TUNIS BUSINESS SCHOOL
UNIVERSITY OF TUNIS

Climabix

IT320 : Object Oriented Programming
Professor : Ameni Azzouz

Mini-project
by

Yassine Chebbi
Ranim Gsouri

BA/IT FIN/IT SENIOR STUDENTS

January 2025

Table of Contents

Chapter 1

Introduction : Weather and Accident Data Analysis For Future predictions...

Welcome to our project. The weather and accident data anaysis project focuses on developing a comprehensive system for data analysis, prediction, and user interaction, incorporating various technologies and methodologies. Starting with weather and accident data, we conducted an in-depth analysis to uncover relationships and correlations between variables such as weather conditions (visibility, wind speed, temperature, pressure) and accident severity.

Through visualization tools and regression models, we aimed to derive insights and make predictions about accident outcomes based on environmental factors. The project also included data ingestion pipelines to clean, manage, and process multiple datasets efficiently. To enhance usability, we implemented a user management system using abstraction and polymorphism to define distinct roles such as administrators, data scientists, and viewers, each with tailored access to components like visualization tools and data cleaning modules.

The system was further developed into a full-fledged web application using Spring Boot, with a basic front-end built using HTML, CSS, and JavaScript. This front-end allows users to interact with the system intuitively, managing roles, performing operations. The purpose of this project is to create a scalable and user-friendly platform for analyzing, predicting, and interacting with data to support informed decision-making and risk management.

Chapter 2

Project Description

2.1 Data Gathering

Five databases were gathered from different sources.

The types of data that were used are: csv files and Maven-supported databases

1- Tunis Carthage Weather.csv

2- London Weather.csv

3- Lieux.csv

4- Vehīcules.csv̄

5- Usagers.csv

The following section will detail the steps of data preparation and analysis.

2.2 Data Ingestion and Cleaning

2.2.1 Data Ingestion Project

- **Interface Design:**

- Created a DataProcessor interface that defines the method readAndIngest(String filePath) to standardize how data ingestion is handled for different data types.

- **Specialized Processors:**

- Implemented two classes, WeatherDataProcessor and AccidentDataProcessor, which provide specific functionality for weather and accident data ingestion respectively.

- **File Reading and Ingestion:**

- The readAndIngest() method in both processor classes is invoked to simulate reading and processing data from the specified file paths.

- **Dynamic File Handling:**

- Arrays (weatherFiles and accidentFiles) store file paths for weather and accident data.
- A loop iterates through each file path, calling the readAndIngest() method for each.

- **Column Display:**

- A helper method, displayColumnNames(String filePath), is implemented to extract and display column names (header row) from the CSV files.
- This method reads the first line of each file and prints the column names, assisting in understanding the data structure.

2.2.2 Data Cleaning Project:

2.2.2.1 Dependencies and Maven Configuration :

To implement the data cleaning functionalities, Maven was used as the build management tool. The required dependencies were added to ensure access to relevant libraries, particularly for handling CSV files.

2.2.2.2 Overview of the Data Cleaning Process :

The goal of the data cleaning project is to process raw CSV files, identify issues such as missing data or outliers, and generate cleaned datasets for further analysis. Below are the steps involved:

2.2.2.3 Steps and Corresponding Methods :

- **Remove Empty Columns :**

- Description: Identifies completely empty columns in the dataset and removes them to create a compact, clean dataset.
- Method Names in Code:

findEmptyColumns(List<String[]> data) – Finds empty columns.

removeEmptyColumns(String[] row, List<Integer> emptyColumnIndexes) – Removes the identified empty columns.

- **Clean NaN Values :**

- Description: Replaces all NaN values in the dataset with either an empty string or a default value for consistency.
- Method Name in Code: cleanNaNValues(List<String[]> data)

- **Detect and Remove Outliers :**

- Description: Identifies rows with outliers in numeric columns and removes them based on predefined thresholds.
- Method Names in Code:

identifyNumericColumns(List<String[]> data) – Identifies numeric columns in the dataset.

removeOutliers(List<String[]> data, List<Integer> numericColumns) – Removes rows containing outlier values.

2.3 Data Analysis :

2.3.1 Statistical Analysis:

2.3.1.1 Central Tendency Measures :

- Mean: The arithmetic average of a column, indicating the central value of the data.
- Median: The middle value in a sorted column, showing the central position unaffected by outliers.

