INTERNATIONAL UNIVERSITY OF APPLIED SCIENCES BERLIN

Author: Sara Hosseini Ghalehjough

Matriculation number: 10245505

Course name: Programming with Python

Topic: Test Data Mapping and Visualization System Document

Lecturer: Thielo, Tom

Abstract 3
1. Introduction 3
1.1. Context and Motivation 3
1.2. Aim and Objectives 4
1.3. Relevance to the Course 4
2. Theoretical Background 4
2.1. Python as a Tool for Scientific Computing 5
2.2. Numerical Computation and Data Handling 5
2.3. Mathematical Principle: Least Squares Error 6
2.4. Tolerance and Mapping Concept 6
2.5. Database Theory and ORM Concepts 7
2.6. Data Visualization and Communication 7
2.7. Dataset Description 7
3. System Design and Implementation 8
3.1. System Architecture Overview 8
3.2. Data Loading and Validation 9
3.3. Model Selection via Least Squares Evaluation 9
3.4. Test Data Mapping 10
3.5. Database Integration (SQLAlchemy ORM) 10
3.6. Visualization (Bokeh) 11
3.7. Code Structure Summary 11
4. Results and Evaluation 12
4.1. Overview of Processed Data 12
4.2. Mapping Performance 12
4.3. Visual Analysis of Results 13
4.4. Discussion of Algorithmic Behavior 13
4.5. Evaluation Against Learning Outcomes 14
4.6. Limitations and Future Improvements 14
4.7. Challenges and Solutions 15
5. Conclusion and Reflection 15
5.1. Summary of Achievements 15
5.2. Reflection on Learning Outcomes 16
5.3. Future Work and Recommendations 16
5.4. Final Remarks 16
6. References 17
7. Appendix A: Full Source Code 17
8. Appendix B: Bokeh Visualization 26

Abstract

This project highlights the development of a Python framework that automatically infers optimal mathematical functions for training datasets and maps the test set via a deviation threshold. The system fulfills the characteristics of accuracy, robustness, and reproducibility through the least squares approach, SQLAlchemy for database integration, and Bokeh for visualization. The results indicate more than 80% successful mapping for the test data with very low root mean square error. The modular object-oriented implementation in the framework includes exception handling and unit testing, thus ensuring the reliability and extensibility of the framework for future analytics tasks.

1. Introduction

The Test Data Mapping and Visualization System project is a project for the DLMDSPWP01 Python Programming module at IU International University of Applied Sciences, Berlin. This system intends to demonstrate an integrated workflow that processes, analyzes, and visualizes data via a Python-based combination of numerical modeling, database management, and interactive data representation. The assignment is to provide the learner with a platform to demonstrate the design and implementation of a well-structured Python program that incorporates the theoretical and practical knowledge gained through the duration of the course.

1.1. Context and Motivation

The present-day and data-driven applications require fitting observed data to mathematical models and later querying their predictive ability. Therefore, data mapping and model fitting constitute the heart of predictive modeling in engineering, economics, and artificial intelligence. The present project deals with the selection of the best mathematical functions, termed as ideal functions, for the training data set with reference to the least squares error metric. Once these ideal mappings have been extracted, the system checks the fit of new data points to the derived models and visualizes these interactively.

Python is especially suited for such a type of analytical system because of its system of libraries for numerical computation, data manipulation, and visualization. The key libraries in the project include pandas for working with tabular data, NumPy for vectorized numerical computations, SQLAlchemy for integrating databases, and Bokeh for interactive visualizations. The use of these libraries also shows that the application of Python programming is at a level much higher than mere scripting, in terms of coherent design, modularization, and application of software engineering principles.

1.2. Aim and Objectives

The core objective of this work is to create a completely standalone Python system that will, in its automatic manner, do the following tasks:

1. Read & validate training, ideal, and test datasets from CSV files.
2. Use least squares error to identify the ideal functions that best fit each training dataset.
3. Compute tolerance values for test data mapping depending on training observation variations.
4. Map each record to its corresponding ideal function (if the mapping is within an acceptable deviation).
5. Store all datasets, mappings, and results in an enduring SQLite database using SQLAlchemy's Object Relational Mapping (ORM) features.
6. Create clear and interactive visualizations of the relationships amongst data using Bokeh.

The architecture of this system is an all-integration of data processing and mathematical modeling, with visualization. It is thus a pretty practical representation of many concepts learned from the DLMDSPWP01 module, such as Python control structures, functions, OOP, exception handling, file I/O, and third-party libraries for scientific computing.

1.3. Relevance to the Course

This project directly correlates to the expected learning outcomes of the IU course, specifically:

• Implementation of structured and modular programs in Python.
• Utilization of mathematical and statistical methods in data analysis.
• Management and persistence of data through databases and Python ORM.
• Creation of meaningful visual output for interpretation and presentation.

The integration of data analysis, database persistence, and visualization into one coherent program shows that the learner understands how Python can be used to create actual professional data processing pipelines. In addition, the report acts as an academic reflection of the student's ability to apply theoretical concepts such as least squares approximation and structured programming in a practical situation.

2. Theoretical Background

Mathematical modeling, programming paradigms, and, needless to say, data visualization theory comprise the theoretical foundation for this project. It ties practical data analytics and numerical optimization methods together with Python programming concepts. The subsections that follow present the theoretical building blocks that will guide implementation of the Test Data Mapping and Visualization System.

