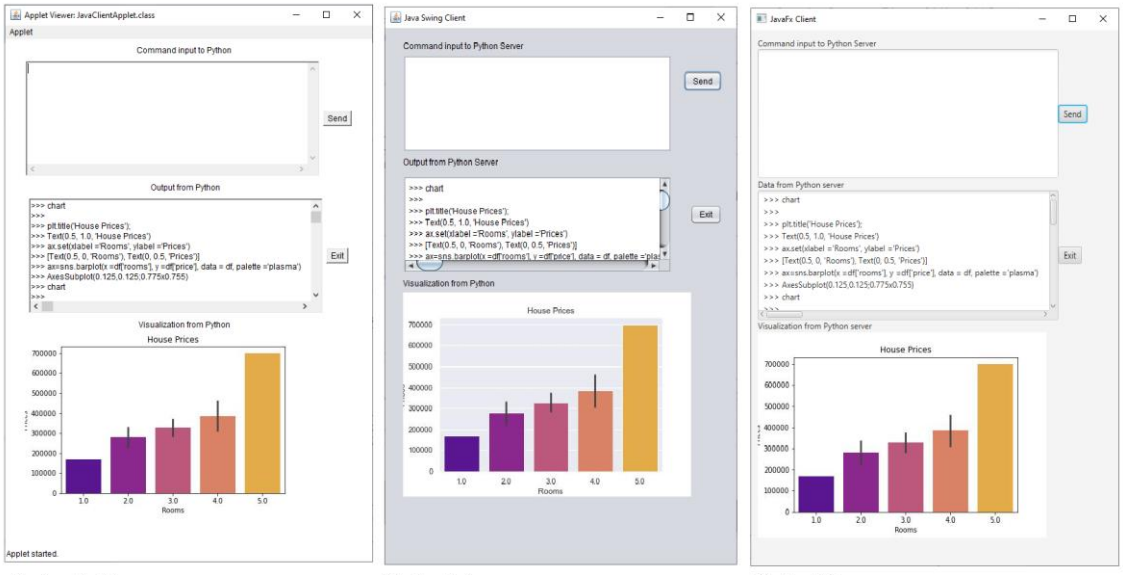
Fig.9. Swarm plot with seaborn

Table 8 shows the program code used to enter the bar plot for Seaborn. The bar plot x-axis label represents the rooms, and the y-axis label represents the price, respectively. The title of the plot is “House Prices" and the strip plot generated for df['rooms’] and df['price'] variables is represented by the x and y axes, respectively. The output-2 was produced on the Python server since a string output was generated.

Table 8. Program code for Bar Plot with Seaborn

Java client (Applet, Swing, Java FX)	Python Server
ax=sns.barplot(x =df['rooms'], y =df['price'], data = df, palette ='plasma') ax.set(xlabel ='Rooms', ylabel ='Prices') plt.title('House Prices'); chart	output-2: AxesSubplot(0.125,0.125;0.775x0.755) output-2: [Text(0.5, 0, 'Rooms'), Text(0, 0.5, 'Prices')] output-2: Text(0.5, 1.0, 'House Prices')

The “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface as said earlier. The Java applet (Fig. 10a. Java Applet), Java Swing (Fig. 10b. Java Swing), and Java FX (Fig. 10c. Java FX) are shown in the visualization from the Python area in the user interface, which is shown in Figure 10 bar plot chart with Seaborn. The chart clearly shows that the highest price falls on a 5-room house, then 4 rooms, and later 3 rooms. We can understand that as rooms increase, areas are also increased, and prices are proportional.



