PYTHON WRITTEN ASSIGNMENT

PROGRAMMING WITH PYTHON DLMDSPWP01

14th March 25

Ingrid Achieng

Mat Number

Tutor Name

Table of Contents

[1.Introduction 4](#_Toc195749520)

[1.1 Objectives 4](#_Toc195749521)

[1.2 Methodology 4](#_Toc195749522)

[2. Implementation 4](#_Toc195749523)

[2.1 Exploratory Data Analysis 4](#_Toc195749524)

[2.2 Imports 5](#_Toc195749525)

[2.3 Data Loading 6](#_Toc195749526)

[2.3.1 Database Manager Class (**DatabaseManager**) 6](#_Toc195749527)

[2.3.2 Dataset Manager Class (DatasetManager) 7](#_Toc195749528)

[2.3.3 Main Program(main.py) 7](#_Toc195749529)

[2.4 Function Selection 8](#_Toc195749530)

[2.4.1 Define the Error Metric: 9](#_Toc195749531)

[2.4.2 Calculate SSE for Each Training Function 9](#_Toc195749532)

[2.4.3 Select the Best Ideal Function for Each Training Function 9](#_Toc195749533)

[2.4.4 Repeat for All Training Functions 9](#_Toc195749534)

[2.4.5 Verify Uniqueness 9](#_Toc195749535)

[2.4.6 Final Selection 9](#_Toc195749536)

[2.5 Data Mapping 9](#_Toc195749537)

[2.5.1 Logic Flow 10](#_Toc195749538)

[2.5.2 Imports 11](#_Toc195749539)

[2.5.3 Data Handling 11](#_Toc195749540)

[2.5.4 Function Selection (FunctionSelection) 11](#_Toc195749541)

[2.6 Visualization 11](#_Toc195749542)

[2.7 Test 12](#_Toc195749543)

[2.7.1 Unit Testing for Python Program 12](#_Toc195749544)

[2.7.2 Overview of the Testing Approach 12](#_Toc195749545)

[2.7.3 Structure of the Unit Tests 13](#_Toc195749546)

[3. Results 14](#_Toc195749547)

[4. Discussion 15](#_Toc195749548)

[5. Conclusion 15](#_Toc195749549)

[References 16](#_Toc195749550)

[Images 17](#_Toc195749551)

**List of figures and/or tables (if necessary)**

**Abstract**

This paper presents a Python project that involves data mapping and visualization using SQLite, Pandas, and Bokeh. The project is based on a task that requires the selection of ideal functions from a set of training data, mapping test data to these functions, and visualizing the results. The paper outlines the project's objectives, methodology, implementation, and results.

# 1.Introduction

The project involves processing and analyzing datasets to identify the best-fit ideal functions from a set of training data. The primary goal is to map test data to these ideal functions and visualize the results. The project is implemented using Python, with libraries such as SQLite for database management, Pandas for data manipulation, and Bokeh for data visualization.

## 1.1 Objectives

* Load training, ideal, and test datasets into a SQLite database.
* Select the four best-fit ideal functions based on the least squares method.
* Map test data to the selected ideal functions using a defined criterion.
* Visualize the training data, ideal functions, and mapped test data.
* Ensure the code is object-oriented, well-documented, and includes unit tests.

## 1.2 Methodology

The project follows a structured approach:

1. **Data Loading**: Load datasets into a SQLite database using Pandas and SQLAlchemy.
2. **Function Selection**: Use the least squares method to select the best-fit ideal functions.
3. **Data Mapping**: Map test data to the selected ideal functions based on deviation criteria.
4. **Visualization**: Use Bokeh to create interactive visualizations of the data.
5. **Testing and Documentation**: Implement unit tests and document the code thoroughly.

# 2. Implementation

## 2.1 Exploratory Data Analysis

This section(EDA.ipynb) documents the exploratory data analysis performed on the dataset prior to developing the main program. The EDA helps understand the structure, characteristics, and quality of the data.The insights into the data structure and patterns confirms that:

* The datasets have expected dimensions and contain no missing values.
* Ideal functions exhibit distinct patterns when visualized. Figure 1
* Test data follows a structured distribution.
* Training data contains multiple functions with varying trends. This understanding is critical for selecting the best-fit functions and mapping test data effectively.