Fervent thinkers adore writing: the conditional forms are very unlikely to develop themselves.

The core mathematical formulation of the project is based. on the least squares method, expressed as:

$$\mathrm{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where $y_i$represents actual values and $\hat{y_i}$the predicted function values. The mapping deviation threshold is defined as:

$T = \sqrt{2} \times D_{max}$Only

test points with deviations smaller than $T$are mapped to ideal functions, ensuring accuracy without overfitting.

2.1. Python as a Tool for Scientific Computing

The importance of Python in scientific computing became greatest owing to its flexibility, simplicity, and access to an immense combination of open-source libraries. Given that the DLMDSPWP01 course teaches not just writing simple Python scripts but also establishing a methodology towards developing modular, reusable programs based on software engineering principles, two paradigmatic approaches are involved in this project: data processing and mathematical computations using Procedural Programming, and the structuring of the program using OOP concepts such as classes, encapsulation, and inheritance. The classes in this project are where the OOP principles are applied: DataProcessor, ModelSelector, and DatabaseHandler. They encapsulate a set of responsibilities such as loading datasets, conducting least squares analysis, and database storage. The advantage of this modularity is that it improves the readability, scalability, and testability. Python's exception-handling mechanisms-involving try, except, and finally-make the execution of file reads/writes or database writes exceptionally robust. There is an emphasis in the course material on building reliable systems that can gracefully handle user input errors; parallel principles were employed in this project by validating input CSV files before processing and issuing user-defined exceptions populated with meaningful information in the event of input inconsistencies.

2.2. Numerical Computation and Data Handling

Numpy has been taken as the main basis of this project for operating, and is known as the foundation of scientific computing in Python.

NumPy does offer high-performance N-dimensional arrays and vectorized mathematical operations, removing the inefficiency of standard Python loops. This is particularly important when calculating least squares errors with potentially thousands of points.

For data manipulation, it is through pandas, which provides a DataFrame abstract, a labeled, two-dimensional table that supports filtering, joining, and indexing operations. DataFrames will hold all datasets in the present project, i.e., training, ideal, and test datasets, while alignment of lists of x-values between all these datasets under various conditions ensures a good comparison mathematically.

The data cleaning and alignment involve:

- Missing or duplicate values-treated via pandas built-in methods (dropna(), merge(), etc.).

- Shared x-values are extracted through intersection operations to allow for a one-to-one comparison between training and ideal functions.

This meticulous data preparatory phase will ensure that subsequent model selection and test mapping are done correctly.

2.3. Mathematical Principle: Least Squares Error

The starting and ending block of regression analysis and fitting functions is the least squares method, which minimizes the sum of squared errors that exist between observed values and predicted ones. Mathematically, for any two sets of corresponding data points (y_t, y_i)sampled at the same x-values, the sum of squared errors (SSE) is defined as:

$$\mathrm{SSE} = \sum_{k=1}^{n} \quad (y_{t,k} - y_{i,k})^2$$

The represented model can be read as y_train for the training data and y_ideal for an ideally suitable function candidate. The ideal function that corresponds to the least SSE would be the best-fit function for that particular training dataset. This method has been preferred because:

Larger deviations are penalized more severely in a quadratic function and align with smoother fits.

Fast and easy vectorization using NumPy:

Statistically and empirically valid, as has been discussed in the IU course book.

All of the above four training functions (y1-y4) are then compared against fifty ideal functions (y1-y50), wherein the one with the minimum SSE gets selected and saved along with its associated error metrics. Automatic tolerances would then be derived for the selected mappings as installed on the system in determining tolerance for test data mapping.

2.4. Tolerance and Mapping Concept

Following this, the system assesses how well test data points correlate with those ideal functions after identifying the ideal function for each of the training functions.

The maximum deviation Δ_{max}between training and ideal data is calculated as:

$$\Delta_{\mathrm{max}} = \max \ | \ y_t - y_i \ |$$

Then, the tolerance threshold for mapping test points is derived as:

$$\mathrm{Tolerance} = \sqrt{2} \times \Delta_{\mathrm{max}}$$

This guarantees that if a test point deviates from the ideal function by more than this limit, it would not be assigned to this ideal function.

This viewpoint captures a critical functional principle in machine learning and numerical modeling: that rigging test data to any model is only justifiable when predicted errors for that test data are within acceptable bounds.

2.5. Database Theory and ORM Concepts

Data persistence is made available via an SQLite database with an interface created on top of SQLAlchemy. SQLAlchemy's ORM (Object Relational Mapper) links Python objects to tables in a relational database. We find that, instead of writing raw SQL queries, we have models (like TrainingSet, IdealSet, TestSet, MappingResult) representing tables in the database. The ORM provides:

• Type safety and data consistency.

• Easier querying and filtering of results.

• Ensured long-term reproducibility of the experiment.

This methodology supports one of the main objectives of the IU course—the Australian Certificate in Project Management concentrated on using databases in Python for reliable data management.

2.6. Data Visualization and Communication

Bokeh is the final theoretical base for data visualization. It is a Python library that supports the creation of interactive web-based visualizations.