10a. Java Applet 10b. Java Swing 10c. Java FX

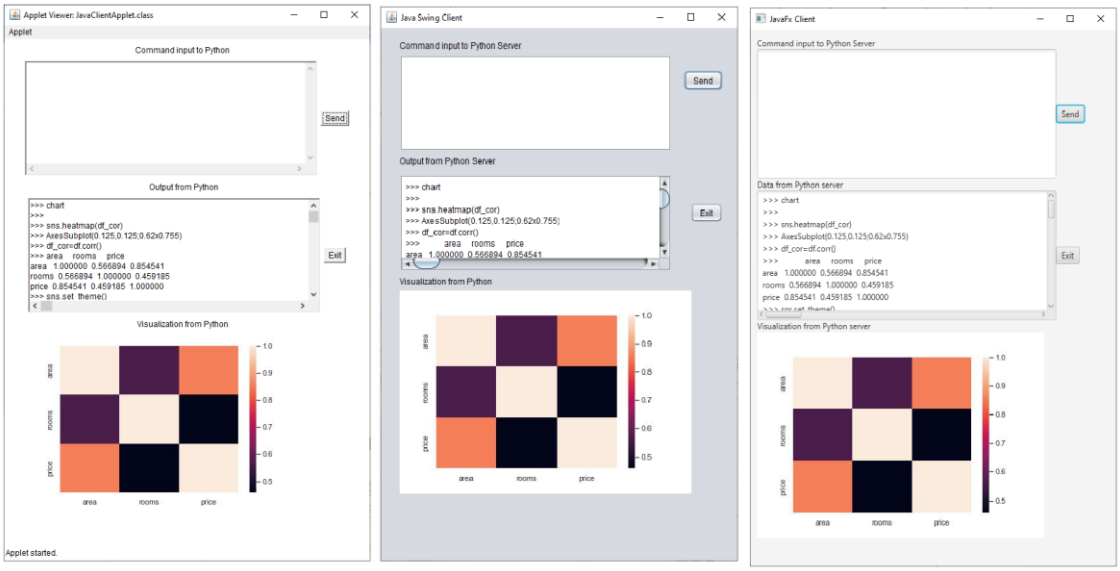
Fig.10. Bar plot with seaborn

Table 9 shows the program code used to enter the heatmap plot for Seaborn. The heatmap plot is generated for correlation of the data frame loaded for the “data.csv”. The output-2 was produced on the Python server since a string output was generated.

Table 9. Program code for heatmap plot with seaborn

Java client (Applet, Swing, Java FX)	Python Server
sns.set_theme() df_cor=df.corr() sns.heatmap(df_cor) chart	output-2: None output-2: area rooms price area 1.000000 0.566894 0.854541 rooms 0.566894 1.000000 0.459185 price 0.854541 0.459185 1.000000 output-2: AxesSubplot(0.125,0.125;0.62x0.755)

The “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface as said earlier. The Java applet (Fig. 11a. Java Applet), Java Swing (Fig. 11b. Java Swing), and Java FX (Fig. 11c. Java FX) are shown in the visualization from the Python area in the user interface, which is shown in Figure 11 heatmap plot chart with Seaborn. The chart shows the correlational factor for each variable: rooms, area, and prices.



11a. Java Applet 11b. Java Swing 11c. Java FX

Fig.11. Heatmap plot with seaborn

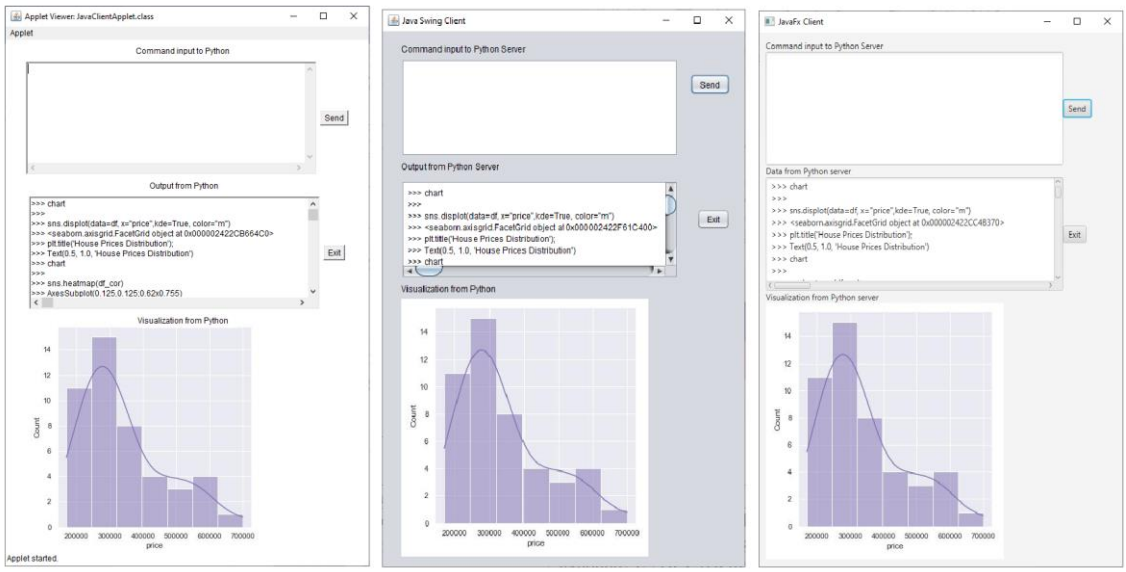
Table 10 shows the program code used to enter the distribution plot for Seaborn. The bar plot x-axis label represents the prices, and the y-axis label represents the count of price, respectively. The distribution plot was generated for df['price'] and output-2 was produced on the Python server since a string output was generated.

