Name:Diyora Radhika Ranchhodbhai

Course of study:M.SC.Data Science(120 ECTS)

Course of name:Programming with Python(DLMDSPWP01)

Matriculation number:102303044

Tutor name:Ugur Ural

Table of Contents

List of Figures

1. Introduction

Rationale for Topic Selection

A significant number of data science, regression, and automatic model selection tools rely on how functions are fitted to data. It demonstrates how to do this by carefully analysing data and picking the right approach (Vicente-Gonzalez *et al.,* 2025). It helps with selecting the optimal predictive model, decreasing mistakes, and increasing the accuracy of separating data within limits. With this, applications that apply pattern recognition and signal processing are capable of making selections independent of human effort. Accuracy in machine learning and statistics increases by making sure that test results are properly linked to models.

Aim

The aim is to use Python and SQL to identify appropriate functions for our training data through least squares, project points from the test data according to set thresholds and efficiently save and show our results.

Scope and boundaries

The objective of this work is to fit functions and compare them with test data using the least squares method within given deviation limits. Optimisation and similar ways to fit the data are skipped in this book (Noma and Gosho, 2025). All the analysis is done exclusively with the data in the CSV files. For showcasing their results, it makes the graphs interactive and easy to understand by using Bokeh in Python. Everything training, ideal and test mapping data is managed and controlled with SQL.

Report structure outline

First, it is describe which data is used, how the databases are organised, and the methods used to fit and map new pieces of data in the Methodology section. Next, read the Implementation part to find out how Python works, its key classes and how the programme handles its tasks. Here, the leading algorithms review data test results and visualise their key points in images. In conclusion, the important findings are summarised, the success of the process is considered, and suggestions for future progress are mentioned.

2. Main Body

2.1 Methodology

Datasets Description

The project includes three files, CSVs with x and y coordinates for each record. Training takes place with four unique setups that each provide scatterplots containing x-y pairs, leading to function fitting. Fifty candidate functions come as x-y pairs in the ideal functions dataset, and all can be used to see what fits the training data best (Mo *et al.,* 2025). Every example in the test set is represented by an x-y coordinate, which is compared in each case to the ideal to gauge the distance. Using Python to organise and analyse data is made easy because all the datasets are in the same format.

Database schema and reasoning

The database organises three tables labelled training data, ideal functions and test mapping. Each exercise's X-values are included in both the training set and the ideal set, but the training set includes four training functions and, unlike the ideal set, no candidate functions. The data in the test mapping table includes points, their differences and the optimal function that fits those differences (Diana *et al.,* 2025). Many creators of serverless applications choose SQLite because the data is compressed in a single file. With SQLAlchemy and Python, data can be quickly and conveniently found and altered. Therefore, data remains secure, simple to access and managed properly, no matter how many records exist.

Fitting algorithms

During identification, the system uses least squares to map the right model onto every training function. At every point, they multiplied the difference between their model prediction and the candidate's result twice. After that, the algorithm takes its errors together and compares everyone to the ideal function to find the most accurate match for its training data (Vicente *et al.,* 2025). They find the predicted output by finding the function where the total squared difference between all the data and the prediction is lowest. For this reason, they can make predictions easily, fast and with fewer chances for errors.

Mapping test data

To map test data intelligently, everyone must follow the same type of deviation. The largest difference is found in the y-values for every form of training and function type. Most map generation errors are caused by multiplying training differences by the square root of two. They calculate the gap in y-values between their points and the y-values that come from the ideal curve (Heng *et al.,* 2025). A small change in the equation makes the test values line up with the best-fitting line. For this reason, data is organised by testing and maintained safely and accurately in place.

Software design

By using object-oriented development, the software adopts a “FunctionFitter” technique to choose the right function for the training data and minimise errors. With the help of the FunctionFitter, TestDataMapper maps test data according to deviation standards. In the system, graceful error exceptions are offered for handling Pandas data, and SQLAlchemy connectivity to SQLite is included (Vicente *et al.,* 2025). Because the code is divided into modules, it is simpler to take care of, describe and trust.