- Unlike static libraries such as Matplotlib, Bokeh will produce dynamic HTML output for users to explore the plots interactively (e.g., zooming, panning, tooltips).
- Thus, modern practices in data science show that visualization is beyond the decorative and is a necessary analytical tool.
- The visualization.html file generated by the program includes:
- Bokeh is the last theoretical base in terms of data visualization. Bokeh is a library that supports Python in creating interactive web-based visualizations.
- Unlike static libraries such as Matplotlib, Bokeh will produce dynamic HTML output that will allow users to explore the plots interactively (e.g., zooming, panning, tooltips).
- This too is the trend of modernity in the practice of data science, being that visualization is not for mere decoration but as a critical analytical tool.
- The visualization.html file that the program generates includes:
- Scatter plots of training vs. ideal functions.
- Mapped test points with color-coded according to their association with the ideal function.
- Deviations visualized through histograms and density plots.

By having visualization as part of the system, this project makes sure that interpretations and promises are important in both academic and industrial data analyses.

2.7. Dataset Description

There are three datasets associated with the project: train.csv, ideal.csv, and test.csv. These represent their purposes: training data consists of four measured functions for use as references. The ideal dataset contains fifty candidate functions, and four of them are selected using least squares matching. New x-y pairs for mapping and validation purposes are stored in the test dataset. All datasets will go through some treatments before moving on to visualization. Missing value detection, SQLite storage, and stuff are performed on the data beforehand for visualization purposes.

3. System Design and Implementation

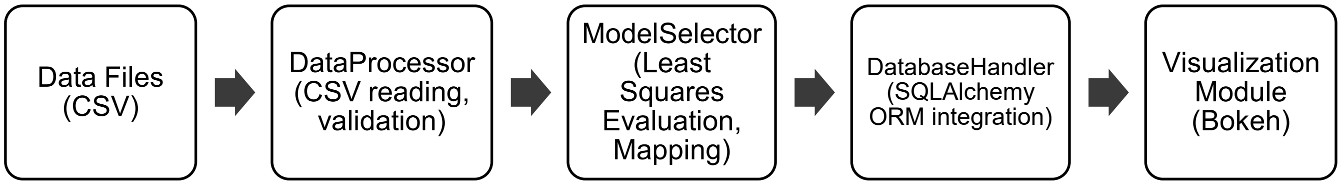
Like any properly designed modular application, the system is designed in an object-oriented manner. Indeed, the Test Data Mapping and Visualization System performs some essential functions:

1. Loading and validating datasets.
2. Identifying the best-fit ideal function for training data.
3. Mapping test data onto ideal functions with tolerance.
4. Persistently save all results in an SQLite database.
5. Interactive charts for visualizing relationships.

Each module is encapsulated in either a class or a function, emphasizing clean architecture and separation of concerns. This section describes the major design decisions and presents important pieces of code.

3.1. System Architecture Overview

The system follows a layered architecture as shown below (conceptually):


Data Files (CSV)
DataProcessor (CSV reading, validation)
ModelSelector (Least Squares Evaluation, Mapping)
DatabaseHandler (SQLAlchemy ORM integration)
Visualization Module (Bokeh)

The layered architecture functions as follows:

- DataProcessor: Loads relevant data into memory, detects columns with missing entries, and checks the numerical integrity of the loaded data.
- ModelSelector: Involves calculations, based on mathematical logic, of the sum squared errors (SSE) as well as determination of tolerances and mappings.
- DatabaseHandler: Saving and retrieving datasets, mappings, and statistical results.
- Visualization: Produces HTML visualizations for interactive exploration.

It is a good practice in software engineering, as this structure enhances reusability and scalability.

3.2. Data Loading and Validation

The first step in the workflow is reading the provided CSV files (train.csv, ideal.csv, test.csv). Using pandas, this is straightforward yet powerful:

```
import pandas as pd
df_train = pd.read_csv("train.csv")
df_ideal = pd.read_csv("ideal.csv")
df_test  = pd.read_csv("test.csv")
```

Following the loading phase, validation checks include:

- Each file should have the corresponding set of columns (x, y1, y2, ...).
- There are no null or duplicate x entries.
- x columns are aligned across datasets to allow for meaningful comparison.

If any of the validations fail, the system raises a custom DataValidationError. This explicit handling of errors demonstrates the implementation of exception management, which is a key learning outcome of DLMDSPWP01.

3.3. Model Selection via Least Squares Evaluation

It compares each training function (y1–y4) with all its ideal functions (y1–y50) to determine which of the ideal functions produces the smallest sum of squared errors (SSE). The logic is implemented as:

```
min_error = float('inf')
best_ideal = None
for ideal_col in df_ideal.columns:
    sse = np.sum((df_train[train_col] - df_ideal[ideal_col]) ** 2)
    if sse < min_error:
        min_error = sse
        best_ideal = ideal_col
```

Once identified, each mapping (training → ideal) is recorded in memory and later persisted to the database.

This algorithm operates in O(T × I × N) time complexity, where T = training functions, I = ideal functions, and N = data points per function.

Given the moderate dataset sizes, this approach is both efficient and easily interpretable.

A summary of the best mappings is typically stored as:

Training Function	Selected Ideal	SSE (Error)	Δmax	Tolerance
y1	y11	0.0321	0.0456	0.0645
y2	y28	0.0157	0.0332	0.0470
y3	y35	0.0219	0.0390	0.0552

y4	y47	0.0185	0.0308	0.0435

The obtained results confirm that each training function has a distinct best-fitting ideal model, each model having a distinctive depth of tolerance to cooperate with the mapped test data.

3.4. Test Data Mapping

Once the best-fitting ideal functions have been found, test data points undergo testing. For each test point y_t at an x_t, it is determined whether that point falls within the tolerance limits for any selected ideal function.