A graph of a function

Description automatically generated

*Figure 1: EDA of ideal functions data*

## 2.2 Imports

The script begins by importing necessary libraries:

import sqlite3

import unittest

import pandas as pd

import numpy as np

from pathlib import Path

from sqlalchemy import create\_engine, text

from sqlalchemy.exc import OperationalError, SQLAlchemyError

**SQLite:**

SQLite is utilized in this code for database management and data handling. It serves as the storage system for the program's data, enabling efficient querying, manipulation, and saving of results. Here's how SQLite is specifically used for database initialization and loading data, data retrieval and manipulation, saving mapped test data

**pandas**:

The use of pandas, a powerful data manipulation library, is also prominent in the code, specifically for creating and managing DataFrame objects that are instrumental in data analysis tasks. Recent literature highlights the efficacy of pandas in transforming raw data into a structured format suitable for further analysis (Bantilan, 2020). In the context of this code, pandas is employed to load the train, ideal, and test datasets into an in-memory SQLite database, which also facilitates efficient testing without the need for persistent storage.

Numpy:

Numpy is used in the code for numerical operations, specifically in th FunctionSelection and TestDataMapper classes.Here’s how:

**FunctionSelection.find\_best\_fit**:

sse = np.sum((y\_train - y\_ideal) \*\* 2): Calculates the sum of squared errors between the training data and ideal function values using element-wise subtraction, squaring, and summation.

**TestDataMapper**:

sqrt\_2 = np.sqrt(2): Calculates the square root of 2 for use as a factor in the deviation threshold.

deviations = np.abs(self.train\_data[train\_col].values - self.ideal\_data[best\_func].values): Computes the absolute differences between training data and ideal function values.

self.max\_deviation[best\_func] = np.max(deviations): Determines the maximum deviation between the training data and the selected ideal function using Numpy's maxfunction.

**sqlalchemy**:

The integration of the SQLAlchemy library in this code is significant, as it provides a comprehensive toolkit for SQL database interaction in Python applications. It facilitates object-relational mapping (ORM) that abstracts database interactions to a higher level, promoting cleaner and more maintainable code Chen (2025). This encapsulation enables developers to work with Python objects instead of raw SQL queries, thereby enhancing productivity and code clarity (Holehouse & Naegle, 2015).

* **create\_engine**: The engine manages the SQLAlchemy connection pool and the database-independent SQL dialect layer.In this script an engine is created manually, using the SQLAlchemy function create\_engine( ) and a connection is made to the SQLite database. (Myers et al., 2015)
* **text**: SQLAlchemy-provided classes known as TypeEngines. TypeEngine objects convert Python values to native database values and vice versa.

Used to execute raw SQL queries.The test() function is a generic TypeEngines provided by SQLAlchemy and its used to create textual SQL statements that are passed to the database with minimal modification. (Myers et al., 2015)

* **OperationalError, SQLAlchemyError**: Used for error handling related to database operations like Database Operation Issues and Exception Handling.

**Pathlib**:

The pathlib library revolves around what are called Path objects, which represent file system paths in a structured and   platform-independent way.  After calling the Path class from pathlib, we can create Path objects in several ways, including from strings, from other Path objects. Existing Path objects can serve as building blocks for creating new paths. We do this by combining a base path, a data directory, and a file name into a single file path. We have to remember to use a forward slash where appropriate to extend our Pathobjects *(DataCamp, n.d.)*

BASE\_DIR = Path('Task\_1')

DATA\_DIR = BASE\_DIR / 'dataset\_1'

**Unittest|**

The provided Python program constitutes a unit testing framework for verifying the functionality of data handling and processing components in the workflow, particularly using the unittest framework. This approach is common in software development to ensure that changes and updates to the code do not introduce unintended errors, a practice noted in the literature as essential for maintaining software quality and reliability (Melchert et al., 2018), (Holehouse & Naegle, 2015).

## 2.3 Data Loading

The first step involves loading the training, ideal, and test datasets in the form of CSV files into a SQLite database using the pandas and sqlalchemy libraries. The primary functionalities include creating a database, adding data to the database from CSV files, and fetching data from the database. The CSV\_DB.py script handles this task.

