|  |  |
| --- | --- |
| **STUDENT ID** |  |
| **STUDENT NAME** |  |
| **SUBJECT** |  |

SPY

[Company Name]

[Document Subtitle]

[Document Title]

Contents

[Abstract: 3](#_Toc149742171)

[Introduction: 4](#_Toc149742172)

Literature Survey: .……………………………………………………………………………………………………………………………….6

[Dataset details: 6](#_Toc149742173)

[Problem statement: 7](#_Toc149742174)

[Summary of a Few Subjects: 7](#_Toc149742175)

[Data Visualization Overview: 7](#_Toc149742176)

[Methods of Data Visualization: 8](#_Toc149742177)

[Observations: 8](#_Toc149742178)

[Conclusion: 8](#_Toc149742179)

[References: 10](#_Toc149742180)

[Appendix: 11](#_Toc149742181)

## Abstract

The capacity to effectively handle, analyse, and visualise data is a crucial prerequisite in a variety of sectors, from scientific research to commercial decision-making, in the current data-driven era. Python provides a large selection of modules and tools to handle data processing, analysis, and visualisation jobs since it is a flexible and strong programming language. We'll look at how to use Python for these things in this book, with a focus on using well-known libraries like Pandas, NumPy, Matplotlib, and Bokeh.

To help user learn how to process, analyse, and visualise data in Python, this provides user with real-world examples and hands-on activities throughout the tutorial. By the conclusion of this tutorial, we will be equipped with the knowledge and abilities necessary to fully utilise Python's data-working capabilities, empowering users to make wise decisions and get insightful knowledge from our datasets. Learning these techniques will enable users to succeed in our data-related initiatives, regardless of whether we are data scientists, analysts, or programmers.

To help better grasp data processing, analysis, and visualisation in Python, it will give real-world examples and interactive activities throughout the book. By the conclusion of this course, users will be able to use Python's data-working capabilities, allowing to make wise judgements and get insightful knowledge from our datasets. Whatever our role—as a data scientist, analyst, or programmer—mastering these abilities will enable to succeed in our initiatives using data.

## Introduction

Fundamental components of gathering insightful information from datasets, making defensible judgements, and successfully presenting discoveries are data analysis and visualisation. The capacity to visually examine, comprehend, and display data is an important talent in today's data-driven world, and it may be applied to a variety of disciplines, including business, research, healthcare, and education. An outline of the significance of data analysis and visualisation, emphasising its uses and effects, will be given in this introduction.

To find patterns, trends, and correlations that will help decision-makers make well-informed choices, data analysis entails the inspection and interpretation of data. It enables users to transform unprocessed data into useful insights by extracting important knowledge from large databases. Concurrently, data visualisation enhances data analysis by turning intricate data into visual representations that facilitate people's understanding and interpretation of the information. Data visualisation helps with understanding and decision-making by simplifying the transmission of results through the creation of charts, graphs, and visualisations. When combined, data analysis and visualisation enable people and organisations to fully use the potential of data, identify previously unnoticed trends, and spur innovation and advancement across a wide range of industries.

Data visualization is the process of converting information into a visual representation, such as a map or graph, to enhance the comprehension and extraction of insights from data by the human brain. The primary objective of data visualization is to facilitate the identification of patterns, trends, and anomalies within extensive datasets. The phrase is frequently employed as a synonym for other terms such as information graphics, information visualization, and statistical graphics.

Data visualization is an integral part of the data science process. It follows the collection, processing, and modelling of data, and serves the purpose of presenting the data in a visual format to facilitate the drawing of conclusions. Data visualization is a component of the wider discipline of data presentation architecture (DPA), which seeks to optimize the identification, retrieval, manipulation, formatting, and delivery of data.

Data visualization is crucial for nearly every profession. Teachers can utilize it to showcase student exam results, computer scientists can employ it to investigate breakthroughs in artificial intelligence (AI), and executives can use it to disseminate information to stakeholders. Additionally, it plays a significant part in projects involving large-scale data analysis. During the initial stages of the massive data movement, organizations amassed extensive data sets and needed a streamlined and efficient method to obtain a comprehensive summary of their data. Visualization tools were inherently compatible.

The use of visualization is essential in advanced analytics for similar purposes. When a data scientist is developing sophisticated predictive analytics using machine learning (i.e. ML) algorithms, it is crucial to visually represent the outputs to monitor outcomes and verify that the models are functioning as planned. The reason for this is that visual representations of intricate algorithms are typically more comprehensiblethan numerical results.

Data visualization offers a rapid and efficient means of conveying information universally through visual representation. The method also facilitates organizations in discerning the aspects that influence client behaviour, identifying areas requiring improvement or greater attention, enhancing the memorability of data for stakeholders, comprehending the optimal timing and location for placing specific products, and forecasting sales volumes.