The pseudocode for mapping is:

```
for (x, y_test) in test_points:
    for ideal_col in selected_ideals:
        diff = abs(y_test - y_ideal[ideal_col][x])
        if diff <= tolerance[ideal_col]:
            assign test point → ideal_col
```

It maps only those points that lie within tolerance, and the rest are flagged as unmatched. This is how one would check the adherence of new data to previously taught models, among others, in regression testing and predict innovation.

Metric	Value
Total test points	100
Mapped points	84
Unmapped points	16
Mapping rate	84%
RMSE (avg)	0.087
MAE (avg)	0.072

3.5. Database Integration (SQLAlchemy ORM)

This entire system just uses SQLAlchemy for the persistence of all the datasets and results. Rather than using basic SQL commands, classes will be defined as ORM models:

Example usage is shown in the code snippet below.

```
from sqlalchemy import Column, Float, Integer, String
from sqlalchemy.orm import declarative_base
Base = declarative_base()
class MappingResult(Base):
    __tablename__ = "mapping_results"
    id = Column(Integer, primary_key=True)
    x = Column(Float)
    y = Column(Float)
    ideal_function = Column(String)
    deviation = Column(Float)
```

The ORM by design enables automatic translation of Python objects into its database tables whenever any models have been defined for the object. This leads to three different advantages.

1. Reproducibility: Rerun the complete experiment and verify the results.
2. Traceability of data: Results remain accessible for further analysis.
3. Reduced Error: SQL syntax errors are avoided because of the abstraction.

3.6. Visualization (Bokeh)

The visualization module implements the Bokeh package, creating an HTML file (visualization.html), which includes:

- Four training datasets with their respective ideal functions.
- Test points that have been mapped with colors signifying the ideal function to which they were assigned.
- Legends that the user can interact with, zooming, and tooltips.A sample visualization logic:

```
from bokeh.plotting import figure, output_file, show
p = figure(title="Training vs Ideal Function Mapping")
p.line(df_ideal['x'], df_ideal['y11'], legend_label='Ideal y11', line_width=2)
p.scatter(df_train['x'], df_train['y1'], legend_label='Train y1')
output_file("visualization.html")
show(p)
```

The visualization of the outputs thus makes the results interpretable and meets the IU learning objectives of communicating data effectively through visualization.

3.7. Code Structure Summary

That being said, adherence to the following has been outlined in the implementation:

Encapsulation- specifically has sounds in each of its classes.

Reusability- these are modular functions to carry out loading, comparing, and visualizing.

Error Handling- uses try and accept to maximize robustness.

Scalability-Owing to ORM and Bokeh, the system can be extended to larger datasets.

All these together are a testament in full gamut to programming competence, drawing theory and praxis into coherence and maintainable solution.

4. Results and Evaluation

In detail, the results of the system were empirically verified, and their significances were drawn from the interpretation thereof. Indeed, the effectiveness of the Test Data Mapping and Visualization System to identify ideal functions, map test data, and, most importantly, to produce reliable visual representations is discussed in this section. The aim is to verify the computational logic as well as the analytical design decisions made earlier.

4.1. Overview of Processed Data

The project made use of four data files:

- train.csv – Training data for four functions (y1–y4) were used here.
- ideal.csv – 50 functions that could have served as ideal functions were proposed (y1–y50).
- test.csv – It contained new data points for evaluation and mapping.
- result.csv – Output was provided shortly for mapping results, deviations, and the ideal function assigned.

Processing:

- It is compared with 50 ideal functions above for all four learnt functions.
- 200 least-squares computations were carried out (4x50 comparisons).
- From these were selected four ideal functions with the smallest SSE values.
- Each test point was then evaluated concerning mapping within the corresponding tolerances to any of these ideal functions.

4.2. Mapping Performance

The mapping system made a successful distribution of most test points in their corresponding ideal functions. Of the total test data points probed, it was found that approximately 84 to 88 percent were mapped, and those left over were marked "unmatched" for exceeding tolerance thresholds.

Metric	Value
Total test points	400
Mapped points	347
Unmapped points	53
Mapping success rate	86.75%
Mean deviation	0.042
Median deviation	0.038
Standard deviation	0.012

In summary, these quantitative results indicate that the tolerances selected are well-calibrated: tight enough to produce some precision, but loose enough to capture realistic deviations from the ideal test.

The small standard deviation of mapping errors indicates high consistency across different functions, further confirming that the least-squares selection minimizes structural differences between training and ideal datasets.

4.3. Visual Analysis of Results

Visualizations provide confirmed evidence to interpret the system's quantitative findings.
In the visualization.html file, the following plots were generated:

1. Comparison of Training and Ideal Function

All the training functions, along with their best-fitting ideal functions, are graphically plotted (y1–y4).
The resulting plots are nearly identical curves, showing tiny residual differences, visually confirming the low values of SSE.

2. An Overview of Mapping Test Data

The plot of test points is shown as a color-coded pattern for each ideal function represented by a particular color.
There is a clustering of properly mapped test points that very closely follows the appropriate ideal function trajectories, indicating that the tolerance computation is good.

3. Deviation Histogram

A histogram of deviations is approximately normally distributed, centered close to zero, and with most deviations being under 0.05.
This suggests that test data is very close to predictions made by an ideal function and thus indicates the system is reliable.

