UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

"Data Plotting System"

Submitted by:

Srivatsa S Rao	***PES1UG21CS626***
Subhash R	***PES1UG21CS630***
Suhrid Sen	***PES1UG21CS635***
Vinamra Dayal Mathur	***PES1UG22CS848***

*6th Semester <**K**> Section*

Prof. Bhargavi Mokashi
Assistant Professor

January - May 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Table of Contents

1. Problem Statement

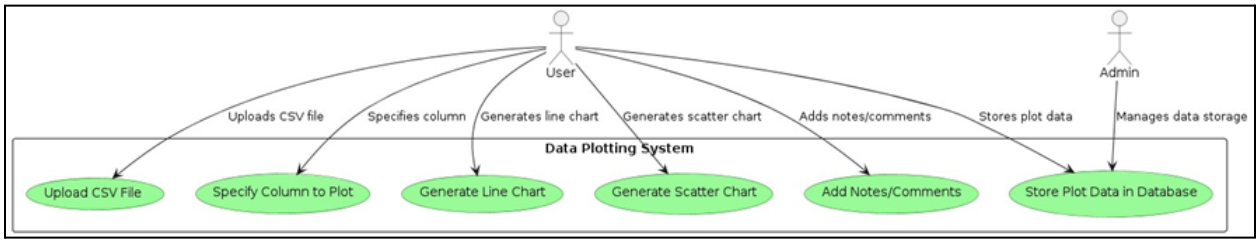
The goal of our project is to develop a data plotting system that allows users to visualize data from CSV files efficiently. Users can upload CSV files, specify the column to be plotted, and generate various types of plots, including line, scatter, bar, and area charts, all presented in separate tabs for easy comparison. The system also provides additional features such as adding a simple moving average to the plotted data, saving the plots as images for later reference, making notes/comments, and storing the plot data in a database that can be extracted later.

Key Features:

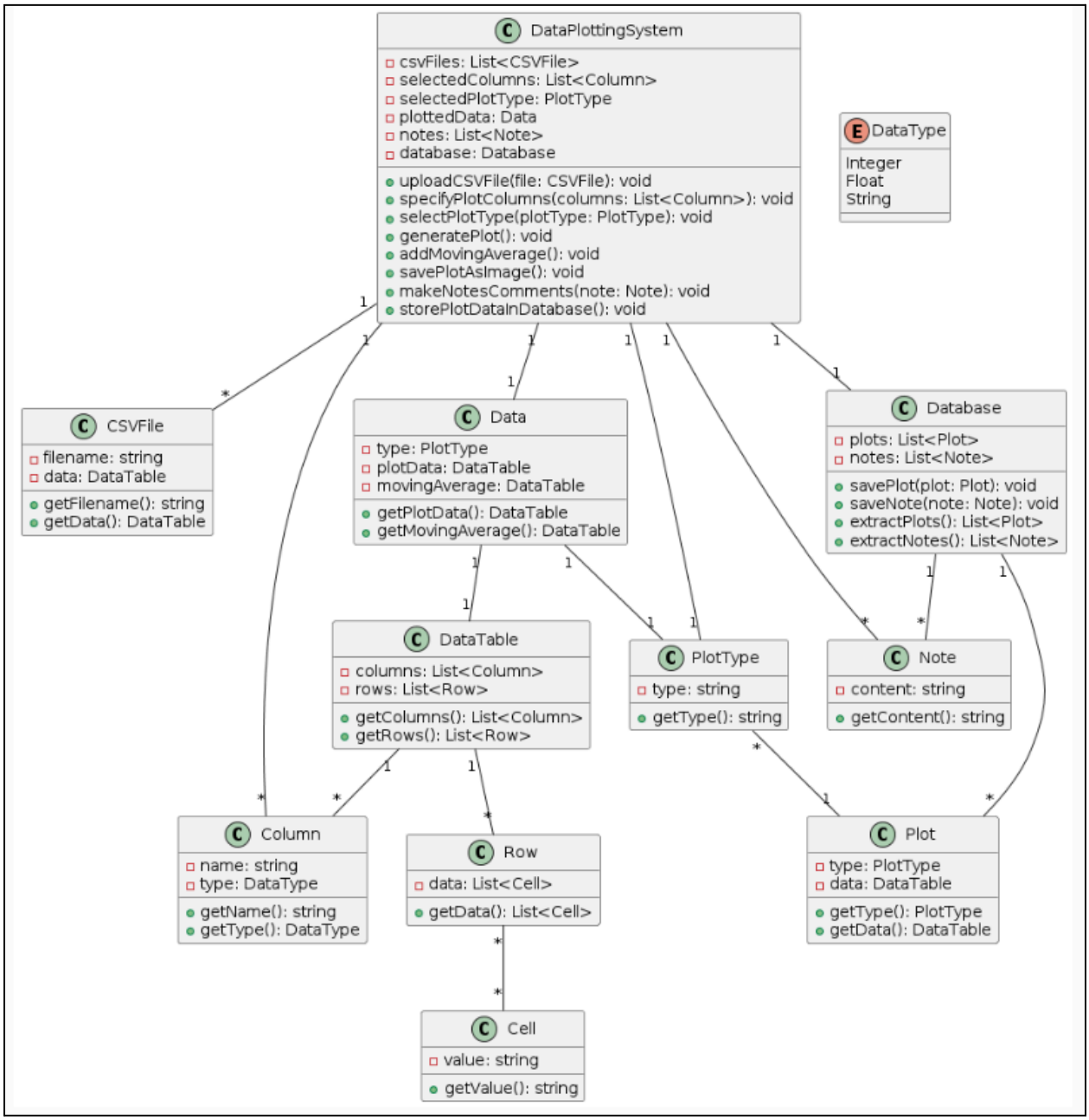
- **Multiple Plot Types:** The system generates four types of plots (line, scatter, bar, area) simultaneously, each displayed in a separate tab.
- **Simple Moving Average:** Users have the option to add a simple moving average to the plotted data.
- **Image Saving:** Plots can be saved as images for later reference and sharing.
- **Note-taking Capability:** Users can make notes on specific plots, enabling them to annotate and record insights.
- **Database Storage:** Plot data along with the notes/ comments can be saved to a database, allowing for easy retrieval and further analysis.

