

2025

IT 320 - Object-Oriented Programming Project

---

# MARKETING MIX MODELING DATA ANALYSIS DASHBOARD

---

AbdelRaouf Lakhoues - Becher Zribi - ElAmine  
Maaloul - Omar Allal - Yassine Marrekchi

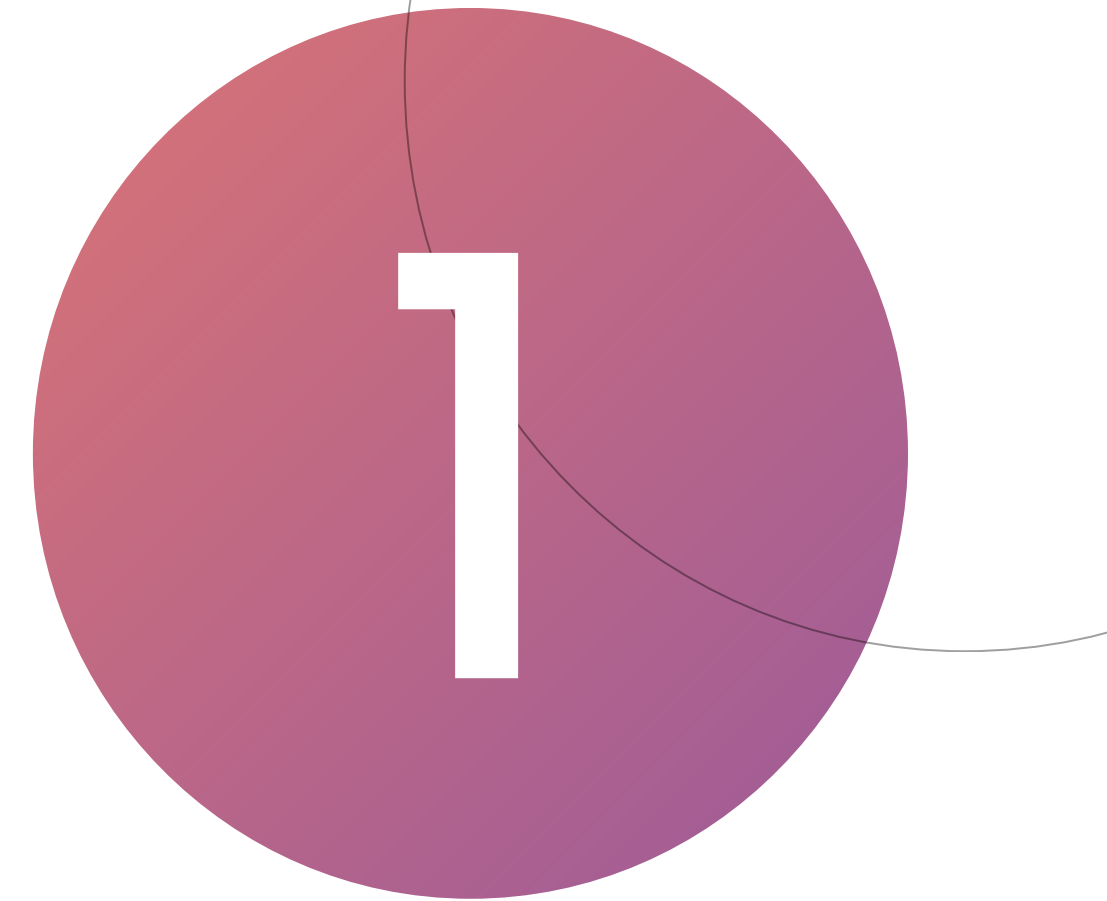
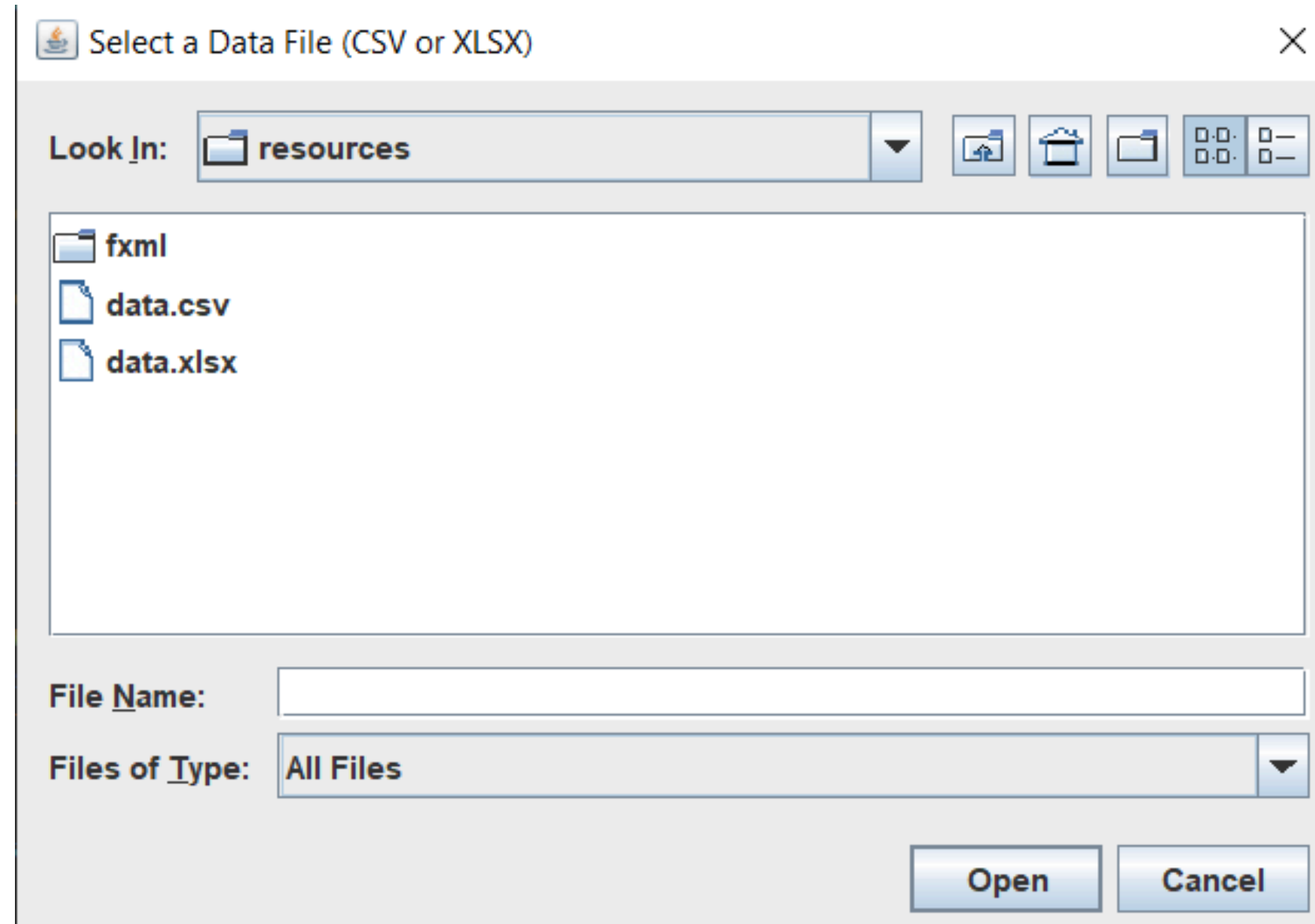
# P r o j e c t   d e s c r i p t i o n