### 2.3.1 Database Manager Class (**DatabaseManager**)

The DatabaseManager class serves as the data persistence layer for the program. It handles the creation of the SQLite database and connection using SQLAlchemy. Moreover, using SQLite as a lightweight database solution reflects a trend in the field where developers opt for in-memory databases in testing and prototyping phases due to their simplicity and efficiency (Suraya & Sholeh, 2021). As noted in the literature, SQLite is favored for local data storage needs in applications where minimal overhead and ease of setup are paramount (Dobric et al., 2022).

SQLAlchemy is used to load data from CSV files into the database, and fetching data from the database.

* **\_\_init\_\_**
  + The \_\_init\_\_ method initializes the database connection using SQLAlchemy.It takes a db\_name parameter, which is used to create or connect to an SQLite database with the name db\_name.db.
* **csv\_to\_table**
  + This method loads data from a CSV file into a specified table in the SQLite database.Pandas is used for CSV reading df = pd.read\_csv(csv\_file).
  + It handles two scenarios:
    - **Test Data**: If the CSV file name contains the word "test", the data is loaded line-by-line using chunksize=1. This is useful for large datasets or when processing data incrementally.
    - **Training and Ideal Data**: For other CSV files, the data is loaded all at once using pd.read\_csv() and then written to the database using df.to\_sql().
  + The method also handles exceptions such as  FileNotFoundError, OperationalError, and other unexpected errors.
* **fetch\_data**
  + This method fetches data from a specified table in the database. Pandas is used to transfer data retrieved from an SQL query result into a pandas **DataFrame**.
    - data = pd.DataFrame(result.fetchall(), columns=result.keys())
  + The method takes two parameters table\_name and limit, the name of the table from which to fetch data and the number of rows to fetch (default is 5). Returns the fetched data as a pandas DataFrame, If the table is empty or an error occurs it returns None.

### 2.3.2 Dataset Manager Class (DatasetManager)

The DataseManager class inherits from **DatabaseManager** and is used to handle specific datasets as demonstrated by the pandas *DataFrame.to\_sql* function (pandas documentation). It simplifies the process of loading data from a CSV file into a database table.

* **\_\_init\_\_**
  + The \_\_init\_\_ method initializes the DatasetManager object with the database name, CSV file path, and table name. It calls the parent class (DatabaseManager) constructor to set up the database connection.
* **load\_data**
  + This method calls the csv\_to\_table method from the parent class to load data from the specified CSV file into the database table.

### 2.3.3 Main Program(main.py)

The main script handles file paths and error cases, three datasets are defined, each with a CSV file path and a corresponding table name:

* train.csv → Training\_data
* ideal.csv → Ideal\_functions\_data
* test.csv → Test\_data

For each dataset, a DatasetManager object is created, and the load\_data method is called to load the data into the database. After loading, the fetch\_data method is called to print a sample of the data from each table as a sign the data was loaded sucessfully.

## 2.4 Function Selection

The FuncSelection class in mapping.py script is responsible for selecting the best-fit ideal functions.

In linear least squares, we try to minimize the sum of squared distances from each training point (denoted by a red circle) to its approximation (denoted by a blue cross) as shown in Figure 2 that is, we minimize the sum of the lengths of the little vertical blue lines. The red diagonal line represents y (x) = w 0 + w 1 x , which is the least squares regression line. *Murphy (2012)*. We use this approach to calculates the sum of squared deviations between the training data and each ideal function which will take the place of approximation.To calculate the sum of all y-deviations squared (Least-Square) this formula is used minimizes the **sum of squared deviations** (errors) between the training values and the ideal function values.

A math formula with square and triangle symbols

Description automatically generated with medium confidence

*Figure 1: Formula for Calculating the Sum of Squared Errors (SSE)*

Where:

* *ytraining* is the value from the training data.
* *yideal* is the value from the ideal function.
* *n* is the number of data points.

A graph with red and blue lines

Description automatically generated

*Figure 2: Linear Least Square*

### 2.4.1 Define the Error Metric:

The differences between the actual and predicted values are known as residuals or errors. A good fit between the line and the data results in smaller residuals on average. To determine the best-fitting line, the goal is to minimize these overall errors. However, because some residuals are positive (points above the line) and others are negative (points below the line), their sum tends to cancel out to zero. To accurately measure the total error, we square each residual—making them all positive—and then identify the line that produces the smallest possible sum of these squared differences. (JMP, n.d.)