Dataset	Column	Mean	Median	Observations
Cleaned_vehicules.csv	choc	2.93	2.00	Most accidents involve moderate initial shocks, centered around 2.
Cleaned_lieux.csv	prof	1.24	1.00	Indicates that most accident-prone roads are flat or have shallow slopes.
Cleaned_London Weather.csv	sea_level	0.11	0.03	Sea level variations are minimal, suggesting stable weather conditions.
Cleaned_Tunis Carthage Weather.csv	Temp_C	8.80	9.30	Temperatures are skewed slightly higher, with a median above the mean.
Cleaned_usagers.csv	place	5.93	7.00	Position 7 is the most frequently occupied in accidents, indicating vulnerability.

Figure 2.3.1.1 : Central tendancy measures calculations and observations

2.3.1.2 : Observations and Insights

- Vehicle Impact Analysis (choc):

The mean (2.93) and median (2.00) indicate that most collisions involve moderate initial shocks.

Indicates that rear-end and frontal collisions are the most common accident types.

- Road Profile Analysis (prof):

A mean of 1.24 and a median of 1 suggest that accidents predominantly occur on flat roads.

Sloped and hilltop areas are less frequently associated with accidents.

- London Weather (sea_level):

Sea level changes are minimal (mean: 0.11), suggesting a lack of extreme atmospheric conditions.

This stability could explain the relatively consistent weather's impact on road safety.

- Tunis Carthage Weather (Temp_C):

Mean (8.80°C) and median (9.30°C) suggest slightly warmer conditions.

The higher median implies temperatures are often on the warmer side, which may influence driver performance and accident rates.

- User Behavior (place):

Position 7 in vehicles appears to be the most vulnerable in accidents, warranting further investigation into vehicle safety measures.

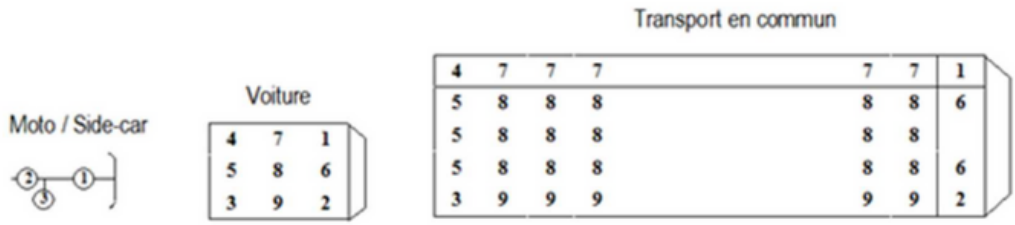


Figure 2.3.1.2: Place Occupied by the passenger in the vehicule

2.3.2 Correlation Analysis:

2.3.2.1 : Pearson Correlation :

- **Implementation Details**

- Class: PearsonCorrelation
- Method: calculateCorrelation(List<Double> xData, List<Double> yData)
- The formula used :

$$r = \frac{n\sum XY - \sum X \sum Y}{\sqrt{(n\sum X^2 - (\sum X))(n\sum Y^2 - (\sum Y))}}$$

- Validates input sizes and handles division-by-zero errors and efficiently computes correlations for datasets of any size.

- **Example Calculations :**

1. Correlation between sea_level and wind_speed:

Output: Pearson correlation: 0.9413.

Empirical Validation:

Real-life empirical relationship: $h = 0.003v^2$ where hhh is the sea level and vvv is the wind speed.

Data-derived relationship: $h = 0.00258967v^2$ (from averaging across the dataset).

Accuracy of the dataset:

$$\text{Accuracy} = \frac{\textit{Data-derived coefficient}}{\textit{Empirical coefficient}} \times 100 = \frac{0.00258967}{0.003} = 91.32\,\%$$

This validates that the dataset reflects real-world relationships with over 91% accuracy.

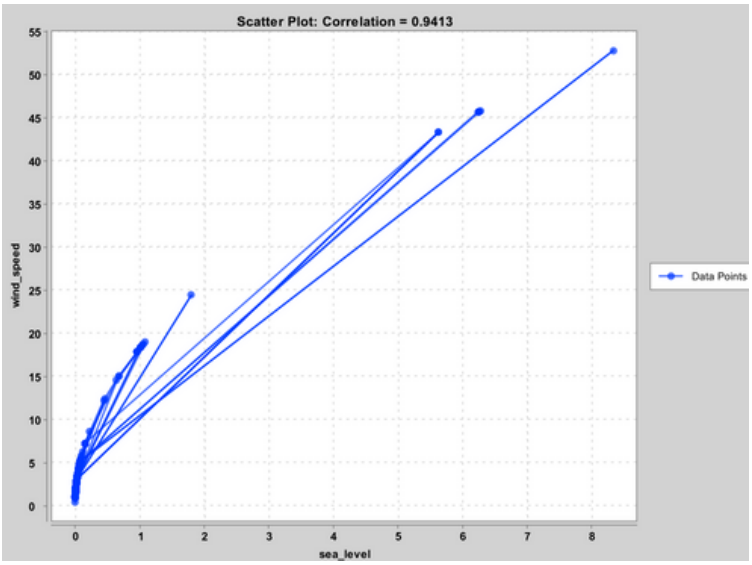


Figure 2.3.2.1.1: Correlation of sea level and wind speed

2. Correlation between Visibility_km and Press_kPa:

Output: Pearson correlation: -0.8915.