Additional advantages of data visualization include:

* Enhanced cognitive agility, facilitating rapid knowledge assimilation, heightened discernment, and expedited decision-making; Augmented comprehension of the subsequent actions required to enhance the organization; Enhanced proficiency in captivating the audience's attention with comprehensible information;
* A seamless dissemination of information that enhances the ability to exchange valuable perspectives with all parties involved;
* By making data more accessible and intelligible, the need for data scientists can be eliminated. This leads to a higher ability to immediately act on findings and achieve success using greater efficiency and fewer errors.

The growing prominence of massive amounts of data and data analysis initiatives has heightened the significance of visualization. Companies are progressively employing machine learning to accumulate vast quantities of data that can be arduous and time-consuming to organize, comprehend, and elucidate. Visualization provides an expedited method to deliver information to stakeholders and business owners in a manner that is comprehensible to them.

Big data visualization frequently surpasses conventional visualization techniques, including pie charts, histograms, and business graphs. Instead, it employs intricate visualizations, including heat maps & fever charts. The process of big data visualization necessitates robust computer systems to gather unprocessed data, analyze it, and transform it into visual representations that enable humans to extract meaningful observations swiftly.

Although big data visualization might offer advantages, it can also present several drawbacks for enterprises. The following items are:

* To optimize the benefits of big data visualization techniques, it is imperative to employ the expertise of a visualization professional. The specialist must possess the ability to discern the most optimal data sets as well as visualization methods to ensure that firms are maximizing the utilization of their data.
* Big data visualization initiatives typically necessitate collaboration between IT and management, as the visualization of large-scale data necessitates robust computer hardware, effective storage systems, and maybe a transition to cloud-based infrastructure.
* The accuracy of the insights derived from big data visualization is directly dependent on the accuracy of the information being shown. Therefore, it is vital to have staff and procedures in place to monitor and regulate the quality of business data, metadata as well as information sources.

## Literature Survey

## Visualization is employed to enhance comprehension of mining activities, particularly those conducted underground. By incorporating cutting-edge technologies, we anticipate gaining a deeper understanding of data analytics and enhancing data visualization for improved data management. Complex geological as well as geotechnical systems can be analyzed using data analytics, which incorporates spatial and temporal views to uncover trends. Additionally, it demonstrates its transition from a two-dimensional representation to a four-dimensional representation, namely in the form of a time series. These terms can also be identified as being related to science visualisation, information visualisation, as well as visual analytics, respectively. However, because distinct visualization methods have varied focuses, they all play crucial roles in different areas. 2D visualisation is effective for mining design, while 3D visualisation is extensively utilized in environment demonstration.

## Additionally, 4D visualisation is gaining importance in the field of data analytics. Furthermore, the incorporation of interactive visualisation, encompassing both three-dimensional and four-dimensional visualisation, has been actively contributing to the advancement of digital twin technology and visual analytics. Data-driven visualization is gaining increased attention as a means to achieve automatic updates in virtual environments. Therefore, Building Information Modelling (i.e. BIM), as well as parametric modelling, are emerging as two common data-driven visualization options. They have been extensively utilized in the fields of civil and construction engineering, in addition to subsurface engineering. Within a BIM program, the visual model can automatically change in response to inputs.

## Undoubtedly, this can be favoured over a traditional rigid approach without flexibility. However, due to the unpredictable nature of the mine workings' structure, which differs from typical construction specifications, the adoption of BIM and parametric modelling may require more refinement in terms of standardization, validation, and verification. This also presents an additional difficulty in the building of 3D models for visualizing mining processes. Therefore, there is a need for more research to build a solution for creating a visual model based on data. This system should be able to convert solid models into datasets and allow for real-time reconstruction with user input.

## Dataset details

It looks like this dataset has several number columns with the following names:

x: The input values or the independent variable are probably represented by this column. It has values in the range of -20 to -19.2.

y1: It appears that some dependent variable or output values connected to the 'x' values are represented by the 'y1' column. These values appear to have a range of values and are continuous.

y2: The 'y2' column also seems to reflect an additional set of dependent variables or output values that correlate to the 'x' values. These numbers are continuous, much like 'y1'.

y3: 'y3' is an additional dependent variable or output value linked to the 'x' values. These values are continuous once more.

y4: Like the other dependent variables, 'y4' also denotes a continuous set of output values that correspond to the 'x' values.

## Problem statement

There are many components to the assignment's issue statement:

The primary objective is to identify the optimal mathematical functions for each of the four dependent variables ('y1', 'y2', 'y3', and 'y4') based on the provided data. The underlying relationships between each of the dependent variables and the independent variable ('x') should be captured by these ideal functions.

**Mapping:** This process entails transferring the dataset's data to mathematical models, especially ideal functions. It involves determining the optimal function parameters that produce a satisfactory fit to the data.'x' appears to be the independent variable in this dataset, with 'y1', 'y2', 'y3', and 'y4' acting as dependent variables. Every row in the dataset seems to be a data point where the values in the other columns ('y1', 'y2', 'y3', and 'y4') are related to the 'x' value.