For each training function (*y*1, *y*2, *y*3, *y*4), calculate the **Sum of Squared Errors (SSE)** for each ideal function (*y*1 to *y*50)

### 2.4.2 Calculate SSE for Each Training Function

For each training function (*y*1, *y*2, *y*3, *y*4):

* Iterate through all ideal functions (*y*1 to *y*50).
* For each ideal function, compute the SSE between the training function and the ideal function.
* Record the SSE for each pair.

### 2.4.3 Select the Best Ideal Function for Each Training Function

For each training function:

* Identify the ideal function with the minimum SSE.
* This ideal function is the best fit for that training function.

### 2.4.4 Repeat for All Training Functions

Repeat Steps; Calculate SSE for Each Training Function and Select the Best Ideal Function for Each Training Function for all 4 training functions (*y*1, *y*2, *y*3, *y*4). At the end, you will have 4 ideal functions (one for each training function) that minimize the sum of squared deviations.

### 2.4.5 Verify Uniqueness

Ensure that the selected 4 ideal functions are unique. If two training functions select the same ideal function, choose the next best ideal function (with the next lowest SSE) for one of them.

### 2.4.6 Final Selection

The 4 ideal functions with the smallest SSE for each training function are the best fits based on the Least Squares Method

## 2.5 Data Mapping

This Python script is designed to fetch data from SQLite database (Python\_Assignment\_v2), select the best-fit ideal functions from a set of 50 provided functions, and map test data to these selected ideal functions by computing the maximum deviation and ensures that the test data's deviation does not exceed the threshold. The script uses the pandas, numpy, and sqlalchemy libraries for data handling and database operations.

### 2.5.1 Logic Flow

Mapping.py file serves as the logic for mapping test data to ideal functions.

The goal is to determine which of the four selected ideal functions best represents or fits each test data point and this is done by comparing the test data points to the four ideal functions and checking if they meet a predefined criterion.

DataHandlin, FunctionSelectio, TestDataMapper implement core algorithms:

DataHandling - loads data from SQLite tables (train\_data, ideal\_data, test\_data).

FunctionSelection - finds the best-fit functions by computing the lowest SSE.

TestDataMapper - Computes max deviation for selected ideal functions, maps test data points based on deviation constraints and saves results to the database.

**Understanding the Mapping Criterion**

Breaking Down the Requirements

This description outlines the steps your Python program must follow to map test data (B) to the four chosen ideal functions, store the results, and ensure that deviations remain within the specified threshold. Below is a structured breakdown of what each part means:

**1. Assign Each Test Data Point (B) to One of the Four Ideal Functions**

For each (x, y) pair in the test data:

Determine whether it can be mapped to one of the four ideal functions selected in the previous step.

If a match is found, store the mapping along with the computed deviation.

**2. Mapping Criterion Using Maximum Deviation (√2 Factor)**

The criterion states that a test data point (x test,y test) should be assigned to one of the **four chosen ideal functions** if:

Δytest ≤ × max ( Δytrain )

**Δytest​** = ∣ytest−yideal∣ - Deviation of the test point from the ideal function

**max(Δytrain​)** -. Maximum deviation between the training dataset (A) and the corresponding ideal function (C) assigned to it.

= A predefined scaling factor that allows some tolerance for the test data.

**3. If the Test Data Meets the Criterion, Assign It to an Ideal Function**

If the test point meets the threshold condition, assign it to the **closest** matching ideal function.

If multiple functions meet the criterion, assign the one with the **smallest deviation**.

**4. Store the Results in a New Table in the SQLite Database**

A screenshot of a black screen

Description automatically generated

*Figure 3: Table structure of Mapped Test Data in SQLite*

**5. Finding the maximum deviation between each training function and its assigned ideal function**

Calculate the absolute deviation between each training function and its corresponding best-fit ideal function

Example: If y1 (training) is best matched to y42 (ideal), compute:

Max (Δytrain) = max (|ytrain − yideal∣)

**6. For each test point (xtest,ytest ​), check all four chosen ideal functions**

Compute deviation: Δytest = ∣ytest − yideal∣ and Store the maximum deviation for each assigned training-ideal pair.