Interpretation:

Strong negative correlation suggests that as atmospheric pressure increases, visibility decreases.

This is consistent with real-world observations where high-pressure systems are often associated with conditions like haze or fog, reducing visibility.

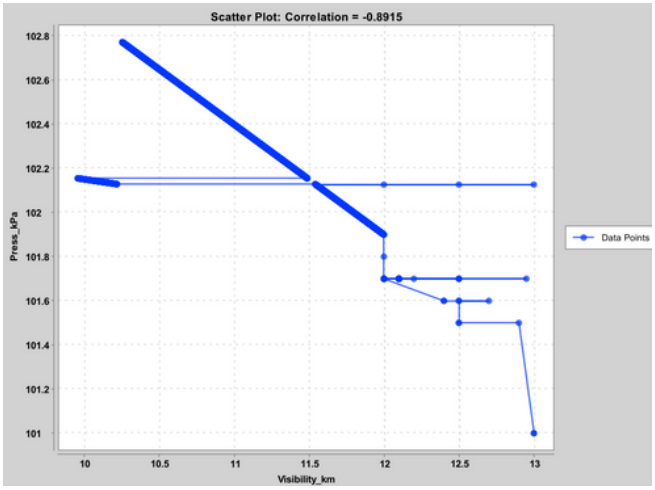


Figure 2.3.2.1.2 Visibility km vs Press kPa correlation

2.3.2.2 : Spearman Correlation :

- **Implementation Details**

- Class: PearsonCorrelation
- Method: calculateCorrelation(List<Double> xData, List<Double> yData)
- The formula used :

$$r_s = 1 - \frac{6\sum d_i^2}{n(n^2-1}$$

- **Example Calculations :**

Correlation between grav (severity of injury) and place (position occupied in the vehicle):

Output: Spearman correlation: 0.0037

Interpretation:

Near-zero correlation indicates no meaningful monotonic relationship between the severity of injuries and the position occupied by users in vehicles.

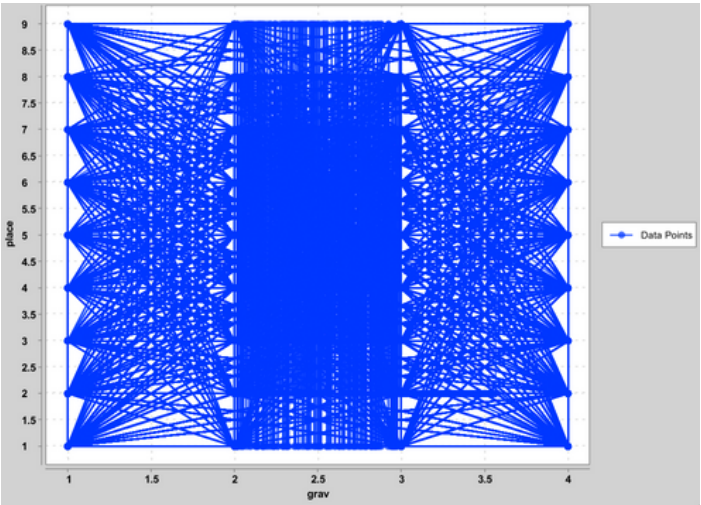


Figure 2.3.2.2 grav vs place correlation

Variables	Correlation Type	Value	Observation
sea_level vs. wind_speed	Pearson	0.9413	Strong positive correlation; dataset reflects a 91% accurate empirical relationship $h = 0.00258967v^2$
Visibility_km vs. Press_kPa	Pearson	-0.8915	Strong negative correlation; as pressure increases, visibility decreases.
grav vs. place	Spearman	0.0037	No meaningful monotonic relationship between injury severity and vehicle position.

2.3.2.3 : Methods in Correlation Analysis :and interpretations:

1. processCorrelation(String filePath, PearsonCorrelation correlation):

2.processSpearmanCorrelation(String filePath, String variableX, String variableY):

3.generateScatterPlot:

- Sea level and wind speed: The dataset shows strong alignment with the real-world empirical relationship, with 91% accuracy.
- Visibility and pressure: A strong negative correlation aligns with physical atmospheric phenomena.
- Severity and position: No significant correlation found, suggesting other factors influence injury severity.