**Marketing Mix Modeling (MMM)** involves various statistical techniques to quantify the **impact of marketing activities on sales** and KPIs, helping businesses **optimize strategies**

The project focuses on developing a **Marketing Mix Modeling (MMM) system** to **analyze marketing campaigns**, helping to assess the **impact of various marketing channels** and **improve decision-making for future campaigns**.

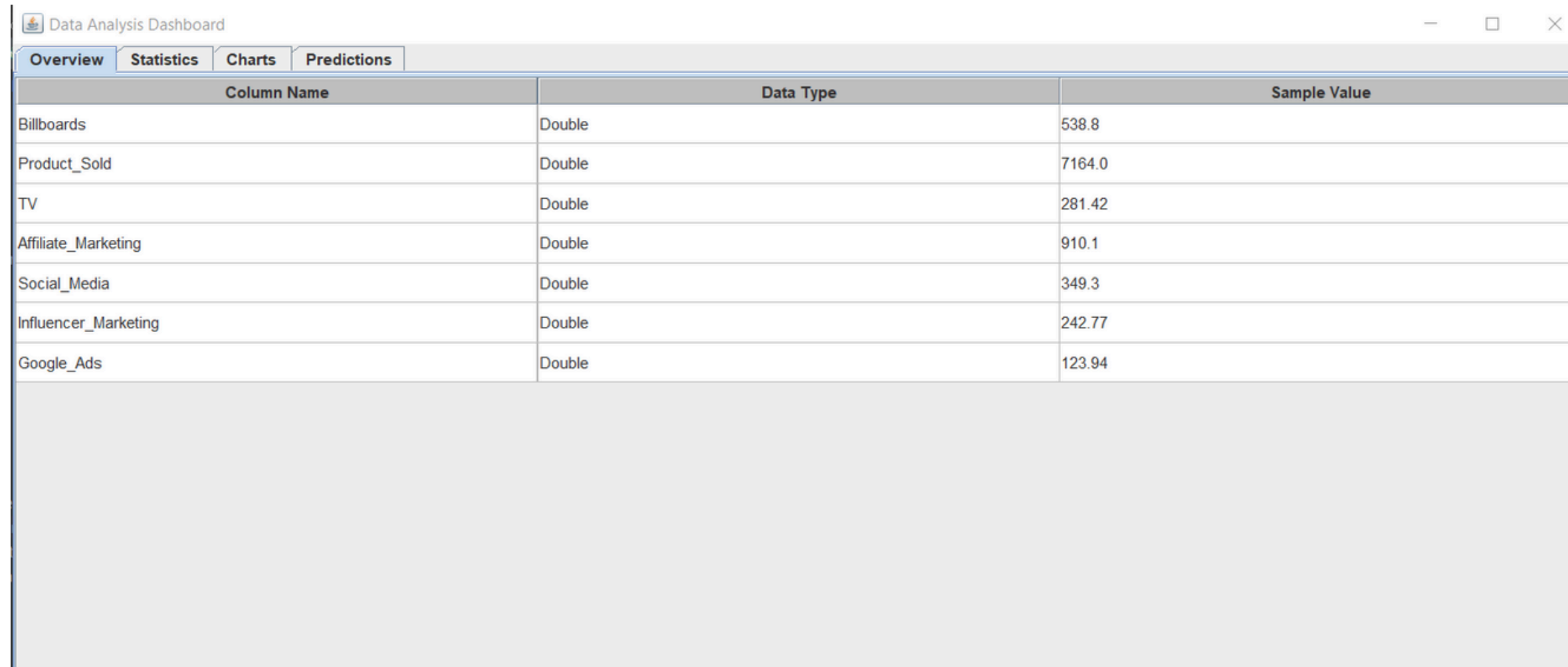
# PROJECT SHOWCASE

## = Choice of data



- At this phase user will have to choose the data file to be analyzed by the system
- This project supports two types of data :
  - .CSV
  - .xlsx