Table 10. Program code for distplot with seaborn

Java client (Applet, Swing, Java FX)	Python Server
plt.title('House Prices Distribution'); sns.displot(data=df, x="price",kde=True, color="m") chart	output-2: Text(0.5, 1.0, 'House Prices Distribution') output-2: <seaborn.axisgrid.FacetGrid object at 0x000002422DCE4DF0>

The “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface as said earlier. The Java applet (Fig. 12a. Java Applet), Java Swing (Fig. 12b. Java Swing), and Java FX (Fig. 12c. Java FX) are shown in the visualization from the Python area in the user interfaces, which is shown in Figure 12 distribution plot with Seaborn. The chart clearly shows the highest number of rooms with a price of 30,000.

Table 11 shows the program code used to enter the box plot for Seaborn. The box plot’s x-axis label represents the rooms, and the y-axis label represents the price, respectively. The title of the plot is “Boxplots" and the box plot generated for df['rooms’] and df['price'] variables is represented by the x and y axes, respectively. The output-2 was produced on the Python server since a string output was generated.



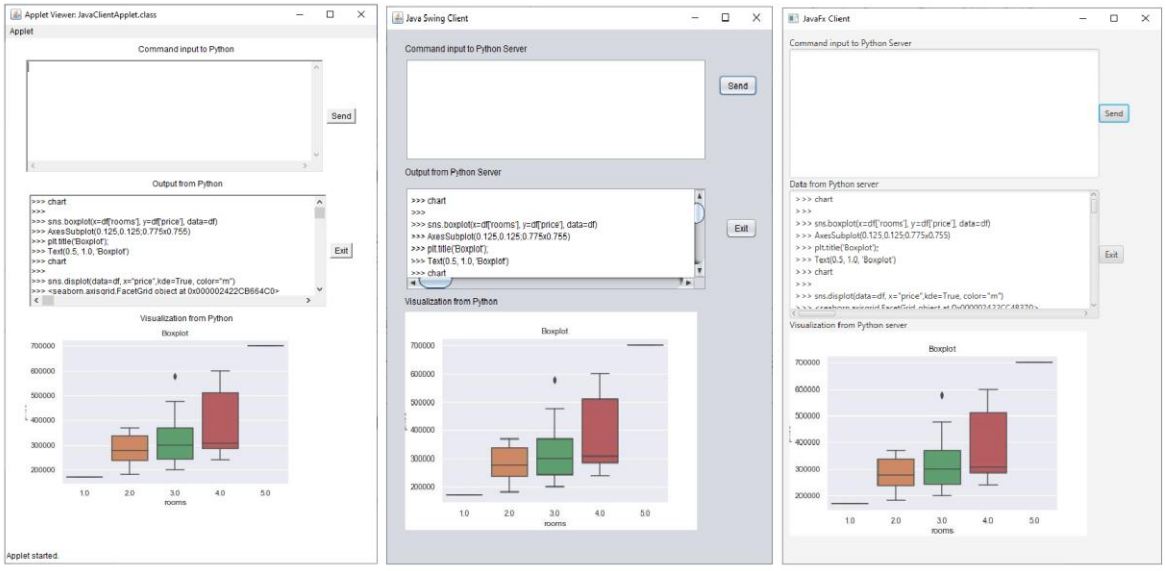
12a. Java Applet 12b. Java Swing 12c. Java FX