2.3.3 Visualization:

- **Implementation Details**

GaussianCurveVisualization

Methods:

1. createVisualization():

Creates a histogram dataset to represent frequency distributions.

Overlays a Gaussian curve based on the calculated mean and standard deviation.

Saves the visualization as a PNG image (e.g., choc_gaussian_curve.png).

2. displayVisualization():

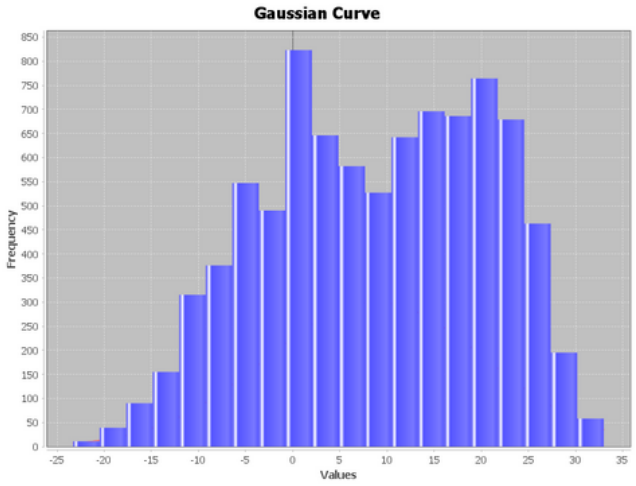
Prints the confirmation of the created visualization.

- Temperature (Temp_C):

File: Cleaned_Tunis Carthage Weather.csv

Output: Gaussian curve for temperature saved as Temp_C_gaussian_curve.png.

Observation: Temperatures are slightly skewed toward warmer conditions.

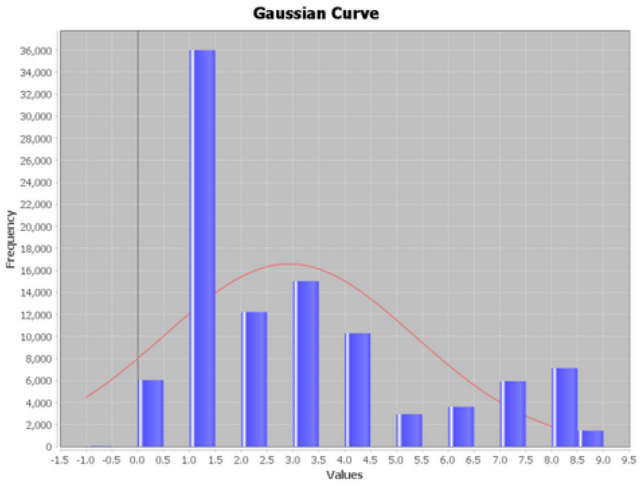


- Impact Severity (choc):

File: Cleaned_vehicules.csv

Output: Gaussian curve for impact severity saved as choc_gaussian_curve.png.

Observation: Most accidents show a central distribution around moderate impact levels.



PieChartVisualization

Methods:

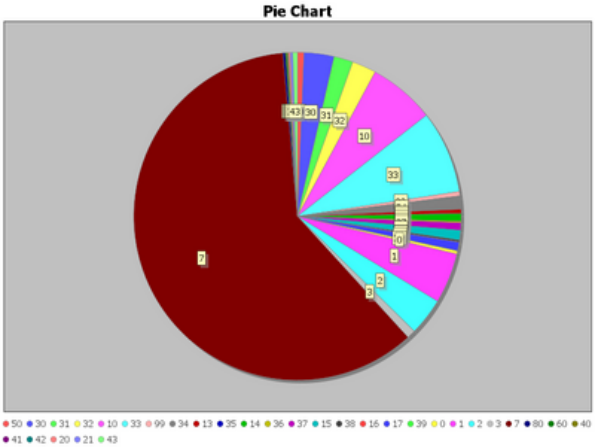
1. createVisualization():
   - Creates a pie chart dataset using DefaultPieDataset.
   - Plots the chart using ChartFactory.createPieChart.
   - Saves the chart as a PNG image (e.g., catv_pie_chart.png).
2. displayVisualization():
   - Prints a confirmation of the created visualization.

- Vehicle Category (catv):

File: Cleaned_vehicules.csv

Output: Pie chart saved as catv_pie_chart.png.

Observation: Vehicle type 7 (light vehicles) dominates the dataset.