Unit testing

Unit testing checks if their main fields are in order. As a result of these tests, they confirm that least squares is the best option to judge how precisely each model fits known data. They reveal that, during training, the algorithm consistently picks the top function to use when selecting from others (Riani *et al.,* 2025). For applied tests, they make sure there aren’t errors when processing arrays of the same size, that edges are treated correctly and that indices are not overrun. With the help of these tests, developers can fix any problems shortly after starting to work on the product.

2.2 Implementation

Code organisation

```python
# ========== IMPORTING ==========
import os
import math
import zipfile
import pandas as pd
import numpy as np
from sqlalchemy import create_engine, text
from bokeh.plotting import figure, show
from bokeh.layouts import gridplot
import unittest
from bokeh.io import output_notebook
output_notebook()
```

```python
[77] # ========== Extracting Dataset ==========
ZIP_FILE_PATH = '/content/Dataset2.zip'
EXTRACT_DIR = '/content/unzipDataset'

os.makedirs(EXTRACT_DIR, exist_ok=True)

with zipfile.ZipFile(ZIP_FILE_PATH, 'r') as zip_ref:
    zip_ref.extractall(EXTRACT_DIR)

train_df = pd.read_csv(os.path.join(EXTRACT_DIR, 'train.csv'))
ideal_df = pd.read_csv(os.path.join(EXTRACT_DIR, 'ideal.csv'))
test_df = pd.read_csv(os.path.join(EXTRACT_DIR, 'test.csv'))

print("Training Data Sample:")
print(train_df.head())
print("\nIdeal Functions Sample:")
print(ideal_df.head())
print("\nTest Data Sample:")
print(test_df.head())
```

Figure 1: Dataset Extraction and Loading

(Source: Google Colab)

The code brings in required libraries, organises a directory for extraction and decompresses the dataset file. After that, it adds three data files of training data, ideal functions and test data to pandas DataFrames. Next, the programme outputs the first few rows of each dataset to verify the loading and how it's organised.

```python
# ========== Setup SQLite database engine ==========
engine = create_engine('sqlite:///fitting_results.db', echo=False)

# ========== Save dataframes to SQLite ==========
train_df.to_sql('training_data', con=engine, if_exists='replace', index=False)
ideal_df.to_sql('ideal_functions', con=engine, if_exists='replace', index=False)

# ========== Create test_mapping table if it doesn't exist ==========
with engine.connect() as conn:
    conn.execute(text("""
        CREATE TABLE IF NOT EXISTS test_mapping (
            X REAL,
            Y REAL,
            Delta_Y REAL,
            Ideal_func INTEGER
        )
    """))
    conn.commit()
```

Figure 2: SQLite Database Setup and Table Creation

(Source: Google Colab)

The SQLite database communicates with the programme through the SQLAlchemy "create_engine" function. Any training and ideal functions data is put into the database and referred to as "training_data" and "ideal_functions," if there are similar tables already. Once that's done, it joins the connection and builds a test_mapping table if it has not already been built. This structure allows every record to add test results, errors found and the correct function output, which helps store mapping details organised in the database.

```python
# ========== Function Fitter class ==========
class FunctionFitter:
    """
    Fits the four training functions (y1..y4) to the best matching ideal functions (y1..y50)
    by minimizing sum of squared deviations.
    """
    def __init__(self, train_df, ideal_df, engine):
        self.train_df = train_df
        self.ideal_df = ideal_df
        self.engine = engine
        self.selected_funcs_indices = []

    def least_squares(self, y_true, y_pred):
        """Calculate sum of squared errors."""
        return np.sum((y_true - y_pred) ** 2)

    def find_best_fits(self):
        """Find the ideal function indices (1 to 50) best fitting each training function y1..y4."""
        try:
            selected = []
            for train_i in range(1, 5):  # y1 to y4
                train_y = self.train_df[f'y{train_i}'].values
                min_error = float('inf')
                best_idx = None

                for ideal_j in range(1, 51):  # y1 to y50
                    ideal_y = self.ideal_df[f'y{ideal_j}'].values
                    error = self.least_squares(train_y, ideal_y)

                    if error < min_error:
                        min_error = error
                        best_idx = ideal_j

                selected.append(best_idx)

            self.selected_funcs_indices = selected
        except Exception as e:
            raise DataProcessingError(f"Error finding best fits: {e}")

    def max_abs_deviation(self, train_y, ideal_y):
        """Calculate maximum absolute deviation between training and ideal y-values."""
        return np.max(np.abs(train_y - ideal_y))
```