2. Models

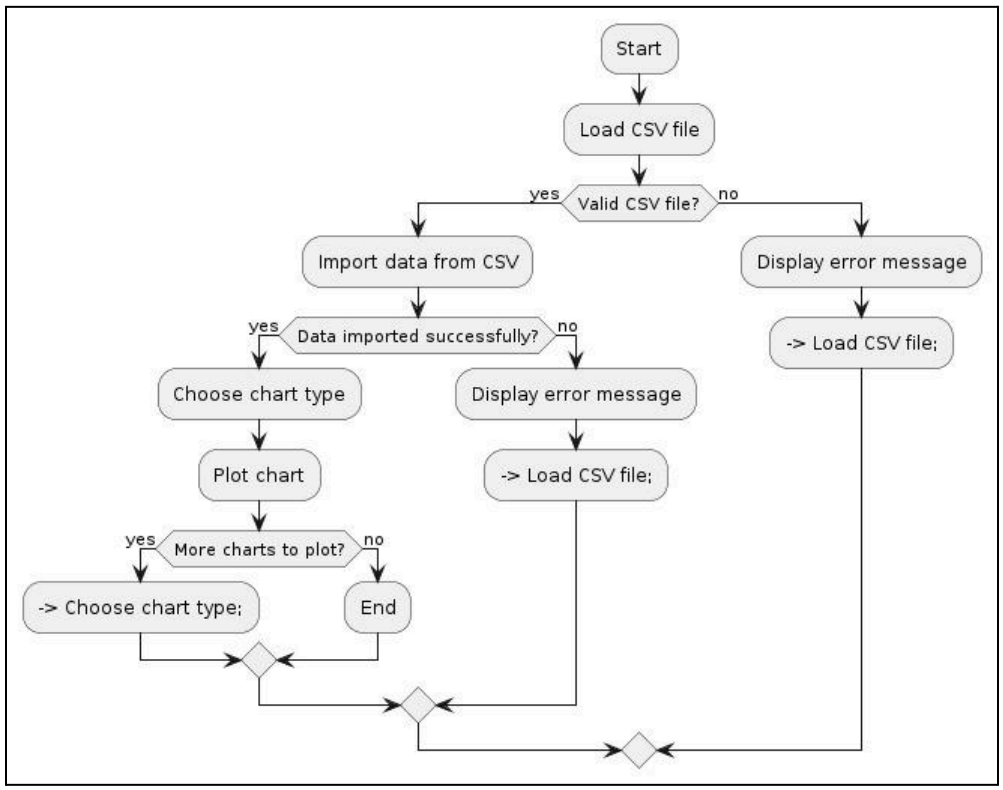
Use Case Diagram



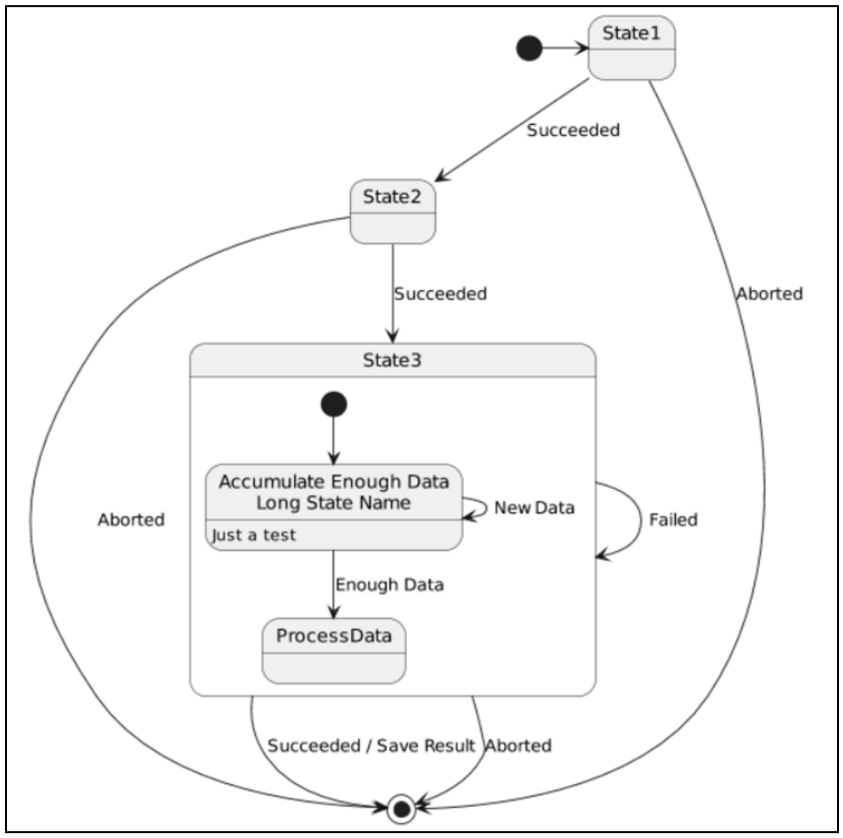
Class Diagram



Activity Diagram



State Diagram



3. Design Architecture

Our MVC architecture:

Model:

- backendSql package: **Package** contains classes related to database operations, such as creating a database, inserting data, removing data, and executing SQL queries.
- parseCsv package: **Package** contains classes for parsing CSV files, which is necessary for reading the user-uploaded data.

View:

- eachPlot_Builder_Product.java: Responsible for building the user interface components for each individual plot.

- eachPlot_FactoryManager.java: This class is a factory manager, responsible for creating different nodes or elements required by each plot
- eachPlot_get_Instance.java: Implementation of the builder class, returns each plots' instance.
- eachPlot_Instance package: This **package** contains classes related to individual plot instances, such as adding a moving average, creating tab panes, saving plots as images, and managing notes.
- addGraphs: This **package** contains all the classes needed for different plot types.

Controller:

- Implement_all_controller.java: The main controller, responsible for coordinating the flow of data and operations between the Model and View components.

4. Design Principles

4.1 Single Responsibility Principle

In most of our classes we have only one method, hence following the SRP principle

4.2 Open/Closed principle

All of the design patterns we have implemented use an interface or an abstract class, hence most of our code base follows the Open/Closed principle.

4.3 Liskov Substitution Principle

The builder pattern we implemented adheres to the Liskov substitution principle.

5. Design Patterns

5.1 Builder

The Builder pattern is a creational design pattern that separates the construction of a complex object from its representation, allowing the same construction process to create different representations.

We have used the builder patterns to create the different components or nodes involved in creating each plot like different buttons, panes, text areas etc.