The visualizations created provide critical insights into the dataset. Gaussian curves highlight the distribution and variability of numerical data, allowing us to identify central tendencies and outliers. For example, the Gaussian curve for the choc variable (saved as choc_gaussian_curve.png) from Cleaned_vehicules.csv reveals a centralized distribution around moderate accident impacts, while the curve for Temp_C (saved as Temp_C_gaussian_curve.png) from Cleaned_Tunis Carthage Weather.csv shows a slight skew toward warmer temperatures. Additionally, pie charts effectively depict the dominance of specific categories within the data. The pie chart for the catv variable (saved as catv_pie_chart.png) from Cleaned_vehicules.csv illustrates that light vehicles (category 7) are the most frequently involved in accidents. Together, these visualizations enhance the understanding of the dataset and provide a solid foundation for interpreting relationships and trends within the data.

2.3.3 Regression Analysis:

- **Linear Regression**

- Linear regression estimates the relationship between one dependent variable (grav) and one or more independent variables (e.g., temp, wind_speed, visibility, pressure).
- The regression equation is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p + \varrho$$

where:

- y: Dependent variable (e.g., grav - accident severity).
- xi: Independent variables (e.g., weather conditions).
- βi: Coefficients representing the impact of xi.
- ϵ: Error term.

- **Code Implementation**

1. Data Preprocessing:

- Class: DataPreprocessor.
- Method: preprocessData(String filePath, String[] selectedColumns, int rowLimit):
  - Reads the specified columns (e.g., grav, pressure, temp) from the input files.
  - Handles missing or invalid data.
  - Returns processed data as a list of rows.

2. Regression Analysis:

- Class: GravWeatherAnalysis.
- Method: analyzeGravWeather(List<Double> grav, List<List<Double>> weatherData):
  - Uses OLS (Ordinary Least Squares) regression to analyze the relationship between grav (dependent variable) and combined weather variables (independent variables).
  - Outputs regression coefficients, quantifying the impact of each weather variable on accident severity.

Variable	Coefficient	Interpretation
Intercept	396.1102	Baseline severity when all variables are at zero.
pressure	0.0091	A slight positive effect: higher pressure correlates with increased severity.
temp	0.0632	Warmer temperatures are associated with a moderate increase in severity.
sea_level	0.0592	Higher sea levels are correlated with increased accident severity.
wind_speed	-0.0035	Minimal negative impact: higher wind speeds slightly reduce severity.
Visibility_km	-0.7713	Significant negative impact: better visibility reduces accident severity.
Press_kPa	-4.0588	Significant negative impact: higher local pressure correlates with reduced severity.

2.3.4 Future Prediction Using Weather and Accident Data:

Objective

Predict future accident severity using ARIMA forecasts for weather variables and the regression model.

Prediction Process:

1. Weather Variable Forecast:

   Use ARIMA to predict future values for temp, visibility, pressure, and sea_level.

2. Regression Application:

   Input the predicted weather values into the regression model:

$$Severity = 396.1102 + (0.0091 \times pressure) + (0.00632 \times temp) + (0.0592 \times sea\ level) + (-0.0035 \times wind\ speed) + (-4.0588 \times Press\ kPa)$$

Example Prediction:

1. Predicted Weather Conditions:
2. pressure: 1015 hPa
3. temp: 22°C
4. sea_level: 1.2 m
5. wind_speed: 15 km/h
6. Visibility_km: 6
7. Press_kPa: 101.3 kPa
8. Predicted Severity:

Result: Severity = 2.54 (suggesting moderate injury levels).

2.4 OOP Concepts intergration :

2.4.1 User Access System: Explanation and OOP Concepts:

The User Access System is designed to manage and control access to system components based on user roles. Each role (e.g., Administrator, Data Scientist, User, Viewer) has distinct permissions, and the system ensures only authorized users can perform specific actions. Additionally, the system incorporates a notification mechanism to inform users about critical updates, using different communication methods (e.g., Email, SMS, Push Notifications).

- **Key Components :**

1. User Roles and Access Control

- Core Classes:
  - Accessible Interface:
    - Defines the contract for accessing components (accessComponent(String component)).
    - Ensures all implementing classes provide their own logic for access control.
  - Person Abstract Class:
    - Serves as the base class for all user roles (Administrator, Data Scientist, User, Viewer).
    - Encapsulates common attributes (id, name, email) with appropriate getter methods.
    - Provides shared behaviors like login and logout.
  - Specialized Roles:
    - Administrator:
      - Inherits from Person and provides unrestricted access to all components.
    - DataScientist:
      - Grants access only to specific components, such as Data Cleaner and Data Ingestion, while denying others.
    - User:
      - Restricted to accessing the VisualizationTool component.
    - Viewer:
      - Similar to User, with access limited to VisualizationTool

2. Notification System

- Core Classes:
  - Notification Abstract Class:
    - Defines the shared structure for all notification types with attributes recipient and message.
    - Implements the send method, which is abstract and must be overridden by
  - Specialized Notifications:
    - EmailNotification.
    - SMSNotification.
    - PushNotification.
  - NotificationManager:
    - Centralized class for managing notifications.
    - Demonstrates polymorphism by dynamically calling the send method based on the notification type.