## = Overview of data



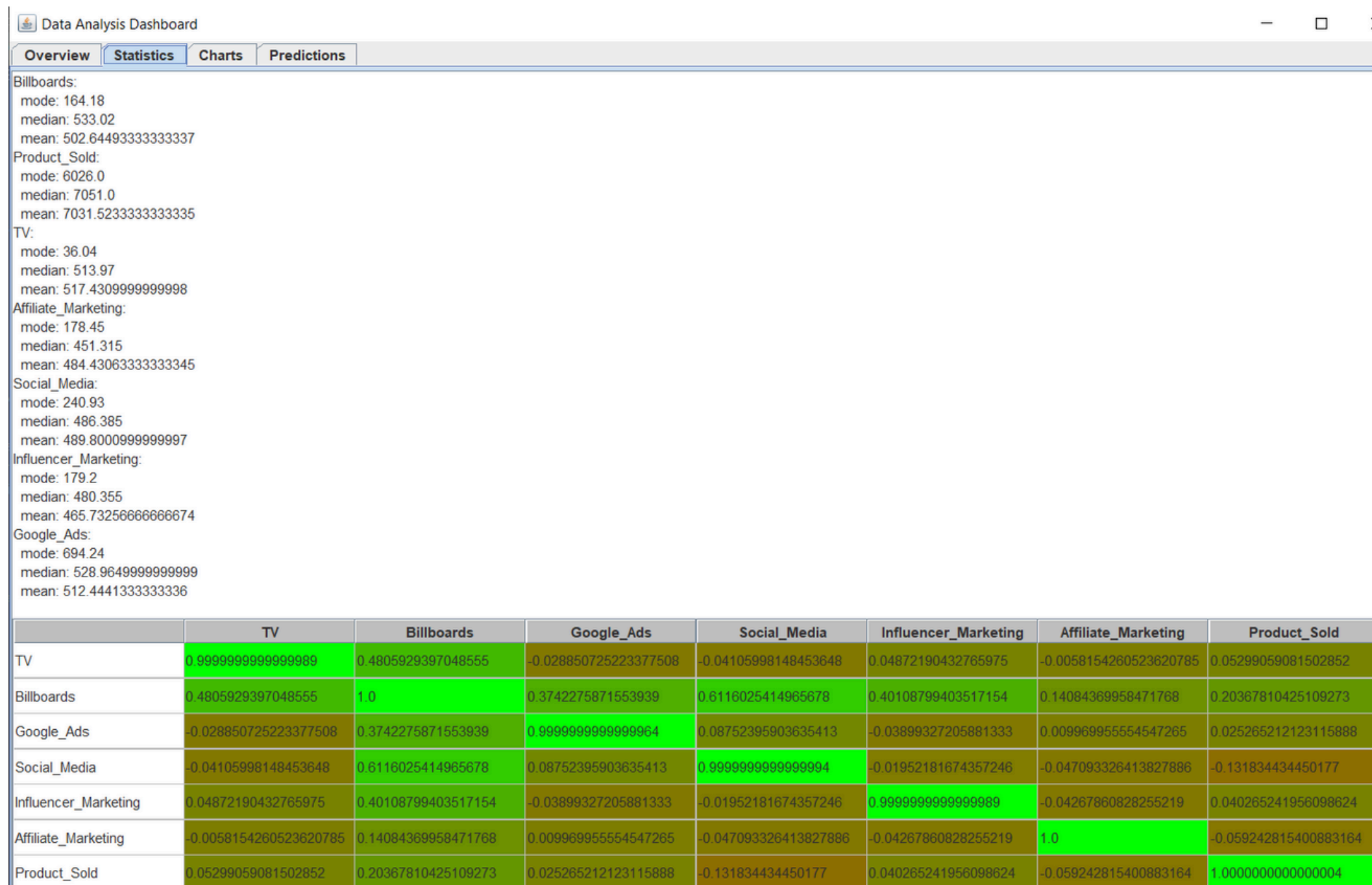
The screenshot shows a web application titled "Data Analysis Dashboard" with four tabs: Overview, Statistics, Charts, and Predictions. The Overview tab is active, displaying a table with three columns: Column Name, Data Type, and Sample Value. The table lists seven marketing channels, all with a data type of "Double".

Column Name	Data Type	Sample Value
Billboards	Double	538.8
Product_Sold	Double	7164.0
TV	Double	281.42
Affiliate_Marketing	Double	910.1
Social_Media	Double	349.3
Influencer_Marketing	Double	242.77
Google_Ads	Double	123.94

- Once data is chosen the Data Analysis Dashboard will be displayed showcasing the “Overview” Tab
- The Overview tab presents comprehensive information about the type of data contained in this project