Builder setters (application/view/eachPlot_Builder_Product.java):

```java
import application.CONSTANTS;

public class eachPlot_Builder_Product {

    private TabPane     _tabPane;
    private TextField   _plotNameArea;
    private Button      _reomvePaneBt;
    private Button      _savetodDBbt;
    private Button      _addMovingAvgBt;
    private Button      _saveImgBt;
    private TextArea    _makeNotesArea;
    public AnchorPane _container;

    public eachPlot_Builder_Product(eachPlot_Builder builder) {

        this._tabPane          = builder._tabPane;
        this._plotNameArea     = builder._plotNameArea;
        this._reomvePaneBt     = builder._reomvePaneBt;
        this._savetodDBbt      = builder._savetodDBbt;
        this._addMovingAvgBt   = builder._addMovingAvgBt;
        this._saveImgBt        = builder._saveImgBt;
        this._makeNotesArea    = builder._makeNotesArea;
        this._saveImgBt        = builder._saveImgBt;
        this._container        = builder._container;
    }
```

```java
    //builder
    public static class eachPlot_Builder {

        protected TabPane     _tabPane;
        protected TextField   _plotNameArea;
        protected Button      _reomvePaneBt;
        protected Button      _savetodDBbt;
        protected Button      _addMovingAvgBt;
        protected Button      _saveImgBt;
        protected TextArea    _makeNotesArea;
        protected AnchorPane _container;


        public eachPlot_Builder set_plotNameArea (double x, double y) {

            this._plotNameArea = eachPlot_FactoryManager.createTextFieldComponent("PlotName");
            AnchorPane.setTopAnchor(this._plotNameArea,  x); //3.0
            AnchorPane.setLeftAnchor(this._plotNameArea, y); //6.0

            return this;
        }

        public eachPlot_Builder set_reomvePaneBt (double x, double y) {

            this._reomvePaneBt = eachPlot_FactoryManager.createButtonComponent("RemovePaneButton");
            AnchorPane.setTopAnchor(this._reomvePaneBt,  x); //160.0
            AnchorPane.setLeftAnchor(this._reomvePaneBt, y); //6.0

            return this;

        }
```

```
66      public eachPlot_Builder set_savetodDBbt (double x, double y) {
67
68          this._savetodDBbt = eachPlot_FactoryManager.createButtonComponent("SaveToDatabase");
69          AnchorPane.setTopAnchor(this._savetodDBbt,  x); //267.0
70          AnchorPane.setLeftAnchor(this._savetodDBbt, y); //6.0
71
72          return this;
73
74      }
75
76      public eachPlot_Builder set_addMovingAvgBt (double x, double y) {
77
78          this._addMovingAvgBt = eachPlot_FactoryManager.createButtonComponent("MovingAvgButton");
79          AnchorPane.setTopAnchor(this._addMovingAvgBt,  x); //425.0
80          AnchorPane.setLeftAnchor(this._addMovingAvgBt, y); //6.0
81
82          return this;
83
84      }
85
86      public eachPlot_Builder set_saveImgBt (double x, double y) {
87
88          this._saveImgBt = eachPlot_FactoryManager.createButtonComponent("SaveChartAsImage");
89          AnchorPane.setTopAnchor(this._saveImgBt,  x); //630.0
90          AnchorPane.setLeftAnchor(this._saveImgBt, y); //6.0
91
92          return this;
93
94      }
95
```

```
96      public eachPlot_Builder set_makeNotesArea (double y) {
97
98          this._makeNotesArea = eachPlot_FactoryManager.createTextAreaComponent("TextAreaForNotes");
99          AnchorPane.setTopAnchor(this._makeNotesArea,  y); //40.0
100         AnchorPane.setLeftAnchor(this._makeNotesArea, CONSTANTS.SCREEN_WIDTH-480);
101
102         return this;
103
104     }
105
106     public eachPlot_Builder set_tabPane () {
107
108         this._tabPane = eachPlot_FactoryManager.createTabPaneComponent("CreateTabPane");
109         AnchorPane.setTopAnchor(this._tabPane,  5.0+2.0+CONSTANTS.RM_BT_HEIGHT);  //2.0 acts like pa
110
111         return this;
112
113     }
114
```

```
115        public eachPlot_Builder set_container (String bgcolor) {
116
117            System.out.println("indide set container------" + this._plotNameArea);
118
119            this._container = new AnchorPane();
120
121            this._container.getChildren().addAll(this._plotNameArea,
122                                                 this._reomvePaneBt,
123                                                 this._savetodDBbt,
124                                                 this._addMovingAvgBt,
125                                                 this._saveImgBt,
126                                                 this._tabPane,
127                                                 this._makeNotesArea);
128
129            this._container.setStyle("-fx-border-color: black;" +
130                                     "-fx-border-width: 2px;"   +
131                                     "-fx-background-color: " + bgcolor + ";" +
132                                     "-fx-padding:      3;");
133
134            this._container.setPrefWidth(CONSTANTS.SCREEN_WIDTH-5);
135
136            return this;
137
138        }
139
140        public eachPlot_Builder_Product build () {
141
142            return new eachPlot_Builder_Product (this);
143
144        }
145
146    }
```

Calling the builder class (application/view/eachPlot_get_Instance.java) :

```
public class eachPlot_get_Instance {


    public static eachPlot_Builder_Product _getInstance () {

        eachPlot_Builder_Product _get = new eachPlot_Builder_Product.eachPlot_Builder()
                                        .set_plotNameArea(6.0, 3.0)
                                        .set_reomvePaneBt(6.0, 160.0)
                                        .set_savetodDBbt(6.0, 267.0)
                                        .set_addMovingAvgBt(6.0, 425.0)
                                        .set_saveImgBt(6.0, 630.0)
                                        .set_makeNotesArea(40.0)
                                        .set_tabPane()
                                        .set_container("#FAF9F6")
                                        .build();


        return _get;
    }
}
```

5.2 Factory

The Factory Pattern is a creational design pattern that provides an interface for creating objects, but allows the subclasses to decide which class to instantiate. It promotes loose coupling by encapsulating the object creation logic and hiding the object creation details from the client code.

We have used the factory pattern for the creation of different components each of which have been defined in their respective files. This allows us for cleaner maintainability.

Factory manager responsible for returning different classes needed by the abstraction class (application/view/eachPlot_FactoryManager.java) :

```java
public class eachPlot_FactoryManager {

    public static Button createButtonComponent(String type) {
        if ("MovingAvgButton".equals(type)) {
            return new addMovingAvg().createComponent();

        } else if ("RemovePaneButton".equals(type)) {
            return new removePaneButton().createComponent();

        } else if ("SaveChartAsImage".equals(type)) {
            return new saveChartAsImage().createComponent();

        } else if ("SaveToDatabase".equals(type)) {
            return new savetoDB().createComponent();
        }

        return null;
    }

    public static TabPane createTabPaneComponent(String type) {
        if ("CreateTabPane".equals(type)) {
            return new createTabPane().createComponent();
        }
        return null;
    }

    public static TextArea createTextAreaComponent(String type) {
        if ("TextAreaForNotes".equals(type)) {
            return new makeNotes().createComponent();
        }
        return null;
    }

    public static TextField createTextFieldComponent(String type) {
        if ("PlotName".equals(type)) {
            return new plotName().createComponent();
        }
        return null;
    }


}
```

The different classes are then requested by the builder (see all setters 5.1). Hence c**reating the different elements involves two layers of abstraction**, first the builder is called, the builder then calls the factory.

5.3 Flyweight

The Flyweight pattern is a structural design pattern that aims to reduce the memory usage by sharing common data between multiple objects, instead of storing the same data in each object.

We have used the flyweight pattern in the parsing of the csv file, when plotting data, there are times when the data values are the same in different files, so if the value already exists in memory, then it just returns that value, instead of creating new memory space for that value.

Code for Flyweight pattern instance (application/model/parseCsv/parseFlyweight.java):

```java
package application.model.parseCsv;

import java.util.HashMap;
import java.util.Map;


public class parseFlyweight {

    private static final Map<String, String> flyweights = new HashMap<>();

    public static String getFlyweight(String value) {

        if (!flyweights.containsKey(value)) {

            System.out.println("creating new flyweight since does not exist: " + value);
            flyweights.put(value, value);

        } else {
            System.out.println("Value already exists, getting from previous flyweight: " + value);
        }

        return flyweights.get(value);
    }

}

```

Using the parseFlyweight class (application/model/parseCsv/ParseCsv.java):

```java
                String line;
                while ((line = reader.readLine()) != null) {
                    String[] columns = line.split(CONSTANTS.DELIMITER);
                    if (columnIndex >= 0 && columnIndex < columns.length) {

                        String flyweight = parseFlyweight.getFlyweight(columns[columnIndex]);
                        columnValues.add(flyweight);
                        //columnValues.add(columns[columnIndex]);
                    }
                }
            }
```

5.4 Strategy

The Strategy Pattern is a behavioral design pattern that allows you to define a family of algorithms, encapsulate each one, and make them interchangeable. It enables the algorithm to vary independently from the clients that use it.

We have used the strategy pattern in the different types of graphs that will be plotted by the system, namely line, scatter, bar and area graphs. Each graph types implements the defined graph strategy.