- **Object-Oriented Programming Concepts :**

1. Encapsulation:

- Implementation:
  - Person class encapsulates shared attributes (id, name, email) and provides controlled access through getter methods.
  - Notification subclasses (EmailNotification, SMSNotification, etc.) encapsulate their specific behavior while exposing only the send method for execution.
- Advantages:
  - Data is protected from direct modification.
  - Implementation details are hidden, ensuring better maintainability and security.

2. Inheritance:

- Implementation:
  - Person is the parent class for Administrator, DataScientist, User, and Viewer, enabling shared behavior like login and logout.
  - Notification is the base class for EmailNotification, SMSNotification, and PushNotification, promoting reuse of shared attributes and behaviors.
- Advantages:
  - Reduces code duplication by reusing common methods and attributes.
  - Promotes scalability by allowing easy addition of new user roles or notification types.

3. Polymorphism:

- Implementation:
  - Dynamic Method Dispatch:
    - The accessComponent method in Administrator, DataScientist, User, and Viewer is overridden to provide role-specific access behavior.
  - Notification System:
    - The NotificationManager class uses polymorphism to call the appropriate send method for EmailNotification, SMSNotification, or PushNotification dynamically.

4. Abstraction :

- Implementation:
  - Person Class:
    - Abstract class that defines the common structure for all user roles while leaving role-specific behaviors to subclasses.
  - Notification Class:
    - Abstract class that enforces the implementation of the send method in all notification types.
- Advantages:
  - Provides a blueprint for subclasses, ensuring consistency while promoting flexibility.

OOP Concept	Where Applied
Encapsulation	Person and Notification classes encapsulate attributes and behaviors.
Inheritance	Administrator, DataScientist, User, and Viewer inherit from Person.
Polymorphism	Dynamic invocation of accessComponent and send methods based on object type.
Abstraction	Abstract classes (Person, Notification) define structure for all subclasses.

2.4.1 OOP Concepts: Data Ingestion and cleaning :

1. Encapsulation:

- AccidentDataProcessor and WeatherDataProcessor:
  - Encapsulate the logic for reading and ingesting accident and weather data in the readAndIngest(String filePath) method.
  - Use the DataProcessor interface to enforce consistent access to the readAndIngest method across implementations.
- DataCleaner:
  - Encapsulates multiple data cleaning functionalities (e.g., removing empty columns, handling NaN values, detecting outliers) in private helper methods like findEmptyColumns, removeEmptyColumns, cleanNaNValues, and removeOutliers.
  - Methods are modular, each handling a specific task, ensuring maintainability.
- Main Class:
  - Encapsulates the workflow of processing files by instantiating WeatherDataProcessor and AccidentDataProcessor, and iterating through file paths to process data

2. Inheritance:

- AccidentDataProcessor and WeatherDataProcessor:
  - Both classes implement the DataProcessor interface, inheriting the contract for the readAndIngest method.
  - Enables polymorphism where objects of these classes are treated as DataProcessor types.

3. Polymorphism:

- Interface Polymorphism:
  - The DataProcessor interface defines the readAndIngest method, which is overridden in both AccidentDataProcessor and WeatherDataProcessor to provide specific implementations for accident and weather data processing.
  - In the Main class, DataProcessor objects (weatherProcessor, accidentProcessor) dynamically call their respective implementations based on the actual class type at runtime.

4. Abstraction :

- DataProcessor Interface:
  - Provides an abstraction for data processing by defining the readAndIngest method without specifying how it should be implemented.
  - Concrete implementations (AccidentDataProcessor, WeatherDataProcessor) provide the specific behavior for the abstract method.

- Data Cleaning in DataCleaner::
  - Abstracts complex operations (e.g., handling NaN values, detecting outliers) through method calls like cleanData without exposing the internal details to the caller

2.4.1 OOP Concepts: Data Analysis :

2.4.1.1 : Correlation :

1. Inheritance

- Abstract Base Class (Correlation):
  - Purpose: The Correlation class serves as the base class for all correlation types.
  - Attributes :
    - variableX and variableY: Represent the two variables for which the correlation is calculated.
  - Methods:
    - Abstract method calculateCorrelation: Must be implemented by subclasses.
    - Getters (getVariableX, getVariableY) to access the variable names.