2

# = Statistical & Correlation analysis

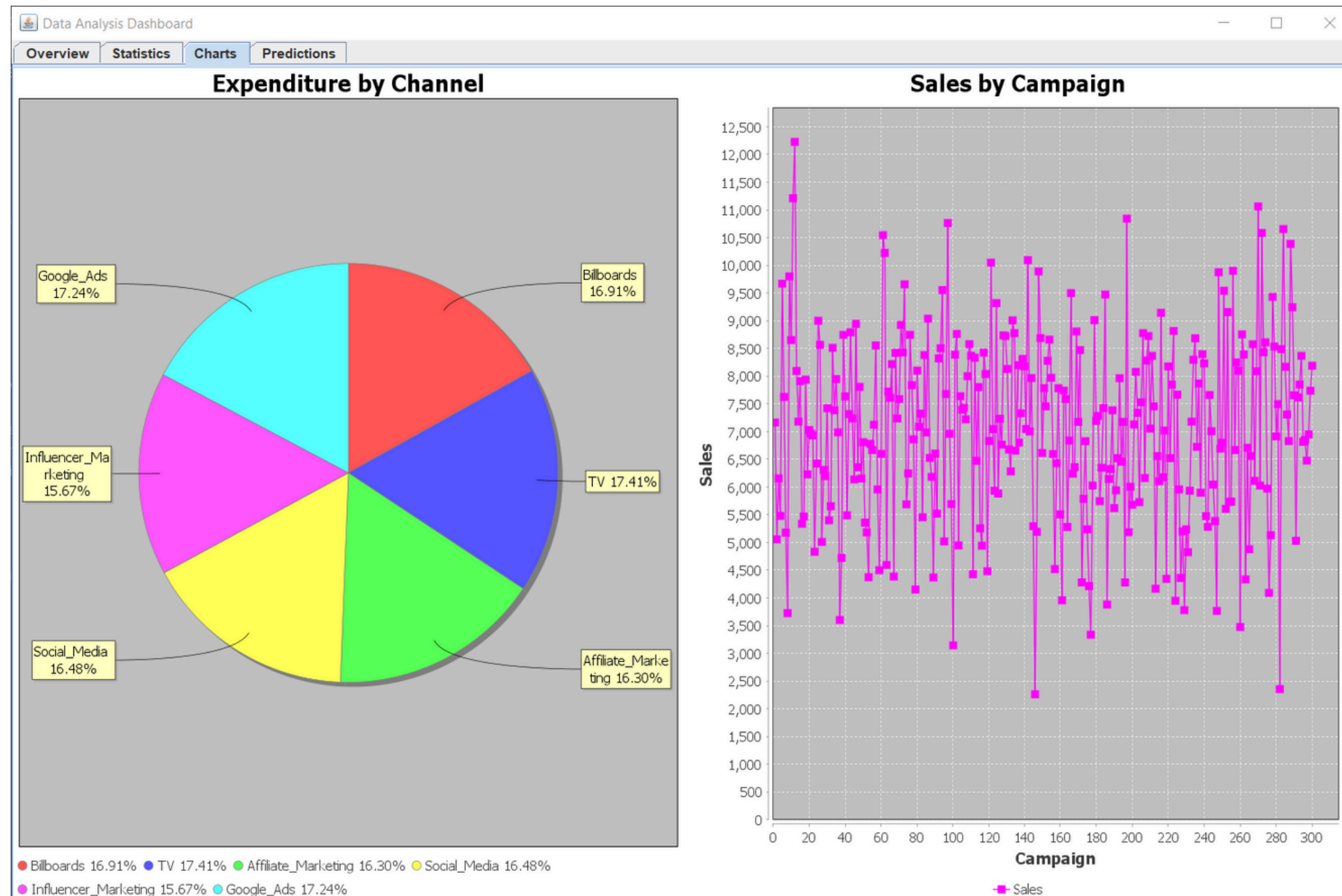


- Additionally, the data analysis dashboard provides statistical metrics related to the data
- Correlation Matrix is also presented in this tab in order to distinguish features relationships, which features causing multi collinearity ...



## Visualizations & EDA

- The Charts tab in the dashboard provides a graphical representation of marketing expenditure distribution and sales trends across campaigns



- Pie Chart: Expenditure Distribution:
  - Displays the percentage share of each advertising channel based on average total expenditure.
  - Helps identify dominant and underutilized channels.
- Line Chart: Sales by Campaign:
  - Tracks product sales across different campaigns.
  - Highlights trends, peaks, and underperforming campaigns.