Fig.12. Distplot with seaborn

Table 11. Program code for Boxplot with Seaborn

Java client (Applet, Swing, Java FX)	Python Server
plt.title('Boxplot'); sns.boxplot(x=df['rooms'], y=df['price'], data=df) chart	output-2: Text(0.5, 1.0, 'Boxplot') output-2: AxesSubplot(0.125,0.125;0.775x0.755)

As said earlier, the “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface. The Java applet (Fig. 13a. Java Applet), Java Swing (Fig. 13b. Java Swing), and Java FX (Fig. 13c. Java FX) are shown in the visualization from the Python area in the user interface, which is shown in Figure 13 box plot with Seaborn. The chart clearly shows that 1 and 5 rooms have only one, and other rooms have more.



13a. Java Applet 13b. Java Swing 13c. Java FX

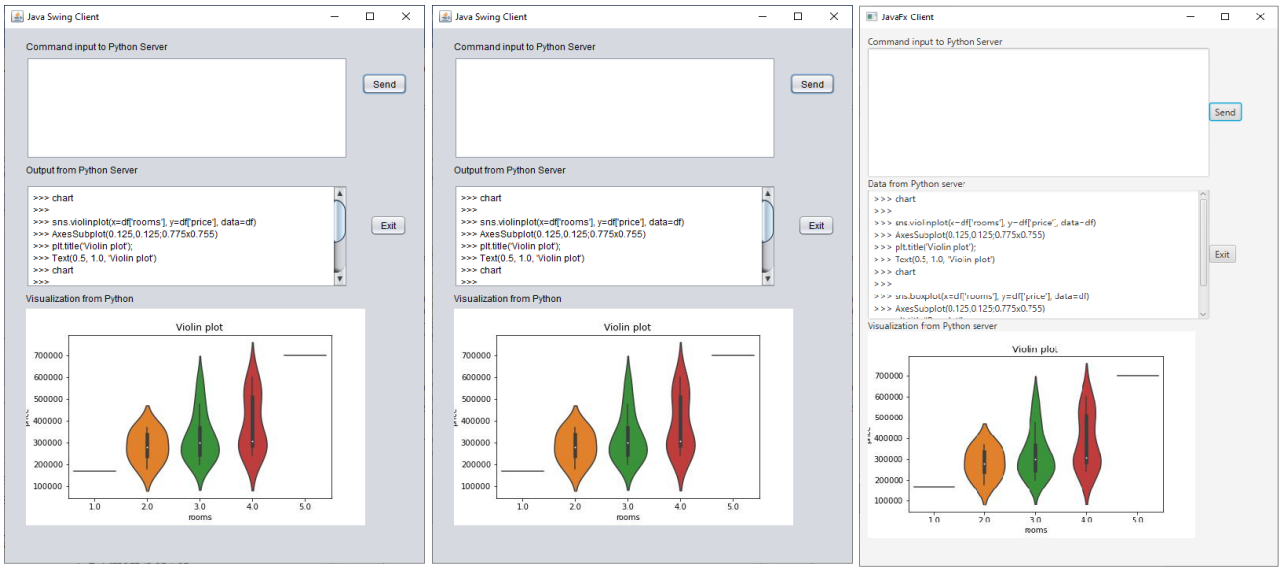
Fig.13. Boxplot with seaborn

Table 12 shows the program code used to enter the violin plot for Seaborn. The violin plot’s x-axis label represents the rooms, and the y-axis label represents the prices, respectively. The title of the plot is “Violin plot" and the strip plot generated for df['rooms’] and df['price'] variables is represented by the x and y axes, respectively. The output-2 was produced on the Python server since a string output was generated.