- Concrete Subclasses:
  - PearsonCorrelation
  - SpearmanCorrelation
  - Specialized Correlation Classes :
    - CorrelationBetweenVar1Var2 and CorrelationBetweenVar3Var4 extend PearsonCorrelation to define specific variable pairs (sea_level vs. wind_speed and Visibility_km vs. Press_kPa)

2. Polymorphism :

- Method Overriding:
  - Subclasses (PearsonCorrelation, SpearmanCorrelation) override the abstract calculateCorrelation method to provide specific implementations
- Dynamic Dispatch:
  - In the Main class, the processCorrelation method accepts a PearsonCorrelation object, which could be any subclass of Correlation.
  - This allows the same method to handle both Pearson and Spearman correlation objects dynamically without modifying the method implementation.
- Example:

```
processCorrelation(filePath, new CorrelationBetweenVar1Var2());
processCorrelation(filePath, new CorrelationBetweenVar3Var4());
```

OOP Concept	Where Applied
Inheritance	PearsonCorrelation and SpearmanCorrelation inherit from Correlation.
Polymorphism	Dynamic invocation of calculateCorrelation in the processCorrelation method.
Encapsulation	Attributes variableX and variableY encapsulated in the Correlation class.
Abstraction	Correlation is an abstract class, enforcing the implementation of calculateCorrelation.

2.4.1.2 : Visualization :

1. Polymorphism :

Here, Polymorphism is extensively used to dynamically handle different types of visualizations (

Gaussian curves and Bar Charts , pie charts) through a common interface.

- Interface Polymorphism:
  - The VisualizationTool interface defines the contract for all visualizations with methods createVisualization and displayVisualization
  - Concrete classes (GaussianCurveVisualization, PieChartVisualization) implement this interface, providing their specific implementations

- Dynamic Dispatch:
  - In the Main class, objects of type VisualizationTool are created dynamically as either GaussianCurveVisualization or PieChartVisualization based on the visualization type

- Example:

```
VisualizationTool gaussianVisualization = new GaussianCurveVisualization(values,
    outputFileName);
gaussianVisualization.createVisualization();
gaussianVisualization.displayVisualization();
```

- The method createVisualization is called dynamically, invoking the specific implementation depending on the object type.

2. Inheritance:

- Interface Inheritance:
  - The VisualizationTool interface acts as the parent for all visualization classes, enforcing consistency across different implementations.
- Shared Structure:
  - Both GaussianCurveVisualization and PieChartVisualization inherit the method contract from VisualizationTool and provide their specific logic for creating visualizations.

2.4.1.3: Regression Analysis:

1. Abstraction:

Abstraction is applied to separate high-level tasks like regression analysis and data preprocessing from their specific implementations.

- GravWeatherAnalysis:
  - Abstracts the logic for multiple linear regression using the Apache Commons Math library to estimate regression coefficients.
  - Provides a single method, analyzeGravWeather, to handle the relationship between accident severity (grav) and weather data without exposing implementation details like matrix preparation or regression algorithm.
- DataPreprocessor:
  - Abstracts data extraction logic through the preprocessData method, which processes selected columns from a CSV file
  - Shields the caller from details like file parsing, handling missing values, or selecting columns.

2. Encapsulation:

- DataPreprocessor:
  - Encapsulates logic for reading and preprocessing data from files in a single method, preprocessData, which accepts parameters for file path, selected columns, and row limits.
- GravWeatherAnalysis:
  - Encapsulates the regression logic, including data transformation and coefficient estimation, within the analyzeGravWeather method.
  - Protects raw regression calculations (e.g., matrix setup) from external access, simplifying usability.

3. Polymorphism

- Dynamic Behavior:
  - The MainAnalysis class dynamically combines weather and accident severity data by invoking the methods preprocessData (from DataPreprocessor) and analyzeGravWeather (from GravWeatherAnalysis).

OOP Concept	Where Applied
Abstraction	GravWeatherAnalysis abstracts regression logic; DataPreprocessor abstracts data handling.
Encapsulation	Regression and preprocessing logic are encapsulated in their respective classes.
Polymorphism	Methods like preprocessData dynamically handle different datasets.

2.5 Spring Boot User Access System with CRUD Operations

- **API for CRUD Operations :**

The application uses Spring Boot to expose endpoints for managing users. Key operations include creating, reading and deleting users.

- Base User (Person):
  - GET /api/people: Retrieves all users.
  - POST /api/addPerson: Adds a new generic user.
  - DELETE /api/deletePerson/{id}: Deletes a user by ID.