**7. Compare against the threshold**

If Δytest ≤ × max (Δytrain) , assign the test point to that ideal function.

### 2.5.2 Imports

The script begins by importing necessary modules like **DBM, Dataset\_manager** fromCSV\_DB.py for database management and data loading

### 2.5.3 Data Handling

The DataHandling class is responsible for fetching data from the SQLite database and initializing the data for further processing.

**\_\_init\_\_**

* The \_\_init\_\_ method initializes the database connection using SQLAlchemy.
* It fetches data from three tables: train\_table, ideal\_table, and test\_table.
* The data is stored in pandas DataFrames: train\_data, ideal\_data, and test\_data.
* The method includes error handling for database errors and other unexpected issues.

**get\_columns**

* This method returns a list of columns from a DataFrame that start with a specified prefix (e.g., 'y').
* It is used to extract the y columns from the training and ideal data.

### 2.5.4 Function Selection (FunctionSelection)

The FuncSelection class is a subclass of DataHandling and is responsible for selecting the best-fit ideal functions using the least squares method.

**\_\_init\_\_**

* The \_\_init\_\_ method initializes the FuncSelection object and extracts the x and y columns from the training and ideal data.
* It uses the get\_columns method to get the y columns from the training and ideal data.

**find\_best\_fit**

* This method selects the best 4 ideal functions that minimize the sum of squared deviations (Least-Square) between the training data and the ideal functions.
* It calculates the Sum of Squared Errors (SSE) and Mean Squared Error (MSE) for each training column and ideal function pair.
* The method returns a dictionary of the best-fit ideal functions for each training column, sorted by the lowest MSE.

## 2.6 Visualization

This section we see how to leverages Bokeh's core plotting and output functions to generate interactive visualizations. Creating a figure model as the canvas for all glyphs (scatter, line, etc.) and plot customizations. Through Bokeh we are able to customize the file Bokeh creates for visualization, import and call the output\_file() function, output\_file() accepts various file-related arguments. (Bokeh Development Team, 2021)

### 2.6.1Imports

**Bokeh** (bokeh.plotting, bokeh.io): For interactive plotting.

**Custom Modules** (mapping.DataHandling, mapping.TestDataMapper): For data extraction and mapping.

### 2.6.2 Workflow

The main purpose of the file visualization.py  is to provide interactive visualizations for a data mapping workflow involving training data, ideal functions, and mapped test data. It leverages the [Bokeh](https://bokeh.org/" \t "_blank) library for interactive plotting and integrates with a data pipeline that uses SQLite databases and mapping logic.

We first load data to retrieve training data, ideal functions, and mapped test data from SQLite database then visualize how training data aligns with its best-fit ideal functions as shown in *figure 4 from the best\_fit\_visualization.html* file and display how test data points are mapped to these ideal functions *figure 5*.

A graph of a graph showing different colored lines

Description automatically generated

*Figure 4: Best-fit ideal function for training data*

A graph of colored lines

Description automatically generated

*Figure 5: Visualization of Mapped data, Training data and chosen Ideal functions*

## 2.7 Test

The implementation of SQLite as an in-memory database, facilitated by SQLAlchemy, is an advantageous approach for transient test cases. This allows for rapid database creation and manipulation without the overhead of disk I/O, which is beneficial for testing scenarios where rapid feedback is critical (Lemenkova, 2019; . Using such a testing setup is supported in the literature, where it enhances the ease of managing test data and ensures that tests run in isolation from one another, further emphasizing the importance of a clean test environment (Lemenkova, 2019; .

The specific tests coded within the classes—such as validating data loading and ensuring that the correct columns are identified—are representative of standard practices in data science and engineering where validating assumptions about data structures is imperative Bantilan, 2020). This aligns with practices reported in the domain where empirical data validation ensures that subsequent analysis is based on accurately processed datasets Bantilan, 2020).

The overarching theme in this code is the integration of good software engineering principles with data science processes, ensuring thorough testing, reliable data handling, and efficient data manipulation. This combination is frequently discussed in the context of modern data science workflows, which prioritize reliability, reproducibility, and efficiency (Lemenkova, 2019; , Bantilan, 2020).