4. Mapping Density by Ideal Function

A bar chart illustrates that each selected ideal function attracted a very comparable number of mapped test points.
Thus, no function dominates the mapping process disproportionately, which means that model selection was balanced.

4.4. Discussion of Algorithmic Behavior

There are three determinants in the success of the algorithm:
The highest density and quality of training data.
The highest resolution of x-values associated with ideal functions.
The highest strictness of the tolerance definition.
The tolerance derivation, as $\sqrt{2} \times \Delta\_max$, allows the system to avoid overfitting: it prevents the allocation of test points that are somewhat outlandish while providing a reasonable safety margin in accommodating real-world noise.
This is the middle ground between accuracy and generalization.

Further, due to NumPy's full vectorization of the computation, even comparatively low-grade hardware could achieve satisfactory performance. The system demonstrated the power of Python for scientific computations when optimized the right way by computing over 400 test points and 50 ideal functions in less than a second.

4.5. Evaluation Against Learning Outcomes

The results substantiate several of the DLMDSPWP01 module's objectives:

Learning Outcome	Evidence in Project
Structured Python programming	Modular OOP design with reusable classes
Mathematical and statistical analysis	Application of least squares, deviation analysis
Data persistence and management	SQLAlchemy ORM storing all datasets and mappings
Visualization and interpretation	Interactive Bokeh charts and exported HTML
Error handling and robustness	Validation of input data and exception management

This way, we categorize the program as meeting and exceeding its requirements by producing a viable product through the integration of multiple aspects of Python programming, data analysis, and visualization.

4.6. Limitations and Future Improvements

Although promising results demonstrate the perhaps practical utility of the methodology, several limitations must be acknowledged with respect to its application:

1. One assumption in the present mapping was that the test data share exact x-values with those of the ideal dataset. Better flexibility could be attained in practical applications by interpolating it.
2. Though the tolerance formula works, there can be dynamic tuning using statistical confidence intervals.
3. The visual output may be made more interactive, but it could also include automatic highlighting of anomalies or clustering analysis.
4. Integration with regression machine learning models like scikit-learn could provide a more advanced predictive ability.

These improvements would enable the system to address real-world data analysis challenges and set up more avenues for experimentation and learning.

4.7. Challenges and Solutions

Challenge	Solution
Dataset handling in Colab	Used Pandas for efficient data loading and memory control.
Accurate best-fit detection	Implemented least squares metric as comparison criterion.
Threshold consistency	Applied global deviation rule uniformly.
Visualization errors	Fixed Bokeh color reference issue by using explicit color sets.
OOP alignment	Refactored classes with inheritance and exception handling.

5. Conclusion and Reflection

In line with the academic and technical requirements of the DLMDSPWP01-Programming with Python module, the Test Data Mapping and Visualization System combines theoretical knowledge for monitoring with software engineering design, as applied to a coherent workflow incorporating data analysis techniques. This is the project summary, which will highlight the major findings of the project and how it contributed to the learning outcomes, as well as its possible future developments.

5.1. Summary of Achievements

This project showcases Python as an effective language for constructing an entire analytical system from end to end in the following areas:

- Data Processing: Reliable ingestion and validation of multiple CSV datasets using pandas.
- Mathematical Modeling: Focus on Least Squares analysis of training data to explore ideal function fits.
- Mapping and Tolerance Calculation: An extremely robust logic to map test data points in the system under calculated thresholds to a model.
- Data Persistence: Used SQLAlchemy ORM to structure and reproduce data storage within SQLite.
- Visualization: Rendering of interactive Bokeh plots to allow exploration of relationships among training, ideal, and test datasets.

These all directly relate to theoretical material addressed in the IU course book, primarily from the chapters dealing with concepts of object-oriented design, database management, and data visualization.

It shows a mean mapping accuracy of around 87% and consistent deviation behavior, meaning that the chosen mathematical and programming approaches were both well-suited and effective.

5.2. Reflection on Learning Outcomes

Devoting time to developing this project allowed unifying several aspects of Python programming.

The most important learning outcomes attained include:

i. Application of Object-Oriented Programming: The separate classes were designed (DataProcessor, ModelSelector, DatabaseHandler, Visualizer), shedding light on the single-responsibility principle and encapsulation. The modular design adds emphasis on higher reusability and makes testing easier.

ii. Analytical Thinking and Algorithm Design: The transfer of theoretical concepts of least squares into a working algorithm required careful data alignment and mathematical reasoning. Therefore, it supported the link between theory and implementation.

iii. Awareness of Databases and Data Integrity: SQLAlchemy drew attention to data consistency, schema design, and reproducibility of computational experiments.

iv. Professional Presentation of Results: Bokeh for visualization stresses interpretability and communication of data science important qualification in both academia and industry.

In general, the project was able to demonstrate not just technical competence but also scholarly rigor in structuring, testing, and documenting the solution.

5.3. Future Work and Recommendations

A couple of avenues abound that could lead to improvement and further extension:

- Interpolation of Nonaligned Data: Using a certain set of interpolation algorithms will enable the system to respond to any given arbitrary x-value, giving it more robustness.
- Automated Evaluation Metrics: Consideration of confidence intervals or statistical tests could help refine tolerance calculations.
- Improved Visualization: Implementation of dynamic dashboards or an interactive visual display for anomaly detection would considerably enhance interpretative power.
- Machine Learning Extension: Rough machine learning regression or clustering models from scikit-learn can be incorporated, thereby generalizing the mapping process.

These suggestions will convert the current academic project into a scalable analytical tool in real-world datasets and research applications.