- Role-Specific Endpoints:
  - Administrators:
    - GET /api/admins: Retrieves all administrators.
    - POST /api/addAdmin: Adds a new administrator.
  - Data Scientists:
    - GET /api/datascientists: Retrieves all data scientists.
    - POST /api/addDataScientist: Adds a new data scientist.
  - Users
    - GET /api/users: Retrieves all users with the role of "User".
    - POST /api/addUser: Adds a new user.

- **Web-Based Dashboard:**

- A web-based dashboard provides an intuitive interface for managing users. The HTML template (index.html) includes: User Actions such as Viewing all users, Add new users with specific roles (Administrator, Data Scientist, or User), deleting users from the table.
- The dashboard interacts with the API endpoints using JavaScript fetch requests.

- **H2 Database Integration :**

The system uses an H2 in-memory database for data persistence, accessible through a console at /h2-console. This allows administrators to perform SQL operations for direct database management.

SELECT * FROM PERSON;

ID	DTYPE	DEPARTMENT	EMAIL	NAME	SPECIALIZATION
1	DataScientist	*null*	mohamed@gmail.com	mohammed	Machine Learning
2	Administrator	IT	yassinechebbii@gmail.com	yassineChebbi	*null*
3	Administrator	IT	ranimgsouri@gmail.com	ranim	*null*
4	Administrator	IT	ranimetest@gmail.com	ranime	*null*
5	User	*null*	imeno@example.com	imeno	*null*

(5 rows, 2 ms)

Figure 2.5.1 : Person Table

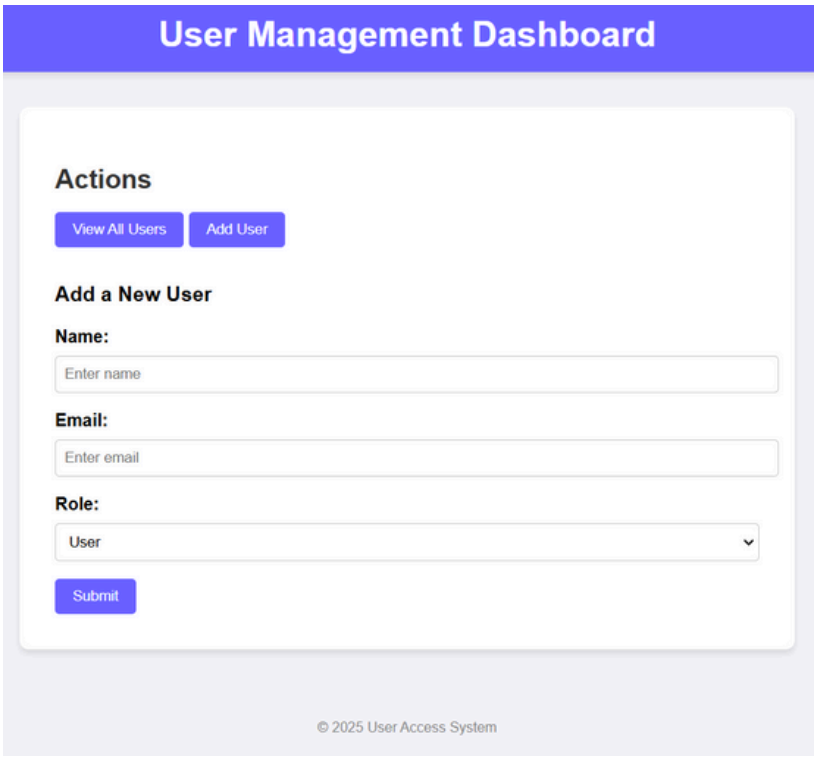


Figure 2.5.2 : User Management Dashboard

Chapter 3

Conclusion : Final Reflections

This project demonstrates the effective integration of object-oriented programming principles, data analysis techniques, and modern frameworks to create a robust and versatile system. From the foundational User Access System, which leverages encapsulation, inheritance, and polymorphism for role management, to advanced data analysis encompassing statistical metrics, correlation studies, and regression models, each component contributes to a cohesive whole. The transition to a Spring Boot application with RESTful APIs and a web-based dashboard further highlights the system's scalability and real-world applicability, enabling intuitive user management and seamless database integration. By incorporating visualization tools like Gaussian curves and pie charts, alongside predictive models such as ARIMA, the system not only interprets historical data but also forecasts future trends with actionable insights. Ultimately, this project takes into account the value of modularity, maintainability, and adaptability, providing a foundation for future enhancements and serving as a comprehensive solution for data-driven decision-making.