Table 12. Program code for violin plot with seaborn

Java client (Applet, Swing, Java FX)	Python Server
plt.title('Violin plot'); sns.violinplot(x=df['rooms'], y=df['price'], data=df) chart	output-2: Text(0.5, 1.0, 'Boxplot') output-2: AxesSubplot(0.125,0.125;0.775x0.755)

As said earlier, the “chart” instruction from Java will instruct Python to create a visualization plot named "plot.jpg" and send it to the client, which will be shown in the client interface. The Java applet (Fig. 14a. Java Applet), Java Swing (Fig. 14b. Java Swing), and Java FX (Fig. 14c. Java FX) are shown in the visualization from the Python area in the user interface, which is shown in Figure 14 violin plot with Seaborn. The chart clearly shows that 1 and 5 rooms have only one, and other rooms have more. On the other side of the violin, the price fell from 20,000 to 40,000.



14a. Java Applet 14b. Java Swing 14c. Java FX

Fig.14. Violinplot with seaborn

The overall result of the sample “data.csv” was observed using Matplotlib and Seaborn using a Python server from Java client applications such as Java applet, Java Swing, and Java FX. The observation and visualization shown in various plots show that the house price increases if the rooms are increased, or the house area increases. This research output provides a tool for Python and Java connectivity for data visualization. This research addresses the research design in Section 3.1, the algorithm approach in Section 3.2, and the program code implementations in Section 3.3, which confirm how Python's data visualization through Matplotlib and Seaborn can be effortlessly incorporated into Java GUI applications such as Java applets, Java Swing, and Java FX. This research considered the following essential factors: Python libraries are utilized through TCP socket programming without additional third-party libraries or plugins in the program code implementation in Section 3.3, and they can be utilized in various working environments [34]. The advantages and challenges conferred in Section 1.3 and the signification in Section 1.2 of this research are that make use of Python's data visualization capabilities in Java GUI applications, and the disadvantages of visualization of plots are communicated through a “plot.jpg” picture file that was prepared using Matplotlib and Seaborn, and the visualization became static and could not be interactive. Therefore, the current research did not focus on the interactive features of visualization libraries such as Altair, Bokeh, Plotly, and Pygal, which can be focused on in future research aspects. Another big challenge is to transfer Python visualization over TCP socket communication to load visualization in Java-owned chart libraries or Java-supported third-party libraries. Moreover, the comparative analysis and visualization of Java's own data analysis and visualization with the present research is an opportunity for further engagement in the research. As a conclusion, the research findings will help software developers, data analysts, and scientists choose the optimal Python visualization tool for Java GUI applications for their needs.

5. Summary and Conclusion

AI data scientists utilize Python, which has mathematical libraries and methods that help solve math problems and analyze and visualize data. Python data visualization libraries such as Matplotlib and Seaborn can be integrated into Java applications, providing a vast ecosystem of tools for sophisticated data representations. This research succeeded in integrating Python’s data visualization strengths into Java GUI applications, which will create powerful and versatile applications. This research paper explores the integration of Python's data visualization libraries into Java GUI

applications. The findings highlighted the benefits of leveraging Python's expertise in visualization while maintaining GUI applications in Java. Additionally, it addressed the challenges associated with data visualization. Practical examples illustrate the effectiveness of this integration. Developers seeking to enhance data visualization capabilities in Java GUI applications are encouraged to consider this approach, considering the specific requirements and constraints of their projects. The paper commences by providing Python's popular data visualization libraries for Java GUI applications, such as Matplotlib and Seaborn. Finally, the paper concludes with recommendations for developers seeking to enhance Python's data visualization capabilities in Java GUI applications such as Java Applets, Java Swing, and Java FX. This integration can be recommended in fields like data science, finance, engineering, healthcare, and others. Python libraries can enhance the user experience, save development time and resources, and improve interoperability with existing Java GUI application systems. The integration can have a significant impact in academic and research settings, offering a competitive edge in industries where data presentation is critical. The research also fosters collaboration and knowledge sharing between Python and Java developers. Future research could focus on comparative analysis and visualization of Java's own with the present research, providing interactive data visualization using Altair, Bokeh, Plotly, and Pygal libraries, performance comparisons and security factors in different environments, and integrating Python data visualization into Java web applications with Jakarta Faces as well as no code and low code implementations for the same.