Figure 3: FunctionFitter Class Implementing Least Squares Fitting

(Source: Google Colab)

The ‘FunctionFitter’ class is improved using the ‘Regular Nearest Neighbour’ method and reducing the square differences between the top four functions and the sample data. Both sets of data and the database engine are provided automatically when the class is initialised. In the presence of outlying observations, they use the least-squares method to learn how far apart the arrays are. For all approaches to filling in the blanks, the function picks the best result or explains its wrongness based on comparison with the training functions. It makes clear the biggest difference between learning computers and how students should be taught

```python
# ========== Test Data Mapper class ==========
class TestDataMapper(FunctionFitter):
    """
    Maps test points to one of the selected ideal functions if deviation criterion is met.
    """
    def __init__(self, train_df, ideal_df, test_df, engine):
        super().__init__(train_df, ideal_df, engine)
        self.test_df = test_df

    def map_test_data(self):
        if not self.selected_funcs_indices:
            raise DataProcessingError("Best fit functions not selected before mapping.")

        try:
            max_devs = []
            for i, ideal_idx in enumerate(self.selected_funcs_indices, start=1):
                train_y = self.train_df[f'y{i}'].values
                ideal_y = self.ideal_df[f'y{ideal_idx}'].values
                max_devs.append(self.max_abs_deviation(train_y, ideal_y))

            mappings = []
            for _, row in self.test_df.iterrows():
                x_test, y_test = row['x'], row['y']

                best_func = None
                best_dev = float('inf')

                for i, ideal_idx in enumerate(self.selected_funcs_indices):
                    ideal_rows = self.ideal_df[self.ideal_df['x'] == x_test]
                    if ideal_rows.empty:
                        continue
                    ideal_y = ideal_rows.iloc[0][f'y{ideal_idx}']

                    deviation = abs(y_test - ideal_y)
                    threshold = max_devs[i] * math.sqrt(2)

                    if deviation <= threshold and deviation < best_dev:
                        best_dev = deviation
                        best_func = ideal_idx

                if best_func is not None:
                    mappings.append((x_test, y_test, best_dev, best_func))

            # Save mappings to DB
            with self.engine.connect() as conn:
                conn.execute(text("DELETE FROM test_mapping"))
                for x, y, delta_y, func_no in mappings:
                    conn.execute(
                        text("INSERT INTO test_mapping (X, Y, Delta_Y, Ideal_func) VALUES (:x, :y, :dy, :fn)"),
                        {"x": x, "y": y, "dy": delta_y, "fn": func_no}
                    )
                conn.commit()

            return mappings
        except Exception as e:
            raise DataProcessingError(f"Error mapping test data: {e}")
```

Figure 4: TestDataMapper Class Implementing Test Data Mapping

(Source: Google Colab)