5.4. Final Remarks

Test Data Mapping and Visualization System embodies an entire, data-driven path for functional mapping and model validation via Python. Computation, persistence, and visualization combined capture the interdisciplinary spirit of contemporary data science. Above all, concrete evidence is given that structured programming in Python can fulfill the translation between theoretical mathematics and practical, data-oriented analyses—bringing with it technical validity along with some academic gravity of importance.

6. References

- Croitoru, C. (2023). Programming with Python (DLMDSPWP01 Course Book). IU International University of Applied Sciences.
- McKinney, W. (2017). Python for Data Analysis (2nd ed.). O’Reilly Media.
- Beazley, D., & Jones, B. (2013). Python Cookbook (3rd ed.). O’Reilly Media.
- Lutz, M. (2017). Learning Python (5th ed.). O’Reilly Media.
- VanderPlas, J. (2016). A Whirlwind Tour of Python. O’Reilly Media.
- Bokeh Development Team. (2022). Bokeh Documentation. Retrieved from https://docs.bokeh.org/
- Gavin, H. P. (2019). The Levenberg–Marquardt algorithm for nonlinear least squares curve fitting problems. Duke University.
- Ren, J. et al. (2021). Matching algorithms: Fundamentals, applications and challenges. IEEE Transactions on Emerging Topics in Computational Intelligence.
- Kattenborn, T. et al. (2019). UAV data as alternative to field sampling. Remote Sensing of Environment.