References

[1] Daivi, "10 Python Data Visualization Libraries to Win Over Your Insights," 2023. https://www.projectpro.io/article/python-data-visualization-libraries/543 (accessed Sep. 30, 2023).

[2] B. Melissa, "12 Python Data Visualization Libraries to Explore for Business Analysis," 2022. https://mode.com/blog/python-data-visualization-libraries/ (accessed Oct. 03, 2023).

[3] "Using Matplotlib." https://matplotlib.org/stable/users/index

[4] "User guide and tutorial." https://seaborn.pydata.org/tutorial.html

[5] "A Grammar of Graphics for Python." https://plotnine.readthedocs.io/en/v0.12.3/

[6] plotly, "Plotly Fundamentals," Github, 2020. https://plotly.com/python/plotly-fundamentals/

[7] Labdhisheth, "Visualizing Geographical Data using geoplotlib," 2021. https://medium.com/@labdheesheth/visualizing-geographical-data-using-geoplotlib-d732953abcd5

[8] "Bokeh documentation." https://docs.bokeh.org/en/latest/

[9] "Folium help." https://pypi.org/help/

[10] "Vega-Altair." https://altair-viz.github.io/user_guide/data.html

[11] "Pygal documentation." https://www.pygal.org/en/stable/documentation/index.html

[12] "Using JavaFX Charts." https://docs.oracle.com/javafx/2/charts/jfxpub-charts.htm

[13] "JFreeChart 1.5.3 API." https://www.jfree.org/jfreechart/javadoc/index.html

[14] F. R. Gilbert and D. B. Dahl, "jsr223: A Java platform integration for R with programming languages Groovy, JavaScript, JRuby, Jython, and Kotlin," *R J.*, vol. 10, no. 2, pp. 440–454, 2019, doi: 10.32614/RJ-2018-066.

[15] E. Hehman and S. Y. Xie, "Doing Better Data Visualization," *Adv. Methods Pract. Psychol. Sci.*, vol. 4, no. 4, 2021, doi: 10.1177/25152459211045334.

[16] C. El Hachimi, S. Belaqziz, S. Khabba, and A. Chehbouni, "Data Science Toolkit: An all-in-one python library to help researchers and practitioners in implementing data science-related algorithms with less effort[Formula presented]," *Softw. Impacts*, vol. 12, no. January, p. 100240, 2022, doi: 10.1016/j.simpa.2022.100240.

[17] A. Berti, S. van Zelst, and D. Schuster, "PM4Py: A process mining library for Python[Formula presented]," *Softw. Impacts*, vol. 17, no. July, p. 100556, 2023, doi: 10.1016/j.simpa.2023.100556.

[18] A. Kumar Rathore and R. Rajnish, "Comprehensive review of data visualization techniques using python," *Amity J. Comput. Sci.*, vol. 3, no. 2, pp. 42–48, 2017.

[19] I. Stancin and A. Jovic, "An overview and comparison of free Python libraries for data mining and big data analysis," 2019 *42nd Int. Conv. Inf. Commun. Technol. Electron. Microelectron. MIPRO 2019 - Proc.*, pp. 977–982, 2019, doi: 10.23919/MIPRO.2019.8757088.

[20] S. Shah, R. Fernandez, and S. Drucker, "A system for real-time interactive analysis of deep learning training," *Proc. ACM SIGCHI Symp. Eng. Interact. Comput. Syst. EICS 2019*, no. 2, 2019, doi: 10.1145/3319499.3328231.

[21] K. Munawar and M. S. Naveed, "The Impact of Language Syntax on the Complexity of Programs: A Case Study of Java and Python," *Int. J. Innov. Sci. Technol.*, vol. 4, no. 3, pp. 683–695, 2022, doi: 10.33411/ijist/2022040310.

[22] B. Roziere, M. A. Lachaux, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, no. NeurIPS, 2020.