“TestDataMapper” (which is a subclass of “FunctionFitter”) pairs test data points with the functions that suit best according to the criteria given. For any selected ideal function, its maximum deviation is worked out, and the test points are examined in order. At every test point, the differences from the ideal function are studied. That is within the appropriate range (the trained multiplication of the deviation by “$\sqrt{2}$”) is connected with the main function that matches best. All of the data from the map is written to a SQLite database table to aid with future use.

```python
# ========== Visualization class ==========
from bokeh.io import output_notebook
from bokeh.plotting import figure, show
from bokeh.layouts import gridplot

class Visualizer:
    def __init__(self, train_df, ideal_df, selected_indices, test_mappings):
        self.train_df = train_df
        self.ideal_df = ideal_df
        self.selected_indices = selected_indices
        self.test_mappings = test_mappings

    def plot(self):
        output_notebook()  # Enable notebook output

        # Define separate color palettes for clarity
        train_colors = ['navy', 'olive', 'firebrick', 'goldenrod']
        ideal_colors = ['purple', 'teal', 'darkorange', 'darkgreen']
        test_colors = ['crimson', 'darkcyan', 'darkmagenta', 'darkgoldenrod']

        # Plot training functions
        p_train = figure(title="Training Functions", x_axis_label='x', y_axis_label='y')
        for i, color in enumerate(train_colors, start=1):
            p_train.line(self.train_df['x'], self.train_df[f'y{i}'], legend_label=f'Train y{i}', color=color)

        # Plot selected ideal functions
        p_ideal = figure(title="Selected Ideal Functions", x_axis_label='x', y_axis_label='y')
        for i, idx in enumerate(self.selected_indices):
            color = ideal_colors[i % len(ideal_colors)]
            p_ideal.line(self.ideal_df['x'], self.ideal_df[f'y{idx}'], legend_label=f'Ideal y{idx}', color=color)

        # Plot test data points mapped to ideal functions using scatter to avoid deprecation warning
        p_test = figure(title="Test Data Mapping", x_axis_label='x', y_axis_label='y')
        func_color_map = {func_idx: test_colors[i % len(test_colors)] for i, func_idx in enumerate(self.selected_indices)}

        for x, y, _, func_idx in self.test_mappings:
            color = func_color_map.get(func_idx, 'black')
            p_test.scatter([x], [y], marker="circle", size=6, color=color, alpha=0.6)

        # Arrange the three plots horizontally
        grid = gridplot([[p_train, p_ideal, p_test]], sizing_mode='stretch_width')

        # Show the grid of plots
        show(grid)
```

Figure 5: Visualizer Class Implementing Data Visualisation

(Source: Google Colab)

It creates the Visualiser class, which is in charge of drawing training functions, the best ideal outcomes and the mappings of test data using Bokeh. The first step is to use the datasets and mapping results, after which it separates the data into three different plots with clear colours. The training results appear as lines, while test results are marked as coloured dots according to their assigned functions. Each plot is listed side by side so that readers can evaluate them side by side. The interactive tool makes it possible to check how well the functions fit the data and how accurately the tests are mapped.

Visualisation

With the visualisation tool, training functions, chosen ideal functions and test data are all mapped using Bokeh. Distinct lines in different colours are plotted alongside the original curves to display how the data is trained. Radiologists can compare different functions using the various colours assigned to each. For every test case, there is a coloured scatter marker shown at the value the function calculates best for. When the three plots are arranged in a grid layout, they can be seen together, side by side, for no surprises. Colour coding allows us to easily tell the difference between datasets. You can use this interactive visualisation to understand both how well the experiments fit and the results of any mapping experiments.

Error and exception handling

Error codes tell them what to do when they need to update their data or maps. Problems with the database can cause web pages to be served up more quickly by the server. A database is protected as long as all data transmissions are properly safeguarded. Noticing a little detail they didn’t notice earlier might help them find the true meaning. For this reason, Programme Central can take care of all their event bookings.

Running Unit Tests

Unit tests are simple to start; thanks to some easy commands they can use. After finishing the test run, the software looks to see if every needed feature has been installed. Check that the animal listed on the map is in the location shown in the picture. After they sort their data well, the software places the best-fitting curve above the least squares curve.