4

# = Prediction

Data Analysis Dashboard								
Overview	Statistics	Charts	Predictions					
Campaign	Billboards	Product_Sold	TV	Affiliate_Marketing	Social_Media	Influencer_Marketing	Google_Ads	Predicted Sales
Campaign 1	538.8	7164.0	281.42	910.1	349.3	242.77	123.94	7164.0
Campaign 2	296.53	5055.0	702.97	132.43	180.55	781.06	558.13	5055.0
Campaign 3	295.94	6154.0	313.14	464.23	505.71	438.91	642.96	6154.0
Campaign 4	61.27	5480.0	898.52	432.27	240.93	278.96	548.73	5480.0
Campaign 5	550.72	9669.0	766.52	841.93	666.33	396.33	651.91	9669.0
Campaign 6	612.27	7627.0	507.13	965.77	142.96	171.79	230.67	7627.0
Campaign 7	555.02	5177.0	486.64	366.25	271.62	70.04	203.26	5177.0
Campaign 8	184.57	3726.0	762.09	251.74	97.85	116.67	176.61	3726.0
Campaign 9	778.31	9801.0	638.6	515.16	759.04	857.13	796.32	9801.0
Campaign 10	542.51	8652.0	591.48	931.65	329.15	577.38	400.23	8652.0
Campaign 11	577.85	11210.0	667.17	852.99	970.28	789.52	898.19	11210.0
Campaign 12	947.73	12227.0	787.33	742.36	992.3	871.55	884.15	12227.0
Campaign 13	865.19	8093.0	537.75	211.92	626.87	702.96	768.61	8093.0
Campaign 14	661.81	7181.0	735.78	585.74	102.96	588.45	282.0	7181.0
Campaign 15	207.04	7910.0	631.14	778.49	358.88	515.59	939.62	7910.0
Campaign 16	515.21	5337.0	334.12	271.07	153.75	685.88	565.22	5337.0
Campaign 17	558.81	5468.0	194.33	155.53	531.53	485.4	571.82	5468.0
Campaign 18	890.21	7937.0	955.97	307.14	609.33	62.99	343.96	7937.0
Campaign 19	188.8	6228.0	974.14	408.68	74.83	820.43	606.22	6228.0
Campaign 20	889.82	7026.0	93.47	710.91	31.38	178.92	702.23	7026.0
Campaign 21	958.04	6966.0	853.65	38.61	347.92	120.96	810.77	6966.0
Campaign 22	798.01	6929.0	79.78	399.95	252.3	680.24	881.29	6929.0
Campaign 23	121.5	4830.0	596.8	559.05	145.03	106.09	364.02	4830.0
Campaign 24	168.27	6425.0	297.32	636.42	968.13	238.44	38.43	6425.0



- The "Predictions" tab displays sales predictions based on a decision tree model. It shows:
  - Input Features: The original dataset columns used for prediction.
  - Predicted Output: The predicted sales values generated by the model.



# OOP PRINCIPLES IN ACTION

# Inheritance: Reusing and Extending Functionality

- Inheritance allows a child class to inherit attributes and methods from a parent class, promoting code reuse and extensibility. In this case:
  - **DataSource** (Parent Class): Defines a common structure for file reading.
  - **CSVReader** and **XLSXReader** (Child Classes): Extend DataSource and implement specific file-reading logic.

 DataSource.java:

```
1 package com.mmm.data;
2
3 import java.io.IOException;
4 import java.util.List;
5 import java.util.Map;
6 //inheritance
7 public abstract class DataSource {
8     protected String filePath;
9
10    public DataSource(String filePath) {
11        this.filePath = filePath;
12    }
13
14    // Abstract method to be implemented by subclasses
15    public abstract List<Map<String, String>> readData() throws IOException;
16 }
17
```

 CSVReader.java:

```
public class CSVReader extends DataSource {
    public CSVReader(String filePath) {
        super(filePath);
    }

    @Override
    public List<Map<String, String>> readData() throws IOException {
        List<Map<String, String>> records = new ArrayList<>();
        try (Reader reader = new FileReader(filePath)) {
            // CSV logic
        }
    }
}
```

 XLSXReader.java:

```
public class XLSXReader extends DataSource {
    public XLSXReader(String filePath) {
        super(filePath);
    }

    @Override
    public List<Map<String, String>> readData() throws IOException {
        List<Map<String, String>> records = new ArrayList<>();
        // XLSX logic
    }
}
```

Feature	DataSource (Parent Class)	CSVReader (Child Class)	XLSXReader (Child Class)
Defines filePath	✓ (Stores file path)	✓ (Uses super(filePath))	✓ (Uses super(filePath))
Implements readData()	✗ (Abstract method, forces implementation)	✓ (Processes CSV files)	✓ (Processes XLSX files)
Enforces structure	✓ (Standard method for all data sources)	✓ (Overrides readData() for CSV)	✓ (Overrides readData() for XLSX)
Avoids redundancy	✓ (Centralizes file handling logic)	✓ (No need to redefine filePath)	✓ (No need to redefine filePath)

## ≡ Inheritance: Reusing and Extending Functionality

The Model class implements the MachineLearningModel interface, meaning it inherits the required method signatures (train and predict) and provides concrete implementations for them.