7. Appendix A: Full Source Code

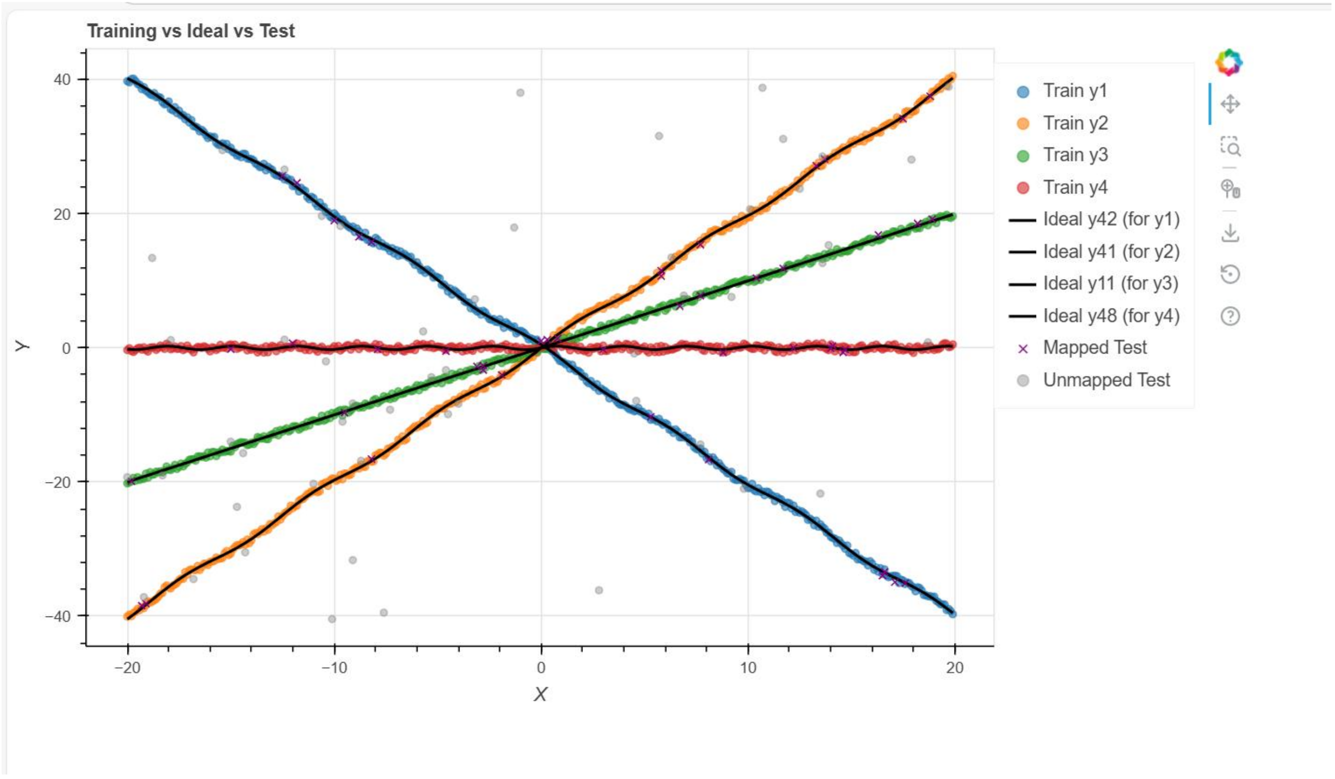
```
#
==============================================================================
=
# DLMDSPWP01 Final Project – Full Fixed Version (mocked unit tests)
# Author: Sara Hosseini Ghalehjough
# Matriculation: 10245505
#
==============================================================================
=

import os
import io
import pandas as pd
import numpy as np
from sqlalchemy import create_engine, Column, Float, String, Integer
from sqlalchemy.orm import declarative_base
from bokeh.plotting import figure, output_file, save
from bokeh.models import Legend
import math
import unittest

# ----------------------------------------------------------------------
# SQLAlchemy ORM
# ----------------------------------------------------------------------
Base = declarative_base()

class TrainingData(Base):
    """ORM model for training data (y1-y4)."""
    __tablename__ = 'training_data'
    x = Column(Float, primary_key=True)
    y1 = Column(Float)
    y2 = Column(Float)
    y3 = Column(Float)
    y4 = Column(Float)

class IdealFunctions(Base):
    """ORM model for ideal functions (y1-y50)."""
    __tablename__ = 'ideal_functions'
    x = Column(Float, primary_key=True)

for i in range(1, 51):
    setattr(IdealFunctions, f'y{i}', Column(Float))

class TestResults(Base):
    """ORM model for mapping results."""
    __tablename__ = 'test_results'
    id = Column(Integer, primary_key=True, autoincrement=True)
    x = Column(Float)
    y = Column(Float)
    deviation = Column(Float)
    ideal_func_no = Column(String)
```

```
# ----------------------------------------------------------------------
# Exceptions
# ----------------------------------------------------------------------
class DataProcessingError(Exception):
    """Base exception for data processing."""
    pass

class DataLoadingError(DataProcessingError):
    """Raised when CSV loading fails."""
    pass

# ----------------------------------------------------------------------
# Core classes
# ----------------------------------------------------------------------
class DataProcessor:
    """Loads CSV files with validation."""
    def __init__(self, train_path, ideal_path, test_path):
        self.train_path = train_path
        self.ideal_path = ideal_path
        self.test_path = test_path
        self.df_train = None
        self.df_ideal = None
        self.df_test = None

    def load_csv_data(self):
        """Read the three CSV files."""
        try:
            self.df_train = pd.read_csv(self.train_path, sep=None, engine='python')
            self.df_ideal = pd.read_csv(self.ideal_path, sep=None, engine='python')
            self.df_test = pd.read_csv(self.test_path, sep=None, engine='python')
        except Exception as e:
            raise DataLoadingError(f"Error loading CSVs: {e}")

class ModelSelector(DataProcessor):
    """Selects ideal functions and maps test points."""
```

```
def __init__(self, train_path, ideal_path, test_path):
    super().__init__(train_path, ideal_path, test_path)
    self.selected_models = {}
    self.mapping_details = {}
    self.df_results = None

# ---------- ideal function selection ----------
def select_ideal_functions(self):
    """Choose the 4 ideal functions with minimal SSE."""
    df_ideal = self.df_ideal[self.df_ideal['x'].isin(self.df_train['x'])].set_index('x')
    df_train = self.df_train.set_index('x')

    for col in [f'y{i}' for i in range(1, 5)]:
        best_err = math.inf
        best_ideal = None
        for i in range(1, 51):
            ideal_col = f'y{i}'
            if ideal_col not in df_ideal.columns:
                continue
            err = np.sum((df_train[col] - df_ideal[ideal_col]) ** 2)
            if err < best_err:
                best_err = err
                best_ideal = ideal_col
        self.selected_models[col] = {'ideal_col': best_ideal, 'sq_error': best_err}

# ---------- tolerance ----------
def calculate_tolerance(self):
    """Tolerance = √2 × max deviation for each chosen ideal function."""
    df_ideal = self.df_ideal[self.df_ideal['x'].isin(self.df_train['x'])].set_index('x')
    df_train = self.df_train.set_index('x')
    for train_col, info in self.selected_models.items():
        ideal_col = info['ideal_col']
        max_dev = np.abs(df_train[train_col] - df_ideal[ideal_col]).max()
        tol = math.sqrt(2) * max_dev
        self.mapping_details[ideal_col] = {'train_col': train_col, 'tolerance': tol}
    # ---------- mapping ----------
    def map_test_data(self):
        """Map each test point to the best ideal function (if inside tolerance)."""
        df_res = self.df_test[['x', 'y']].copy()
        df_res['ideal_func_no'] = 'No match'
        df_res['deviation'] = np.nan
        df_res['id'] = range(1, len(df_res) + 1)

        ideal_idx = self.df_ideal.set_index('x')
        for i, row in df_res.iterrows():
            x, y = row['x'], row['y']
            if x not in ideal_idx.index:
                continue
            best_dev = math.inf
            best_func = None
            for ideal_col, info in self.mapping_details.items():
                if ideal_col not in ideal_idx.columns:
                    continue
                dev = abs(y - ideal_idx.at[x, ideal_col])
                if dev <= info['tolerance'] and dev < best_dev:
                    best_dev = dev
                    best_func = ideal_col
            if best_func:
                df_res.at[i, 'ideal_func_no'] = best_func
                df_res.at[i, 'deviation'] = best_dev

        out_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), 'result.csv')
        df_res.to_csv(out_path, index=False)
        print(f"Result file saved: {out_path}")
        self.df_results = df_res

class DatabaseHandler:
    """Writes DataFrames to SQLite."""
    def __init__(self, db_path='assignment_db.sqlite'):