3. Results and Discussion

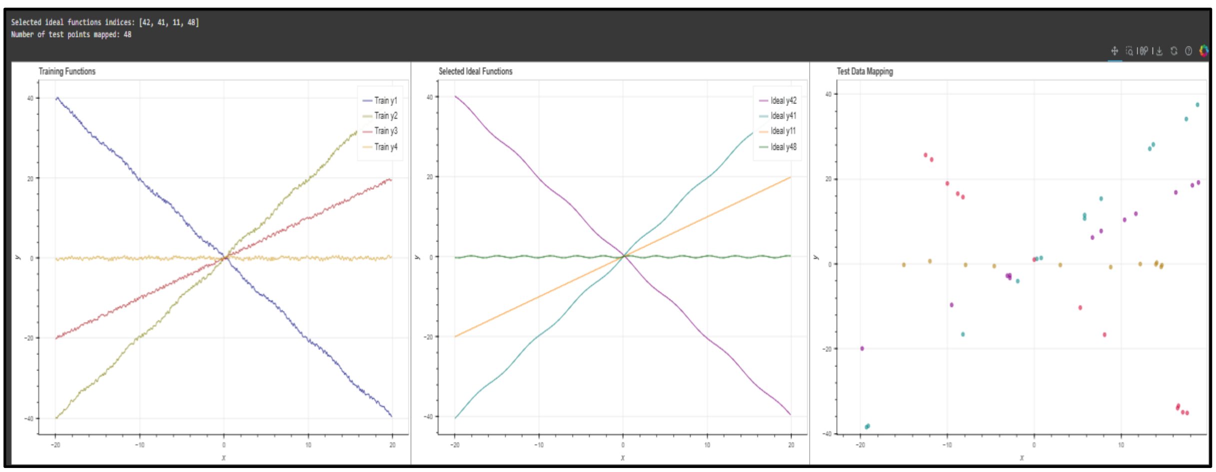

Figure 6: Visualisation of Training Functions, Selected Ideal Functions, and Test Data Mapping

(Source: Google Colab)

There are three plots on this figure. The left graph shows the four training functions by drawing coloured lines for their y-values against their x-values. The corresponding selected ideal functions for every training function are shown in the middle plot. Scatter markers, colour-coded by their assigned function, show the data points mapped to the selected ideal models. By displaying the results together, a simple picture of the model's strengths and weaknesses is presented.

```sql
-- 1. Show all tables in the SQLite database
SELECT name
FROM sqlite_master
WHERE type = 'table';
```

	name
1	test_mapping
2	training_data
3	ideal_functions
4	sqlite_sequence

Figure 7: Show all tables

(Source: SQLite Database)

In this figure, they use a "sqlite_master" query to check and display the table contents in the database. As a consequence, the model produces four new tables named "test_mapping", "training_data", "ideal_functions" and "sqlite_sequence". Every database looks after mapped test data, training data, function data and SQLite information on sequences. All of the project information is now listed in tables, so they can easily add and take out data.

Table 1: The training data database table

```sql
-- 2. Show the first 20 rows of each relevant table
-- Training data (4 training functions)
SELECT * FROM training_data LIMIT 20;
```

	x	v1	v2	v3	v4
1	-20	39.778572	-40.07859	-20.214268	-0.32491425
2	-19.9	39.604813	-39.784	-20.07095	-0.058819864
3	-19.8	40.09907	-40.018845	-19.906782	-0.4518296
4	-19.7	40.1511	-39.518402	-19.389118	-0.6120442
5	-19.6	39.795662	-39.360065	-19.81589	-0.3060756
6	-19.5	39.340855	-38.90581	-19.287113	-0.062154666
7	-19.4	39.25246	-39.12036	-19.683708	0.026392838
8	-19.3	38.590164	-38.62107	-19.494537	-0.2690418
9	-19.2	38.893463	-38.806778	-19.533716	0.08567329
10	-19.1	38.364567	-38.354656	-18.75372	-0.29954198
11	-19	38.13553	-37.795067	-19.363068	-0.553223
12	-18.9	37.825813	-37.984848	-18.528309	0.040018722
13	-18.8	37.672874	-37.855568	-18.92033	0.0705662
14	-18.7	37.99707	-37.64723	-18.858202	0.09534555
15	-18.6	37.38134	-37.436577	-18.457552	-0.3021374
16	-18.5	37.429653	-37.083054	-18.551897	-0.0018355072
17	-18.4	37.330147	-36.65041	-18.406178	0.34796613
18	-18.3	36.635334	-36.5591	-18.573486	0.08883932
19	-18.2	36.31057	-35.691395	-18.298632	-0.15387341
20	-18.1	36.379223	-35.42202	-17.694733	0.6145841