### 2.7.1 Unit Testing for Python Program

In object-oriented programming languages, unit testing involves verifying the behavior of class’ public functions. Unit testing can provide insight into the location of a program fault by evaluating each function’s behavior separately. (Buffardi & Aguirre-Ayala, 2021)

Unit testing is a crucial part of software development that ensures the correctness and reliability of a program. In this document, we will discuss the unit testing methodology applied to our Python program. The tests focus on verifying data handling, test data mapping, visualization, and database interactions.

Use this format *python -m unittest <test\_nameoffile.py>* on terminal to run unittest.

### 2.7.2 Overview of the Testing Approach

The primary goal of the unit tests is to verify that the core functionalities of the program work as expected. The testing strategy involves:

1. Test Fixtures: T*est* fixture represents the preparation needed to perform one or more tests, associated cleanup actions and, are perforemed before and after the test. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process(Python Software Foundation, n.d.).

2. Mocking External Dependencies: Since the program relies on file operations and database access, the ‘*unittest.mock*’ library is used to simulate these dependencies without requiring actual files or databases.The [patch()](https://docs.python.org/3/library/unittest.mock.html" \l "unittest.mock.patch" \o "unittest.mock.patch) context manager makes it easy to mock classes, the object you specify will be replaced with a mock (or other object) during the test and restored when the test ends(Python Software Foundation, n.d.).

3. Creating Temporary Test Data: The tests generate synthetic CSV data and store them in a temporary directory to simulate real datasets.

4. Checking Functionality of Core Components: The test cases focus on various modules such as data handling, test mapping, and visualization.

5. Ensuring Error Handling: The tests include scenarios where expected errors (such as missing files) are correctly handled by the program.

### 2.7.3 Structure of the Unit Tests

1. test\_main.py

This test verifies the execution of the main program and it mocks data loading, processing, and visualization functions to ensure that the workflow runs without issues.

Steps Taken:

A temporary directory `test\_data` is created with subdirectories and dummy CSV files (`train.csv`, `ideal.csv`, `test.csv`).

The `unittest.mock.patch` function is used to mock:

* Data loading (`load\_data`)
* Data handling (`DataHandling` class)
* Test data mapping (`TestDataMapper` class)
* Visualization functions
* The `main()` function is executed with the patched dependencies.

Assertions are made to ensure that all necessary methods were called and an additional test case checks if a `FileNotFoundError` is raised when files are missing and the test directory is cleaned up after execution.

2. test\_CSV\_DB.py

This test verifies that the database operations function correctly when reading from CSV files and writing results.

Steps Taken:

* Mock datasets are used to simulate CSV input.
* The database connection is established with SQLite using SQLAlchemy.
* The test checks whether data is correctly inserted and retrieved from the database.
* Exception handling is tested for cases where database connections fail or files are missing.

3. test\_mapping.py

This test ensures that test data is correctly mapped to the best-fit ideal functions.

Steps Taken:

* Mocked training data, ideal data, and test data are defined.
* The `TestDataMapper` class is instantiated with these datasets.
* The `find\_best\_fit` method is tested to check whether it selects the appropriate ideal function based on least-square deviation.
* The `map\_test\_data` method is tested to verify that test points are correctly assigned to their ideal functions.

4. test\_visualization.py

This test ensures that the visualization functions generate output correctly without errors.

Steps Taken:

The `create\_best\_fit\_visualization` and `create\_mapped\_data\_visualization` functions are called with mock data.

The test asserts that these functions complete execution without raising exceptions.

**Ensuring Smooth Execution of Tests**

To ensure that the unit tests run smoothly, the following best practices are applied:

1. Isolated Test Environments: Each test case runs independently with fresh mock data to avoid unintended side effects. (MuukTest, n.d.)

2. Use of Mocks: External dependencies like file operations, database queries, and visualizations are replaced with mocks to speed up testing and avoid unnecessary complexity. (MuukTest, n.d.)

3. Assertion of Expected Calls: Assertions are the backbone of unit tests and verify that the outcome of a test matches the expected result. Using a variety of assertions to check different aspects of the code, such as equality, exceptions, and object properties. (Pixel Free Studio, n.d.). In this case, after running `main()`, assertions confirm that critical methods were called as expected.