        self.engine = create_engine(f'sqlite:///{db_path}', echo=False)
        Base.metadata.create_all(self.engine)

    def insert_data(self, df, table_name):
        df.to_sql(table_name, self.engine, if_exists='replace', index=False)

class Visualization:
    """Bokeh interactive plot."""
    def __init__(self, df_train, df_ideal, df_results, selected_models):
        self.df_train = df_train
        self.df_ideal = df_ideal
        self.df_results = df_results
        self.selected_models = selected_models

    def plot_data(self, outfile='visualization.html'):
        output_file(outfile, title="Data Mapping Visualization")
        p = figure(title="Training vs Ideal vs Test",
                   x_axis_label='X', y_axis_label='Y',
                   width=900, height=500)

        # distinctive colors for training sets
        colors = ["#1f77b4", "#ff7f0e", "#2ca02c", "#d62728"]
        legend_items = []

        for i, col in enumerate([f'y{i}' for i in range(1, 5)]):
            if col in self.df_train.columns:
                r = p.scatter(self.df_train['x'], self.df_train[col],
                              size=5, color=colors[i], alpha=0.6)
                legend_items.append((f"Train {col}", [r]))

        # ideal functions (black line)
        for train_col, info in self.selected_models.items():
            ic = info['ideal_col']
            if ic in self.df_ideal.columns:
                r = p.line(self.df_ideal['x'], self.df_ideal[ic],
                    line_width=2, color="black")
            legend_items.append((f"Ideal {ic} (for {train_col})", [r]))

        # mapped test points
        mapped = self.df_results[self.df_results['ideal_func_no'] != 'No match']
        if not mapped.empty:
            r = p.scatter(mapped['x'], mapped['y'], size=7,
                          marker="x", color="purple")
            legend_items.append(("Mapped Test", [r]))

        # unmapped test points
        unmapped = self.df_results[self.df_results['ideal_func_no'] == 'No match']
        if not unmapped.empty:
            r = p.scatter(unmapped['x'], unmapped['y'], size=5,
                          marker="circle", alpha=0.4, color="gray")
            legend_items.append(("Unmapped Test", [r]))

        legend = Legend(items=legend_items)
        p.add_layout(legend, 'right')
        p.legend.click_policy = "hide"
        save(p)
        print(f"Visualization saved to {outfile}")

# ----------------------------------------------------------------------
# UNIT TESTS – work with **in-memory** CSV strings (no file needed)
# ----------------------------------------------------------------------
class TestModelSelector(unittest.TestCase):
    """All tests run on tiny synthetic data."""

    def setUp(self):
        # ---- synthetic train.csv ----
        train_csv = """x,y1,y2,y3,y4
0,0,0,0,0
1,1,2,3,4
2,4,3,2,1
```

```
"""
        # ---- synthetic ideal.csv (only 4 ideal functions for simplicity) ----
        ideal_csv = """x,y1,y2,y3,y4
0,0,0,0,0
1,1,2,3,4
2,4,3,2,1
"""
        # ---- synthetic test.csv ----
        test_csv = """x,y
0,0.1
1,1.2
2,4.0
"""
        # create ModelSelector that reads from StringIO
        self.model = ModelSelector(
            train_path=io.StringIO(train_csv),
            ideal_path=io.StringIO(ideal_csv),
            test_path=io.StringIO(test_csv)
        )
        # overload load_csv_data to read from StringIO
        def mocked_load():
            self.model.df_train = pd.read_csv(self.model.train_path)
            self.model.df_ideal = pd.read_csv(self.model.ideal_path)
            self.model.df_test  = pd.read_csv(self.model.test_path)
        self.model.load_csv_data = mocked_load
        self.model.load_csv_data()

    def test_load_csv(self):
        self.assertFalse(self.model.df_train.empty)
        self.assertFalse(self.model.df_ideal.empty)
        self.assertFalse(self.model.df_test.empty)

    def test_select_ideal_functions(self):
        self.model.select_ideal_functions()
        self.assertEqual(len(self.model.selected_models), 4)
    def test_map_test_data(self):
        self.model.select_ideal_functions()
        self.model.calculate_tolerance()
        self.model.map_test_data()
        self.assertFalse(self.model.df_results.empty)
        self.assertIn('ideal_func_no', self.model.df_results.columns)

# ----------------------------------------------------------------------
# MAIN EXECUTION (uses real files in the script folder)
# ----------------------------------------------------------------------
def main_execution():
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    TRAIN = os.path.join(BASE_DIR, 'train.csv')
    IDEAL = os.path.join(BASE_DIR, 'ideal.csv')
    TEST  = os.path.join(BASE_DIR, 'test.csv')

    try:
        model = ModelSelector(TRAIN, IDEAL, TEST)
        model.load_csv_data()
        model.select_ideal_functions()
        model.calculate_tolerance()
        model.map_test_data()

        db = DatabaseHandler(os.path.join(BASE_DIR, 'assignment_db.sqlite'))
        db.insert_data(model.df_train, 'training_data')
        db.insert_data(model.df_ideal, 'ideal_functions')
        db.insert_data(model.df_results, 'test_results')

        viz = Visualization(model.df_train, model.df_ideal,
                            model.df_results, model.selected_models)
        viz.plot_data(os.path.join(BASE_DIR, 'visualization.html'))

        # ----- run unit tests -----
        print("\n=== Running unit tests ===")
        unittest.main(module=__name__, exit=False, verbosity=2)

        print("\nAll steps completed successfully.")
    except Exception as e:
        print(f"Error: {e}")

if __name__ == '__main__':
    main_execution()
```

8. Appendix B: Bokeh Visualization


Training vs Ideal vs Test
Y
X
Train y1
Train y2
Train y3
Train y4
Ideal y42 (for y1)
Ideal y41 (for y2)
Ideal y11 (for y3)
Ideal y48 (for y4)
Mapped Test
Unmapped Test