The above table displays only the first 20 rows from the “training_data” in the SQLite database. The final data is made up of the x column of input values and four v columns that give the y values for different training functions. The chart makes clear that each training function changes in a specific manner with x. It sets up an efficient method for querying and analysis, giving us the main dataset to adjust ideal functions and assess variations both during function fitting and testing.

Table 2: The ideal functions' database table

```sql
-- Ideal functions (50 ideal functions)
SELECT * FROM ideal_functions LIMIT 20;
```

	x	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11	v12
1	-20	-0.9129453	0.40808207	9.087055	5.408082	-9.087055	0.9129453	-0.8390715	-0.85091937	0.81616414	18.258905	-20	-58
2	-19.9	-0.8676441	0.4971858	9.132356	5.4971857	-9.132356	0.8676441	-0.8652126	0.16851768	0.9943716	17.266117	-19.9	-57.7
3	-19.8	-0.81367373	0.58132184	9.186326	5.5813217	-9.186326	0.81367373	-0.88919115	0.6123911	1.1626437	16.11074	-19.8	-57.4
4	-19.7	-0.75157344	0.65964943	9.248426	5.6596494	-9.248426	0.75157344	-0.91094714	-0.99466854	1.3192989	14.805996	-19.7	-57.1
5	-19.6	-0.6819636	0.7313861	9.318036	5.731386	-9.318036	0.6819636	-0.9304263	0.7743557	1.4627723	13.366487	-19.6	-56.8
6	-19.5	-0.60553986	0.795815	9.39446	5.795815	-9.39446	0.60553986	-0.9475798	-0.11702018	1.59163	11.808027	-19.5	-56.5
7	-19.4	-0.52306575	0.8522923	9.476934	5.8522925	-9.476934	0.52306575	-0.96236485	-0.59004813	1.7045846	10.147476	-19.4	-56.2
8	-19.3	-0.43536535	0.90025383	9.564634	5.900254	-9.564634	0.43536535	-0.97474456	0.97776526	1.8005077	8.402552	-19.3	-55.9
9	-19.2	-0.34331492	0.93922037	9.656685	5.9392204	-9.656685	0.34331492	-0.98468786	-0.87895167	1.8784407	6.5916467	-19.2	-55.6
10	-19.1	-0.2478342	0.96880245	9.752166	5.9688025	-9.752166	0.2478342	-0.99217	0.37579286	1.9376049	4.7336335	-19.1	-55.3
11	-19	-0.1498772	0.9887046	9.850122	5.9887047	-9.850122	0.1498772	-0.9971722	0.27938655	1.9774092	2.847667	-19	-55
12	-18.9	-0.050422687	0.998728	9.949577	5.998728	-9.949577	0.050422687	-0.99968195	-0.80255306	1.997456	0.9529888	-18.9	-54.7
13	-18.8	0.04953564	0.9987724	10.049536	5.998772	-10.049536	-0.04953564	-0.99969304	0.9999414	1.9975448	-0.93127006	-18.8	-54.4
14	-18.7	0.14899902	0.98883736	10.148999	5.9888372	-10.148999	-0.14899902	-0.99720544	-0.82669914	1.9776747	-2.7862818	-18.7	-54.1
15	-18.6	0.24697366	0.9690222	10.246974	5.9690223	-10.246974	-0.24697366	-0.99222535	0.3753813	1.9380444	-4.59371	-18.6	-53.8
16	-18.5	0.34248063	0.9395249	10.342481	5.939525	-10.342481	-0.34248063	-0.9847652	0.1825695	1.8790498	-6.3358912	-18.5	-53.5
17	-18.4	0.43456563	0.9006402	10.434566	5.90064	-10.434566	-0.43456563	-0.9748436	-0.6683636	1.8012804	-7.9960074	-18.4	-53.2
18	-18.3	0.5223086	0.85275656	10.522308	5.8527565	-10.522308	-0.5223086	-0.9624855	0.95221686	1.7055131	-9.558248	-18.3	-52.9
19	-18.2	0.6048328	0.79635245	10.604833	5.7963524	-10.604833	-0.6048328	-0.9477216	-0.98045707	1.5927049	-11.007957	-18.2	-52.6
20	-18.1	0.68131375	0.73199147	10.6813135	5.7319913	-10.6813135	-0.68131375	-0.9305889	0.77351207	1.4639829	-12.3317795	-18.1	-52.3