Graph strategy (application/view/eachPlot_Instance/addGraphs/GraphStrategy.java):

```
1 package application.view.eachPlot_Instance.addGraphs;
2 import javafx.scene.layout.AnchorPane;
3
4 interface GraphStrategy {
5     AnchorPane addGraph();
6 }
7
8
```

Implementation of graph strategy (application/view/eachPlot_Instance/addGraphs/addAreaGraph.java, application/view/eachPlot_Instance/addGraphs/addBarGraph.java, application/view/eachPlot_Instance/addGraphs/addLineGraph.java, application/view/eachPlot_Instance/addGraphs/addScatterGraph.java):

```
14 public class addAreaGraph implements GraphStrategy{
15
16     @Override
17     public AnchorPane addGraph() {
18
19         NumberAxis xAxis = new NumberAxis();
20         NumberAxis yAxis = new NumberAxis();
21         //xAxis.setLabel("X Axis");
22         //yAxis.setLabel(csvColName.getText());
23
24         AreaChart<Number, Number> areaChart = new AreaChart<>(xAx
25         areaChart.setTitle(("Plot from: " + Implement_all_control
26         areaChart.setId("areaChart" + "_tabbedPane_" + Implement_
27
28         XYChart.Series<Number, Number> dataSeries = new XYChart.S
29         dataSeries.setName(Implement_all_controller.csvColName.ge
```

```
14 public class addBarGraph implements GraphStrategy{
15
16     @Override
17     public AnchorPane addGraph() {
18
19         CategoryAxis xAxis = new CategoryAxis();
20         NumberAxis yAxis = new NumberAxis();
21         //xAxis.setLabel("X Axis");
22         //yAxis.setLabel(csvColName.getText());
23
24         BarChart<String, Number> barChart = new BarChart
```

Endpoint for setting the strategy
(application/view/eachPlot_Instance/addGraphs/addGraphStrategy.java) :

```
3  import javafx.scene.layout.AnchorPane;
4
5  public class addGraphStrategy {
6
7      private GraphStrategy strategy;
8
9      public void setStrategy(GraphStrategy strategy) {
10         this.strategy = strategy;
11     }
12
13     public AnchorPane addGraph() {
14         return strategy.addGraph();
15     }
16
17 }
18
```

Calling the set strategy endpoint (application/view/eachPlot_Instance/createTabPane.java) :

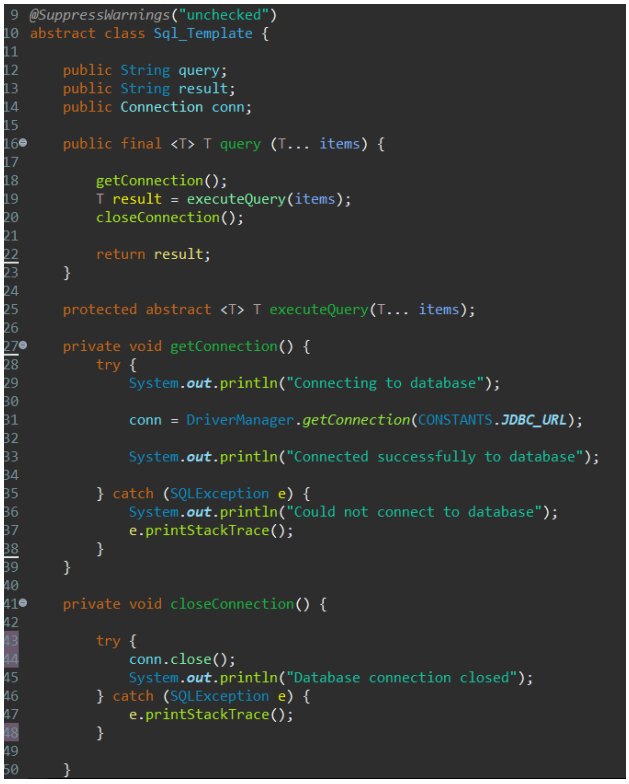
```
43          addGraphStrategy addgraphs = new addGraphStrategy();
44
45
46          //used strategy pattern here
47          addgraphs.setStrategy(new addLineGraph());
48          line.setContent(addgraphs.addGraph());
49
50
51          addgraphs.setStrategy(new addScatterGraph());
52          scatter.setContent(addgraphs.addGraph());
53
54          addgraphs.setStrategy(new addBarGraph());
55          bar.setContent(addgraphs.addGraph());
56
57          addgraphs.setStrategy(new addAreaGraph());
58          area.setContent(addgraphs.addGraph());
59
60
61          tabPane.getTabs().add(line);
62          tabPane.getTabs().add(scatter);
63          tabPane.getTabs().add(bar);
64          tabPane.getTabs().add(area);
65
66          return tabPane;
67      }
68
```

5.5 Template

The Template Pattern is a behavioral design pattern that defines the skeleton of an algorithm in a method, deferring some steps to subclasses. It allows subclasses to redefine certain steps of an algorithm without changing the algorithm's structure.

We have used the template in places where a database connection is needed. Here the two common methods done by every database query is the getConnection() and closeConnection() method. Only the query has been deferred to the subclass that implements the template.

Defining the template (application/model/backendSql/Sql_Template.java):



```java
@SuppressWarnings("unchecked")
abstract class Sql_Template {

    public String query;
    public String result;
    public Connection conn;

    public final <T> T query (T... items) {

        getConnection();
        T result = executeQuery(items);
        closeConnection();

        return result;
    }

    protected abstract <T> T executeQuery(T... items);

    private void getConnection() {
        try {
            System.out.println("Connecting to database");

            conn = DriverManager.getConnection(CONSTANTS.JDBC_URL);

            System.out.println("Connected successfully to database");

        } catch (SQLException e) {
            System.out.println("Could not connect to database");
            e.printStackTrace();
        }
    }

    private void closeConnection() {

        try {
            conn.close();
            System.out.println("Database connection closed");
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }
```

Using the template by a subclass (application/model/backendSql/Remove_FromDb.java, application/model/backendSql/Insert_IntoDb.java, application/model/backendSql/extractEntire_instance.java, application/model/backendSql/extractDataForPlot.java):

```java
public class Remove_FromDb extends Sql_Template {

    @SuppressWarnings("unchecked")
    @Override
    protected <T> T executeQuery(T... items) {

        int _id = (int) items[0];

        query = "DELETE FROM plotData WHERE id = ?";

        try {

            PreparedStatement pstmt = conn.prepareStatement(query);
            pstmt.setInt(1, _id);
            pstmt.executeUpdate();

        } catch (SQLException e) {
            System.out.println("Error deleting row: " + e.getMessage());
            e.printStackTrace();
        }

        System.out.println("Instance with id: " + _id + "sucessfully removed from db");
        return (T) "Removed from DB";

    }
}
```

```java
public class extractEntire_instance extends Sql_Template {

    private ResultSet rs;

    @SuppressWarnings("unchecked")
    @Override
    protected <T> T executeQuery(T... items) {

        query = "SELECT * FROM plotData";

        try {

            Statement stmt = conn.createStatement();
            rs = stmt.executeQuery(query);

        } catch (SQLException e) {
            e.printStackTrace();
        }

        return (T) rs;

    }

}
```

6. Github Link

https://github.com/subhash-023/Data_Plotting_System

7. Individual Contributions

Layout of our project (directory structure visualization) for reference:

- Srivatsa S Rao (PES1UG21CS626)
    - application/Implement_all_controller.java
    - application/view/eachPlot_Builder_Product.java
    - application/view/eachPlot_FactoryManager.java
    - application/view/eachPlot_get_Instance.java

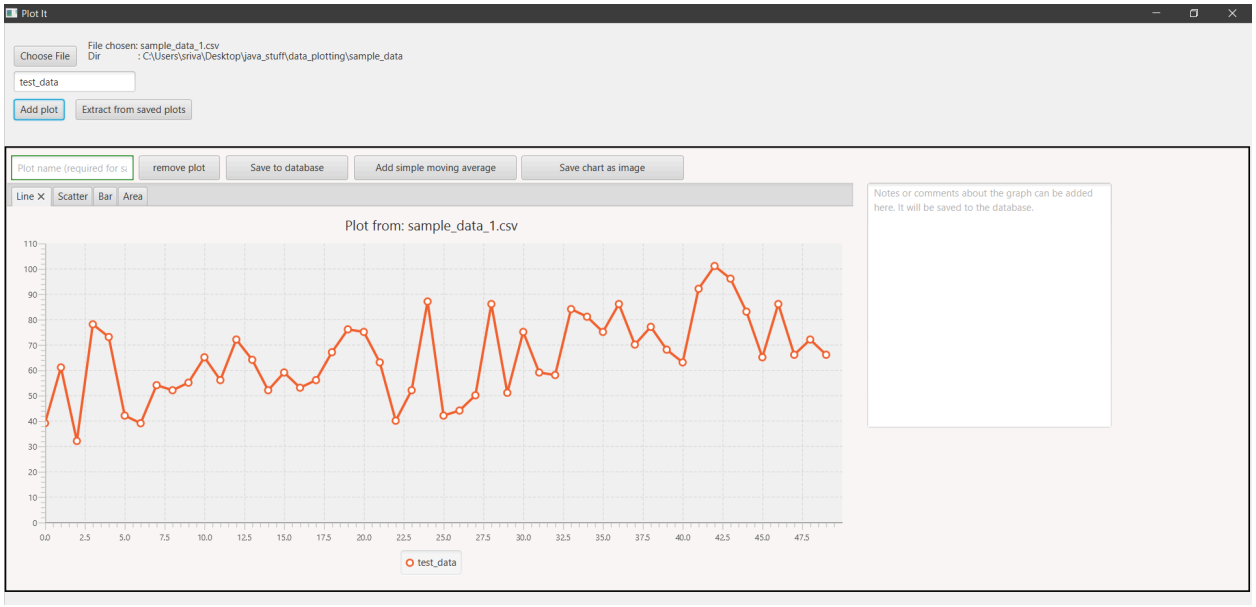
    - application/view/fileChooserWindow.java
- Subhash R (PES1UG21CS630)
    - application/model/backendSql/CreateDB.java
    - application/view/eachPlot_Instance/addGraphs/GraphStrategy.java
    - application/view/eachPlot_Instance/addGraphs/addGraphStrategy.java
    - application/view/eachPlot_Instance/addGraphs/addGraphsaddAreaGraph.java
    - application/view/eachPlot_Instance/addGraphs/addBarGraph.java
    - application/view/eachPlot_Instance/saveChartAsImage.java
    - application/view/eachPlot_Instance/savetoDB.java
- Suhrid Sen (PES1UG21CS635)
    - application/view/eachPlot_Instance/createTabPane.java
    - application/view/eachPlot_Instance/eachPlot_Instance_Factory.java
    - application/view/eachPlot_Instance/makeNotes.java
    - application/view/eachPlot_Instance/removePaneButton.java
    - application/view/eachPlot_Instance/addGraphs/addLineGraph.java
    - application/view/eachPlot_Instance/addGraphs/addScatterGraph.java
- Vinamra Dayal Mathur (PES1UG22CS848)
    - application/model/backendSql/extractDataForPlot.java
    - application/model/backendSql/extractEntire_instance.java
    - application/model/backendSql/Insert_IntoDb.java
    - application/model/backendSql/Sql_Template.java
    - application/model/parseCsv/parseFlyweight.java
    - application/model/parseCsv/ParseCsv.java

8. Output Screenshots

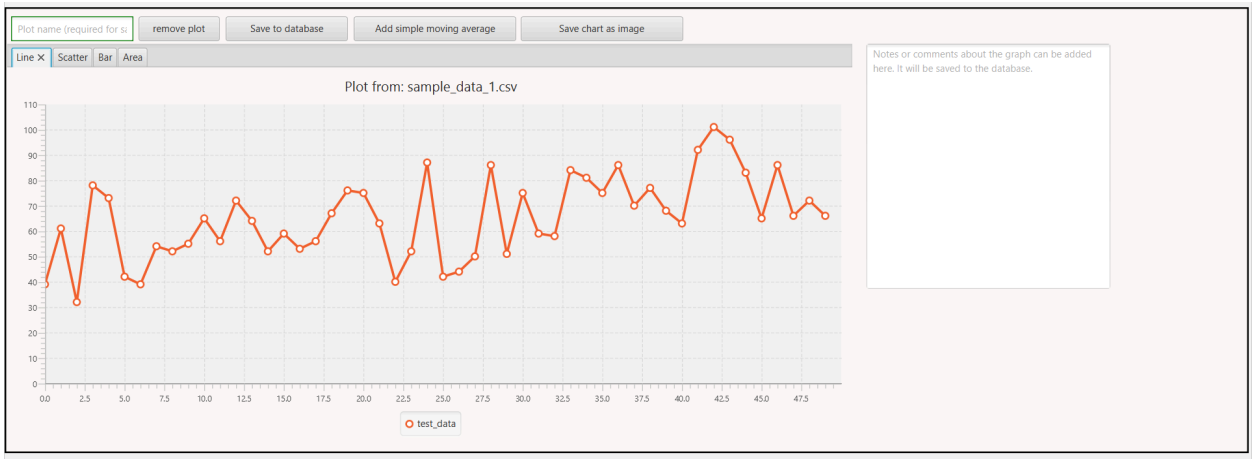
Start window:

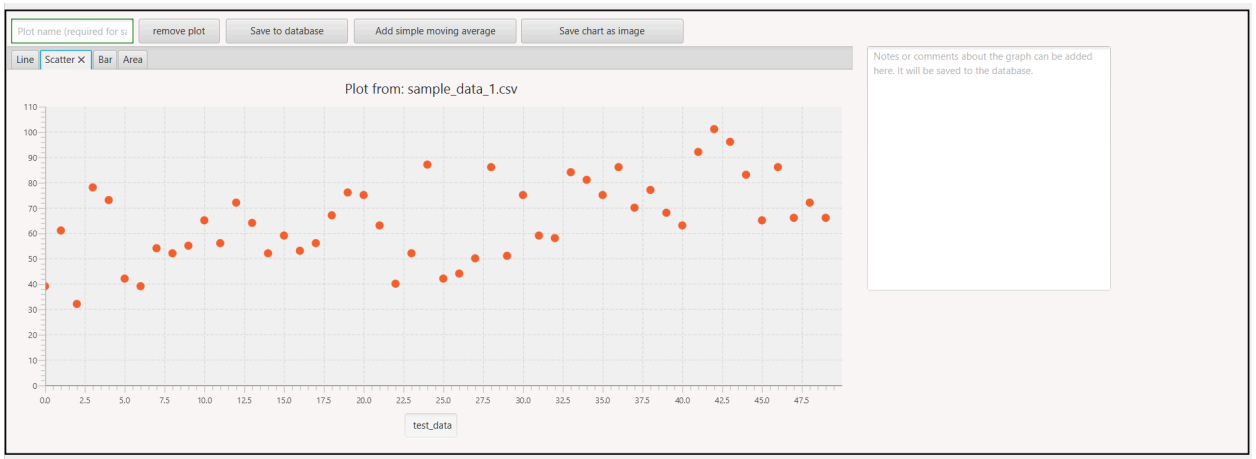
User chooses CSV file and enters the column name of the csv file to be plotted:

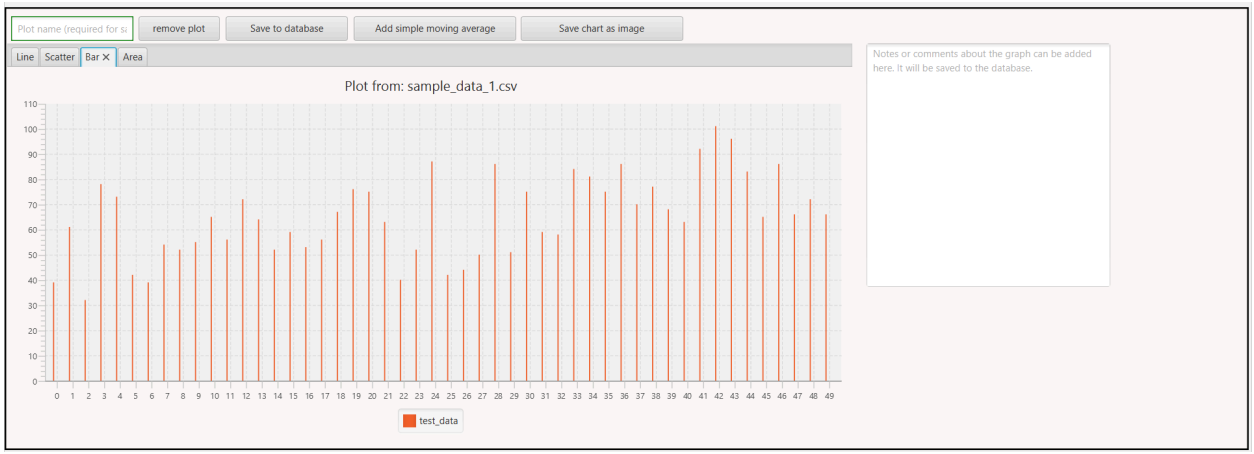
After adding plot:

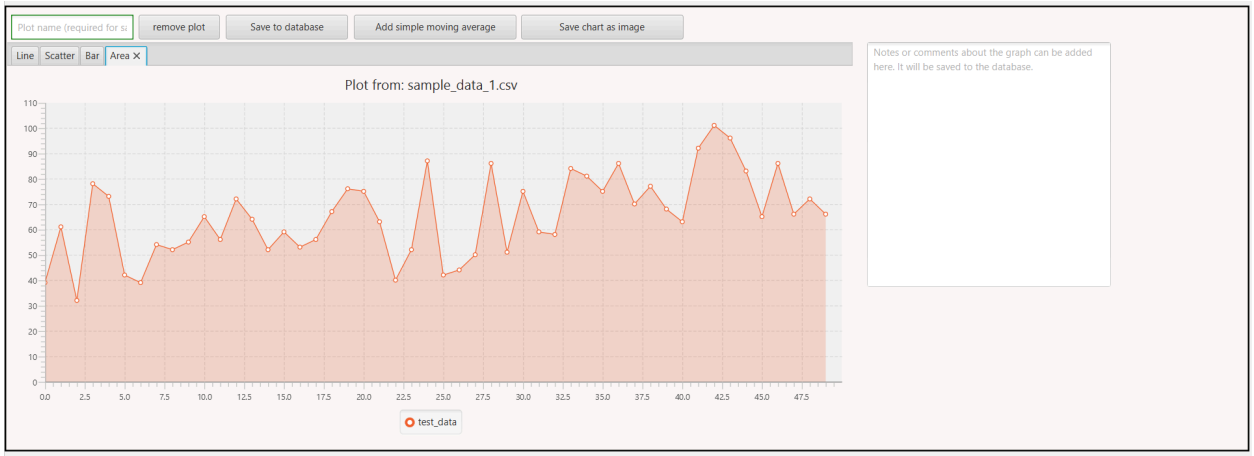


All four different plots (line, scatter, bar, area) plotted by the system:

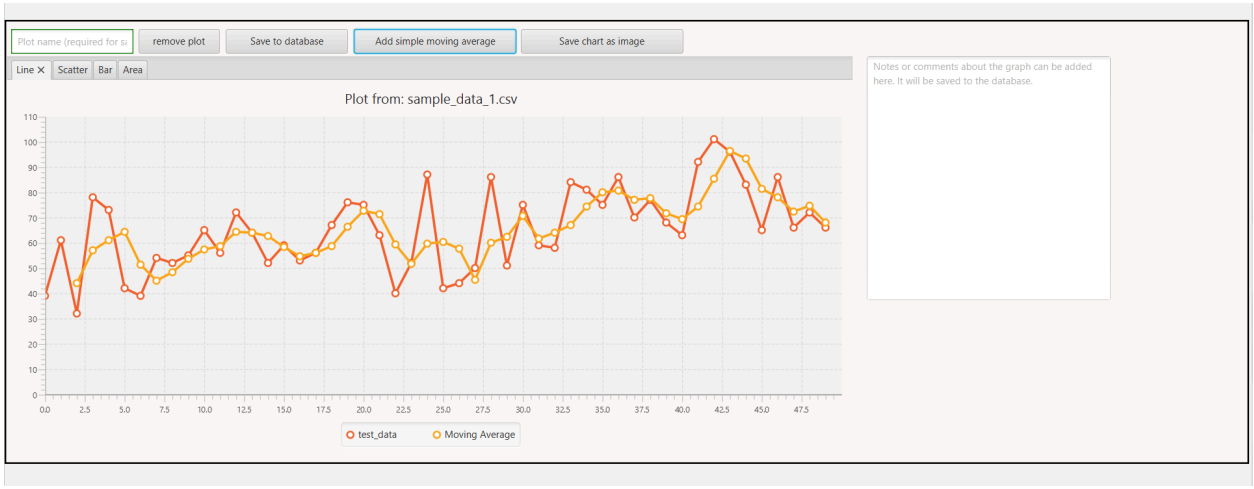


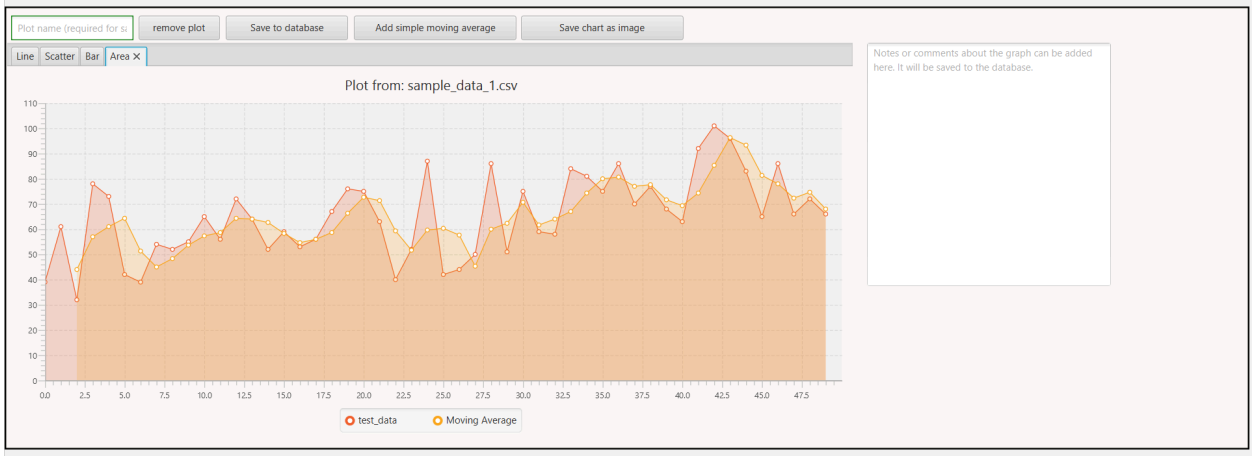




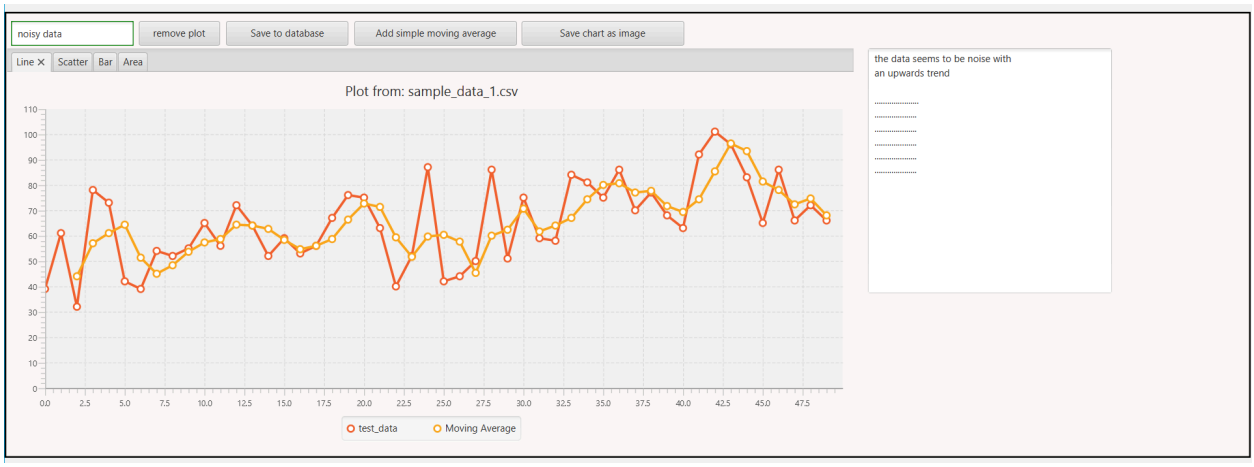


Adding a simple moving average(the moving average as added to only line and area type charts):





After adding a plot name and making notes/comments in the section provided:



Saving the plot as an image:

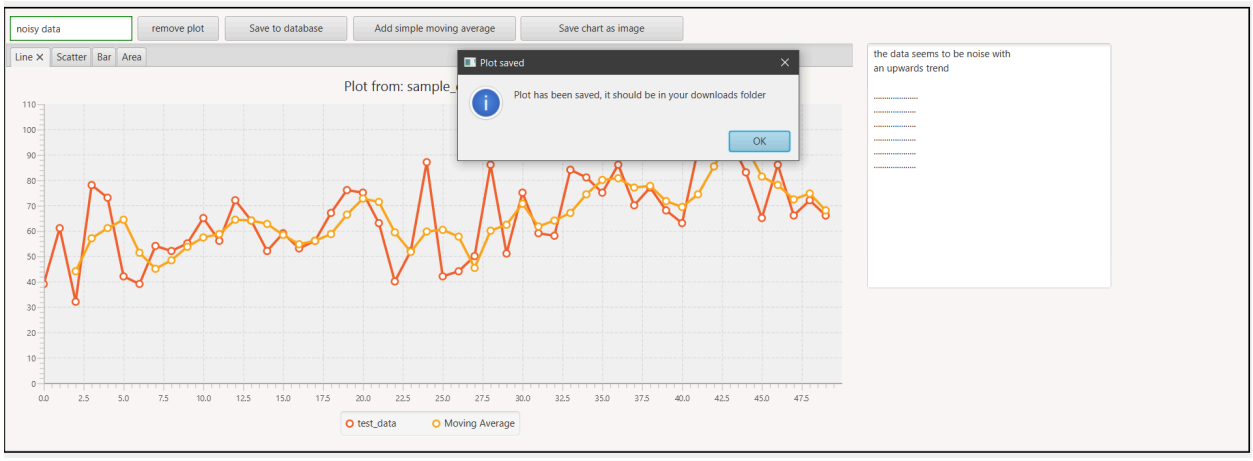


Image in the downloads folder:

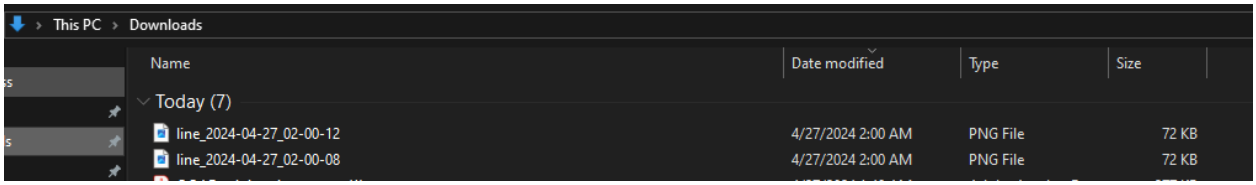
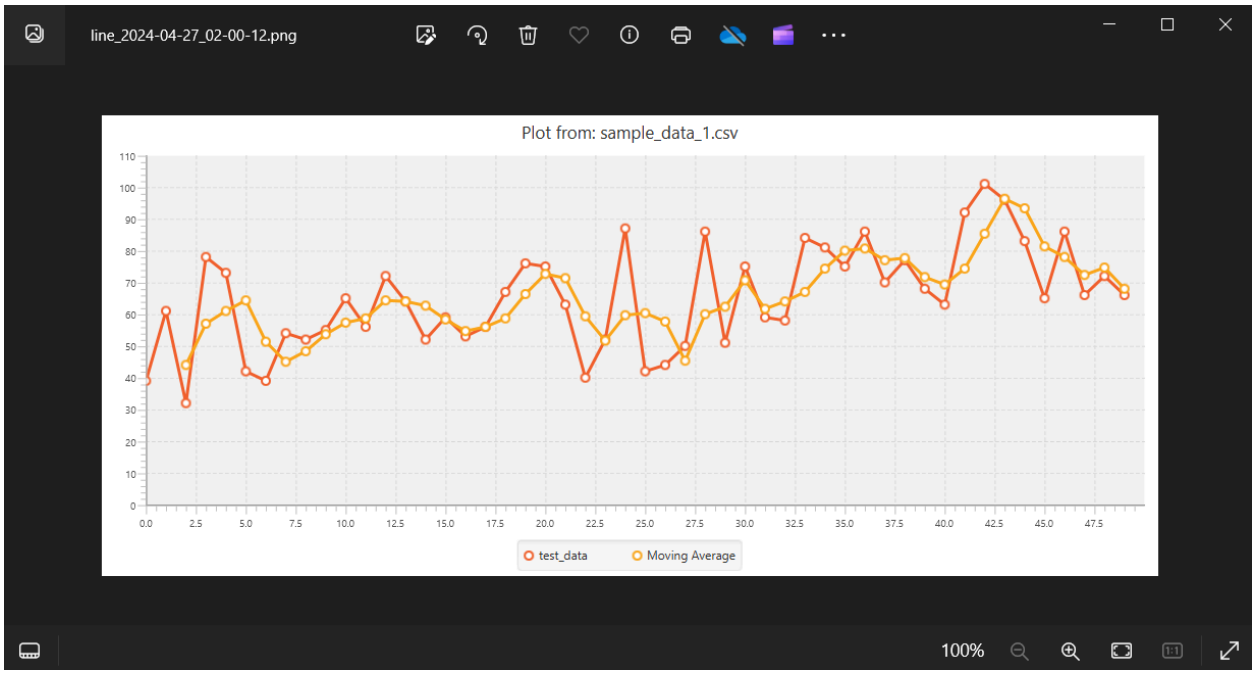
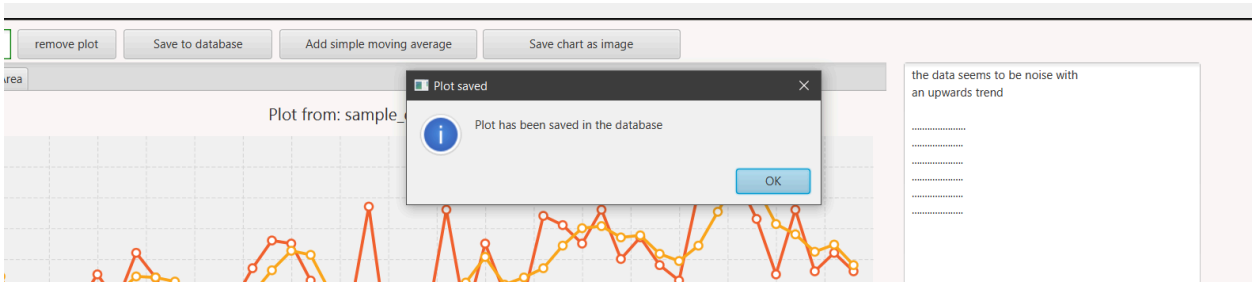


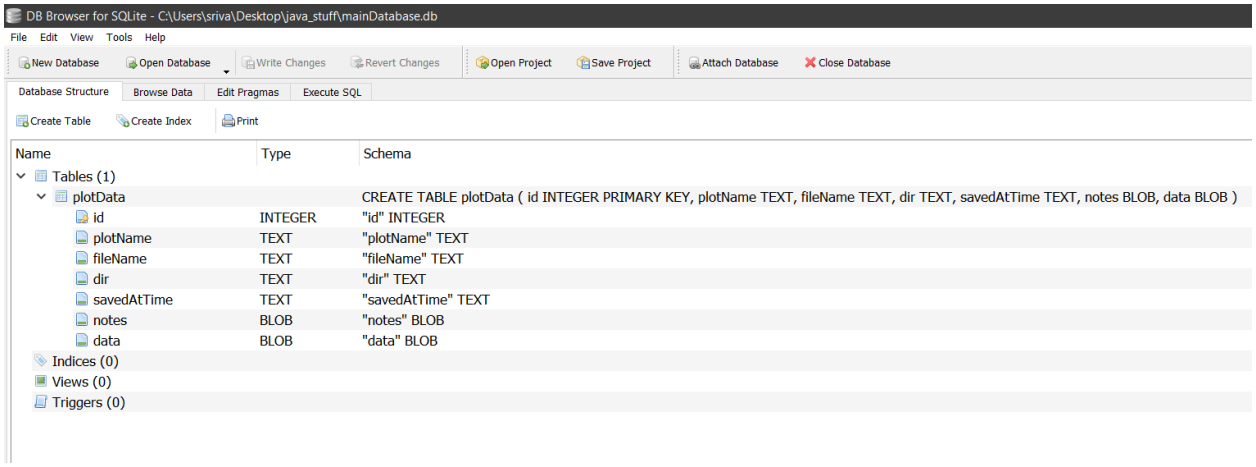
Image of the plot:



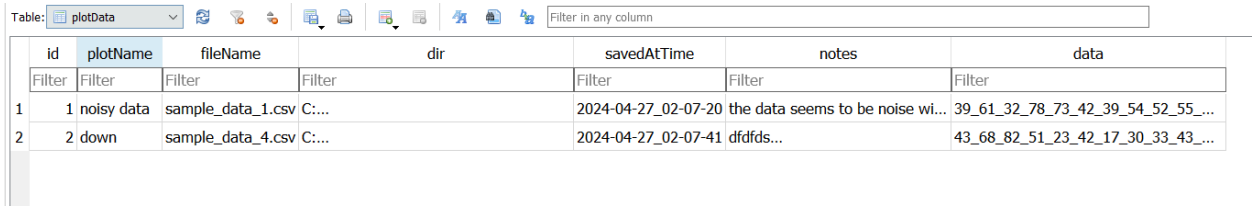
Saving the plot to a SQLite database:



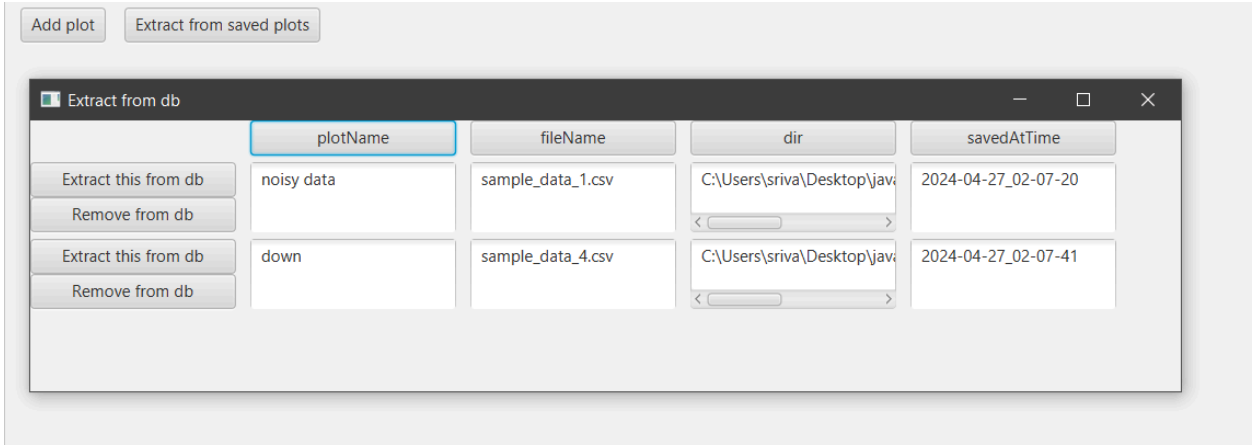
Database schema:



Database table showing the saved plots:



Extracting the saved plot from the database:



After extracting (the saved is plot is retrieved from the database including the notes/comments that the user had made):