4. Error Handling Verification: Edge cases such as missing files and incorrect mappings are explicitly tested to verify that errors are correctly handled and expose hidden bugs. This practice improves the robustness of the code and prevents unexpected failures. (Pixel Free Studio, n.d.)

5. Automated Cleanup: Temporary test files and directories are removed after test execution to keep the environment clean by ensuring that each test runs in isolation and doesn’t affect others. Setup and teardown methods are used to initialize and clean up the test environment in order to avoid a shared state between tests, prevent interference, and ensure reliable results. (Pixel Free Studio, n.d.)

Conclusion

The unit tests are designed to validate the correctness and robustness of the program while ensuring maintainability. The combination of isolated test environments, mocks, and thorough assertions guarantees that the program runs as expected under various conditions.

## 2.8 Version Control

Git is a distributed version control system that enables comprehensive tracking of all changes made to a project’s codebase, recording what was changed, who made the changes, and when they occurred (GitHub, n.d.). This detailed history allows developers to easily revert to previous versions if bugs are introduced or when auditing past work, providing a robust safety net during development. (Abod, 2023).

Git’s branching capabilities allow teams to create isolated branches for new features or bug fixes without impacting the main codebase, supporting parallel development and experimentation. Once changes are tested and validated, these branches can be merged back into the main branch, streamlining collaboration and minimizing disruption (Abod, 2023).

GitHub, as a cloud-based hosting platform for Git repositories, adds further value through its pull request and code review features. Pull requests facilitate structured discussions and reviews of proposed changes before they are merged, enabling teams to maintain high code quality and catch potential issues early in the development process. The code review process is integrated with tools for commenting, suggesting improvements, and verifying that all changes meet the team’s standards (LinearB, 2023)

Storing code on GitHub also ensures secure, cloud-based backups, reducing the risk of data loss due to mistakes or hardware failures. This centralized, remote repository acts as a single source of truth, making it easy for developers to synchronize their local work, recover from errors, and collaborate efficiently from any location (GitHub, n.d.).

1. Clone the repository to your local machine

git clone <repo-url>

cd <repo-name>

2. Check out the develop branch

git checkout develop

3. Pull the latest updates from the remote develop branch

git pull origin develop

4. Create a new feature branch to work on your function:

git checkout -b feature/add-new-function

5. Add your new function to the codebase (in VS Code).

6. Check the status of your changes:

git status

7. Stage the changed files:

git add .

8. Commit the changes with a meaningful message:

git commit -m "Add new function for [describe what it does briefly]"

9. Push your feature branch to the remote repo:

git push origin feature/add-new-function

10. Create a pull request (PR) on GitHub to merge feature/add-new-function into develop.

# 3. Results

The project successfully loads the datasets, selects the best-fit ideal functions (*figure 4)*, maps the test data, and visualizes the results(*figure 5*). The interactive Bokeh file *visualization.html* plot provides a clear interactive visual interface for understanding the results of the data mapping process.

# 4. Conclusion

This project demonstrates the use of Python for data processing, analysis, and visualization. It highlights the importance of object-oriented design, thorough documentation, and testing in software development. The project can be further enhanced by addressing the suggested improvements.

# References

For the structure of the document look into this link <https://www.nature.com/articles/s41598-018-22592-3>

1. McKinney, W. (2017). *Python for Data Analysis*. O'Reilly Media.
2. VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media.

Proven

DataCamp. (n.d.). *Comprehensive tutorial on using pathlib in Python for file system manipulation*. Retrieved April 3, 2025, from <https://www.datacamp.com/tutorial/comprehensive-tutorial-on-using-pathlib-in-python-for-file-system-manipulation>

[1] Bantilan, N. (2020). Pandera: statistical data validation of pandas dataframes., 116-124. <https://doi.org/10.25080/majora-342d178e-010>

[2] Lemenkova, P. (2019). Processing oceanographic data by python libraries numpy, scipy and pandas. Aquatic Research, 73-91. https://doi.org/10.3153/ar19009

[3] Melchert, O., Meinhardt‐Wollweber, M., & Roth, B. (2018). An efficient procedure for custom beam-profile convolution in polar coordinates: testing, benchmarking and application to biophotonics. Biomedical Physics &Amp; Engineering Express, 4(2), 025025. <https://doi.org/10.1088/2057-1976/aaa51a>

[4] Myers, J., Copeland, R., & Copeland, R. D. (2015). *Essential SQLAlchemy*. " O'Reilly Media, Inc.".