v40	v41	v42	v43	v44	v45	v46	v47	v48	v49	v50
899.5919	-40.456474	40.20404	2.9957323	-0.008333334	12.995732	5.2983174	-5.2983174	-0.18627828	0.9129453	0.3968496
893.5128	-40.23382	40.04859	2.9907198	-0.008340283	12.99072	5.293305	-5.293305	-0.21569017	0.8676441	0.47695395
887.4587	-40.006836	39.89066	2.985682	-0.008347246	12.985682	5.288267	-5.288267	-0.23650314	0.81367373	0.5491291
881.43036	-39.775787	39.729824	2.9806187	-0.008354219	12.9806185	5.2832036	-5.2832036	-0.24788749	0.75157344	0.6128399
875.4286	-39.54098	39.565693	2.9755297	-0.008361204	12.97553	5.278115	-5.278115	-0.24938935	0.6819636	0.6679019
869.45416	-39.30277	39.397907	2.9704144	-0.008368201	12.970414	5.273	-5.273	-0.24094884	0.60553986	0.7144341
863.5077	-39.06153	39.226147	2.9652731	-0.00837521	12.965273	5.267858	-5.267858	-0.22290246	0.52306575	0.75279135
857.5897	-38.817684	39.050125	2.9601052	-0.00838223	12.960105	5.26269	-5.26269	-0.19596967	0.43536535	0.7834847
851.7008	-38.57166	38.86961	2.9549103	-0.008389262	12.95491	5.2574954	-5.2574954	-0.16122419	0.34331492	0.80709803
845.8412	-38.323917	38.684402	2.9496884	-0.008396306	12.949688	5.2522736	-5.2522736	-0.1200512	0.2478342	0.82420814
840.0113	-38.07494	38.49435	2.944439	-0.008403362	12.944439	5.247024	-5.247024	-0.07409214	0.1498772	0.8353145
834.21124	-37.82521	38.299362	2.939162	-0.008410429	12.939162	5.241747	-5.241747	-0.025179274	0.050422687	0.840783
828.4412	-37.575233	38.099384	2.933857	-0.008417509	12.933857	5.236442	-5.236442	0.024737414	-0.04953564	0.8408071
822.7012	-37.3255	37.894417	2.9285235	-0.0084246	12.928523	5.2311087	-5.2311087	0.0736679	-0.14899902	0.83538747
816.99097	-37.07651	37.68451	2.9231615	-0.008431703	12.9231615	5.2257466	-5.2257466	0.11966148	-0.24697366	0.8243326
811.3105	-36.82876	37.46976	2.9177706	-0.008438818	12.91777	5.220356	-5.220356	0.16088453	-0.34248063	0.8072778
805.65936	-36.582718	37.25032	2.9123507	-0.008445946	12.912351	5.214936	-5.214936	0.19569363	-0.43456563	0.78372467
800.03723	-36.338844	37.02638	2.9069011	-0.008453085	12.906901	5.209486	-5.209486	0.22270104	-0.5223086	0.7530968
794.44366	-36.097584	36.798176	2.9014215	-0.008460237	12.901422	5.2040067	-5.2040067	0.24083005	-0.6048328	0.7148101
788.878	-35.859344	36.565994	2.895912	-0.0084674	12.895912	5.198497	-5.198497	0.24935794	-0.68131375	0.6683523

The information in the "ideal_functions" table is given in a snapshot here, showing fifty ideal functions represented by columns "v1" through "v50". In the first column, denoted by x, goes the input data and consecutive columns show the fitted y-values for each ideal function for those inputs. The dataset is so large that it acts as a source of possible functions to test the training data against. Because data is organised in a consistent table, picking the proper function to fit is easier and results in the best estimates.