MachineLearningModel.java:

INTERFACE

```
//inheritance
public interface MachineLearningModel {
    void train(List<Map<String, String>> trainData);
    List<Map<String, String>> predict(List<Map<String, String>> testData);
}
```

### Implementation of MachineLearningModel



Model.java:

CLASS

```
public class Model implements MachineLearningModel{
    //inheritance
    private J48 decisionTree;
    private Instances trainingInstances;
    private List<Attribute> attributeList = new ArrayList<>();
    private Map<String, List<String>> categoryValues = new HashMap<>();
}
```

- The Model class implements the MachineLearningModel interface, meaning it inherits the required method signatures (train and predict) and provides concrete implementations for them.

## Polymorphism: Flexibility through Multiple

Polymorphism allows methods to perform differently based on the object that invokes them

 Visualization.java:

```
//polymorphism
public static JPanel createPieChart(java.util.List<java.util.Map<String, String>> data) {
```

```
//polymorphism
public static JPanel createLineChart(java.util.List<java.util.Map<String, String>> data) {
    XYSeries series = new XYSeries("Sales");
    int i= 1;
```

```
public static void displayChart(JPanel chartPanel, String title){
    JFrame frame = new JFrame(title);
    frame.add(chartPanel);
    frame.setSize(800, 600);
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}
```

- The createPieChart and createLineChart methods return a JPanel, but the actual type of chart (pie or line) depends on the method used.

- The displayChart method works polymorphically with any type of JPanel, demonstrating how polymorphism simplifies the visualization logic.

## ⌵ Encapsulation: Securing and Simplifying Access

Encapsulation ensures that internal details of a class are hidden from the outside and accessed only through methods.

 Model.java:

```
private J48 decisionTree;  
private Instances trainingInstances;  
private List<Attribute> attributeList = new ArrayList<>();  
private Map<String, List<String>> categoryValues = new HashMap<>();
```

 DataPreprocessor.java:

```
// encapsulation  
private static java.util.List<java.util.Map<String, String>> removeDuplicates(java.util.List<java.util.Map<String, String>> data) {  
    return new ArrayList<>(new LinkedHashSet<>(data));  
}
```

- The decisionTree, trainingInstances, attributeList, and categoryValues variables are marked private, meaning they can't be accessed directly from outside the class.
- Instead, they are managed through methods like train and createInstances.
- The removeDuplicates method is marked private, meaning it can only be accessed within the DataPreprocessor class. This restricts access and protects the internal logic.



# ≡ Abstraction: Simplifying Complex Systems

Abstraction hides the implementation details and only exposes essential functionality.

 MachineLearningModel.java:

```
1 package com.mmm.model;
2
3 import java.util.List;
4
5
6 public interface MachineLearningModel {
7     void train(List<Map<String, String>> trainData);
8     List<Map<String, String>> predict(List<Map<String, String>> testData);
9 }
```

- The MachineLearningModel interface defines the structure (method signatures) without providing implementation details. Classes like Model provide concrete implementations.

 Dashboard.java:

```
//abstraction
private JPanel createChartsPanel() {
    JPanel chartsPanel = new JPanel();
    chartsPanel.setLayout(new GridLayout(1, 2));
}

//abstraction
private JPanel createPredictionPanel() {
    JPanel predictionPanel = new JPanel(new BorderLayout());

    if (predictions == null || predictions.isEmpty()) {
        JLabel noDataLabel = new JLabel("No Prediction Data Available.");
        noDataLabel.setHorizontalAlignment(SwingConstants.CENTER);
        predictionPanel.add(noDataLabel, BorderLayout.CENTER);
    } else {

```

- the following dashboard methods (createChartsPanel(), createPredictionPanel()...) abstracts the implementation details of creating specific tabs. The initializeUI method calls these methods to compose the dashboard without needing to know the internal logic of each panel.