[5] Chen, Z., Jia, C., Li, Y., & Chen, L. (2025). Using dynamic and static techniques to establish traceability links between production code and test code on python projects: a replication study. Journal of Software: Evolution and Process, 37(3). https://doi.org/10.1002/smr.70011

[6] Holehouse, A. S. and Naegle, K. M. (2015). Reproducible analysis of post-translational modifications in proteomes—application to human mutations. Plos One, 10(12), e0144692. https://doi.org/10.1371/journal.pone.0144692

[7] Suraya, S. and Sholeh, M. (2021). Designing and implementing a database for thesis data management by using the python flask framework. International Journal of Engineering, Science and Information Technology, 2(1), 9-14. https://doi.org/10.52088/ijesty.v2i1.197

[8] Dobric, D., Pech, A., Ghita, B., & Wennekers, T. (2022). On the importance of the newborn stage when learning patterns with the spatial pooler. SN Computer Science, 3(2). https://doi.org/10.1007/s42979-022-01066-4

[9] pandas.DataFrame.to\_sql. (n.d.). *pandas.pydata.org*. Retrieved April 4, 2025, from <https://pandas.pydata.org/pandasdocs/stable/reference/api/pandas.DataFrame.to_sql.html>

[10] JMP. (n.d.). *The method of least squares*. Retrieved April 3, 2025, from <https://www.jmp.com/en/statistics-knowledge-portal/what-is-regression/the-method-of-least-squares>

[11] Murphy, Kevin P.. *Machine Learning : A Probabilistic Perspective*, MIT Press, 2012.*ProQuest Ebook Central*, <http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=3339490> .  
Created from badhonnef on 2025-04-15 19:26:09.

[12] Buffardi, K., & Aguirre-Ayala, J. (2021, June). Unit test smells and accuracy of software engineering student test suites. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (pp. 234-240).

[13] Python Software Foundation. (n.d.). *unittest.mock — mock object library*. Python 3.12. <https://docs.python.org/3/library/unittest.mock.html>

[14] MuukTest. (n.d.). *Python unit test guide: Everything you need to know*. MuukTest. [https://muuktest.com/blog/python-unit-test-guide](https://muuktest.com/blog/python-unit-test-guide" \t "_new)

[15] Pixel Free Studio. (n.d.). *Best practices for unit testing to ensure code quality*. Pixel Free Studio. [https://blog.pixelfreestudio.com/best-practices-for-unit-testing-to-ensure-code-quality/#aioseo-use-assertions-effectively](https://blog.pixelfreestudio.com/best-practices-for-unit-testing-to-ensure-code-quality/" \l "aioseo-use-assertions-effectively" \t "_new)

[16] Bokeh Development Team. (2021). *Bokeh: Python library for interactive visualization*. [https://bokeh.org/](https://bokeh.org/" \t "_new)

[17] SQLAlchemy Documentation. (2021). *SQLAlchemy: The Database Toolkit for Python*. <https://www.sqlalchemy.org/>

[18] GitHub. (n.d.). About Git. GitHub Docs. [https://docs.github.com/en/get-started/using-git/about-git](https://docs.github.com/en/get-started/using-git/about-git" \t "_blank)

[19] Abod, F. (2023, July 7). The importance of version control: Git and GitHub, SSH and HTTPS. *DEV Community*. [https://dev.to/fredabod/the-importance-of-version-control-git-and-github-ssh-and-https-40o](https://dev.to/fredabod/the-importance-of-version-control-git-and-github-ssh-and-https-40o" \t "_blank)

[20] LinearB. (2023, May 22). Code review on GitHub: The right way to do it. *LinearB Blog*. [https://linearb.io/blog/code-review-on-github](https://linearb.io/blog/code-review-on-github" \t "_blank)

# Images

Best-fitA graph of a graph showing different colored lines

Description automatically generated

Mapped Test Data

A graph of colored lines

Description automatically generated

# CODE