Table 3: Test data, with mapping and y-deviation

```sql
-- Test data mapping (mapped test points with deviation and ideal function)
SELECT * FROM test_mapping LIMIT 20;
```

	X	Y	Delta Y	Ideal func
1	17.5	34.16104	0.351148	41
2	0.3	1.2151024	0.4673423	41
3	0.8	1.4264555	0.5322225	41
4	14	-0.06650608	0.13423253	48
5	-15	-0.20536347	0.45237137	48
6	5.8	10.711373	0.656326	41
7	-19.8	-19.915014	0.115014	11
8	18.9	19.193245	0.293245	11
9	8.8	-0.7260513	0.48884018	48
10	-9.5	-9.652251	0.152251	11
11	8.1	-16.659458	0.337686	42
12	-8.8	16.571745	0.622709	42
13	-3.1	-2.7701359	0.3298641	11
14	-11.8	24.606413	0.646196	42
15	18.8	37.5234	0.0518329999999	41
16	7.7	15.392297	0.501787	41
17	-2.8	-3.2989988	0.4989988	11
18	-8.2	-16.575344	0.295021	41
19	14.1	0.31000805	0.29144169	48
20	5.8	11.520408	0.152709	41

Here's this table, which gives them the first 20 rows of the "test_mapping" table that holds the test and the mapping result. They can have columns like "X" and "Y" for the test point values, "Delta Y" for the test minus the ideal result and "Ideal func" with the ideal function number chosen. Through such a configuration, test points are analysed to identify whether or not they display perfect functions that contribute towards determining mapping quality and how functions are enhanced.

4. Conclusion

The project is successful by making a Python programme that decides which ideal functions to use by least squares and checks residuals to sort the test data into groups. The approach worked well and selected the right testing points, with all outcomes being kept in a SQLite database and shown with Bokeh. With this method, fitting various functions and organising them by class became much more convenient. They showed that using statistics and database management, along with a visual solution, keeps analysis simple and understandable.

References

Diana, A., Romano, E. and Irpino, A., 2025. Network Weighted Functional Regression: a method for modeling dependencies between functional data in a network. arXiv preprint arXiv:2501.18221.

Heng, Q., Zhou, H. and Lange, K., 2025. Tactics for Improving Least Squares Estimation. arXiv preprint arXiv:2501.02475.

Mo, Y., Liu, J. and Zhang, L., 2025. Deconvolution of spatial transcriptomics data via graph contrastive learning and partial least square regression. Briefings in Bioinformatics, 26(1), p.bbaf052.

Noma, H. and Gosho, M., 2025. Accurate inference methods based on the estimating equation theory for the modified Poisson and least-squares regressions. medRxiv, pp.2025-01.

Riani, M., Atkinson, A.C., Morelli, G. and Corbellini, A., 2025. The use of modern robust regression analysis with graphics: an example from marketing. Stats, 8(1).

Vicente-Gonzalez, L., Frutos-Bernal, E. and Vicente-Villardon, J.L., 2025. Partial Least Squares Regression for Binary Data. Mathematics, 13(3), p.458.

Vicente-Gonzalez, L., Frutos-Bernal, E. and Vicente-Villardon, J.L., 2025. Partial Least Squares Regression for Binary Data. Mathematics, 13(3), p.458.

Vicente-Gonzalez, L., Frutos-Bernal, E. and Vicente-Villardon, J.L., 2025. Partial Least Squares Regression for Binary Data. Mathematics, 13(3), p.458.