[23] A. Oberoi and R. Chauhan, "Visualizing data using Matplotlib and Seaborn libraries in Python for data science," *Int. J. Sci. Res. Publ.*, vol. 9, no. 3, p. p8733, 2019, doi: 10.29322/ijsrp.9.03.2019.p8733.

[24] K. Nongthombam, "Data Analysis Using Python," *Int. J. Eng. Res. Technol.*, vol. 10, no. 07, pp. 463–468, 2021.

[25] S. Cao, Y. Zeng, S. Yang, and S. Cao, "Research on Python Data Visualization Technology," *J. Phys. Conf. Ser.*, vol. 1757, no. 1, 2021, doi: 10.1088/1742-6596/1757/1/012122.

[26] F. Li, "Research on Data Visualization Technology Based on Python," *Int. J. Multidiscip. Res. Anal.*, vol. 05, no. 05, pp. 907–910, 2022, doi: 10.47191/ijmra/v5-i5-03.

[27] A. H. Sial, S. Yahya, and S. Rashdi, "Comparative Analysis of Data Visualization Libraries Matplotlib and Seaborn in Python," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 10, no. 1, pp. 277–281, 2021, doi: 10.30534/ijatcse/2021/391012021.

[28] L. Addepalli et al., "Assessing the Performance of Python Data Visualization Libraries: A Review," *Int. J. Comput. Eng. Res. Trends*, vol. 10, no. 1, pp. 29–39, 2023.

[29] Z. K. Mundargi, K. Patel, A. Patel, R. More, S. Pathrabe, and S. Patil, "Plotplay: An Automated Data Visualization Website using Python and Plotly," *2023 Int. Conf. Adv. Technol. ICONAT 2023*, pp. 1–4, 2023, doi: 10.1109/ICONAT57137.2023.10079977.
[30] D. Punasya, H. Kushwah, H. Jain, and R. Sheikh, "an Application for Sales Data Analysis and Visualization Using Python and Django," *Int. Res. J. Mod. Eng.*, no. 06, pp. 1757–1762, 2021, [Online]. Available: www.irjmets.com
[31] J. Zhang, "Python based data visualization and configurable teaching system design and implementation," *Proc. IEEE Asia-Pacific Conf. Image Process. Electron. Comput. IPEC 2021*, pp. 1136–1140, 2021, doi: 10.1109/IPEC51340.2021.9421127.
[32] A. M. Sadeq, "Engine Data Analysis and Visualization in Python," no. September 2023, doi: 10.13140/RG.2.2.21054.87360.
[33] A. D. Emily M. Jennings-Dobbs, Shavawn M. Forester, "Visualizing data interoperability for food systems sustainability research—from spider webs to neural networks," *Curr. Dev. Nutr.*, p. 107386, 2023, doi: 10.1016/j.cdnut.2023.102006.
[34] Bala Dhandayuthapani V., "Implementation of Python Interoperability in Java through TCP Socket Communication", International Journal of Information Technology and Computer Science, Vol.15, No.4, pp.50-62, 2023.
[35] A. Nguyen, "Programming Language interoperability in cross-platform software development," Aalto University, 2022.

Authors' Profiles

Dr. Bala Dhandayuthapani V. is presently working as an IT faculty member at the University of Technology and Applied Sciences, Shinas campus, North Al Batinah, Sultanate of Oman. He has 22 years of experience as a faculty member, including in India, Ethiopia, and Oman. He received his PhD in information technology and computer science from Manonmaniam Sundaranar University, India. He received an M.Tech in information technology from Allahabad Agricultural Institute of Deemed University, an M.S. in information technology, and a B.Sc. in computer science from Bharathidasan University. He has published forty peer-reviewed technical research papers in various international journals and conference proceedings. He has also written a textbook entitled "An Introduction to Parallel and Distributed Computing through Java." He has given different invited technical talks and has been involved in a wide range of academic activities.

How to cite this paper: Bala Dhandayuthapani V., "Python Data Analysis and Visualization in Java GUI Applications Through TCP Socket Programming", International Journal of Information Technology and Computer Science(IJITCS), Vol.16, No.3, pp.72-92, 2024. DOI:10.5815/ijitcs.2024.03.07