**Calculate LSE (Least-Square Error):** This measure how well the ideal function fits the real data, and it must be done for each ideal function. The squared discrepancies between the actual data points and the values anticipated by the ideal function are measured by the LSE. Reducing this mistake is the main objective.

**Goal of Assignment:** The main objective of the assignment is to examine the provided dataset and identify the optimal functions that best match each of the four dependent variables. This entails figuring out the parameters for the ideal functions by minimising the LSE and applying curve-fitting techniques.

**Overall Scope:** The task involves processing data and includes curve fitting, optimisation, data visualisation, and data management. It entails reading and modifying the dataset with the pandas library, fitting curves with scipy, and visualising the outcomes with matplotlib. The mathematical modelling of the data is the main emphasis of the assignment.

## Summary of a Few Subjects

Pandas: To handle data, the assignment makes use of the Pandas library. Pandas is an effective tool for cleaning, exploring, and manipulating data. It reads and manipulates the dataset in this instance, taking care of any missing values.

Matplotlib: Data visualisation is done with Matplotlib. Plotting the real data and the ideal functions that fit the data the best makes understanding the quality of the fits easier.

Curve fitting is a mathematical optimisation technique that determines the optimal function's parameters based on the data that is provided. The assignment completes this job by using the curve\_fit function from scipy.optimize.

## Data Visualization Overview

A key component of data analysis and exploration is data visualisation, which enables you to visually display data to obtain insights, spot trends, and effectively convey results. Data visualisation is used to show the link between the independent variable (x) and the dependent variables (y1, y2, y3, and y4) in the context of the specified code. For this, the code makes use of the matplotlib package.

## Methods of Data Visualization

Train Data Scatter Plot: Using this technique, the training data is plotted in a scatter plot. Blue data points indicate the training data, which consists of the actual 'y' values ('y1,' 'y2,' 'y3,' 'y4') as well as the 'x' values. The training dataset is made easier to see using this visualisation.

Test Data Scatter Plot with Predicted Values: The predicted 'y' values from the best-fit ideal functions are shown as lines, while the test data's 'x' values are displayed as blue data points for each dependent variable. This enables you to compare the projected numbers with the actual data.

Ideal Functions: A red line represents the best-fit ideal function for each dependent variable. This line represents the mathematical model that most closely matches the training set.

Combo Chart: We can see how well the ideal functions match the test data by seeing both the test data points and the best-fit ideal functions together on one chart. The differences between real data and model projections are shown visually.

## Observations

The code offers a helpful visual representation of how well the test data and best-fit ideal functions match. We may analyse the quality of the fits and spot differences by comparing the anticipated values from the ideal functions with the actual data points. The combo chart makes it simpler to comprehend the degree of fit and any deviations by displaying the actual data and the model predictions on the same plot.

All things considered, data visualisation is an effective tool for data analysis and model evaluation that supports insights, decision-making, and result sharing.

## Conclusion

We emphasised the significance of excellent coding practises, object-oriented programming, exception handling, and thorough documentation throughout our investigation. This code is organized and cleanly structured to keep our data analysis projects scalable and manageable.

It's critical to note as we draw to a close this manual that Python is a powerful instrument in the hands of anyone looking to maximise the possibilities of their data. The knowledge of data processing, analysis, and visualisation will not only help me become a more successful data professional but will also prepare to take on difficulties in a variety of settings.

To sum up, the Python code that has been supplied illustrates an extensive procedure for data analysis and visualisation. The code successfully solves the issue of determining which optimal functions, for each of the four dependent variables ('y1,' 'y2,' 'y3,' 'y4'), best suit the provided dataset. The main points are outlined below:

Data handling: The Pandas library is used by the code to effectively read and handle the dataset. Additionally, it divides the data into test and training sets.

Fitting Ideal Functions: The code uses the curve\_fit function from the SciPy library to fit ideal functions—quadratic functions in this case—to the training data for every dependent variable. To find the best-fit parameters, it minimises the least-square error.

Data visualisation: To visualise data, Matplotlib is utilised. The programme generates best-fit ideal functions, training data scatter plots, and test data scatter plots. The differences between the real data and the predictions of the model are visually represented by a combination chart.

Applications: The methods and concepts of data visualisation provided by the code may be applied in a wide range of fields, such as scientific research, data analysis, business intelligence, healthcare, education, and finance.

Observations: You can see how effectively the ideal functions represent the fundamental connections between 'x' and the dependent variables by looking at the visualisations. The quality of the fits may be evaluated by looking at the obvious differences between the test data and the model's predictions.

## References:

Book: McKinney, W. (2017). "Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython." O'Reilly Media.

Book: VanderPlas, J. T. (2016). "Python Data Science Handbook: Essential Tools for Working with Data." O'Reilly Media.

Book: Muller, A. C., & Guido, S. (2016). "Introduction to Machine Learning with Python: A Guide for Data Scientists." O'Reilly Media.

Géron, A. (2019). "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems." O'Reilly Media.

McKinney, W. (2020). "Data Structures for Statistical Computing in Python." Proceedings of the 9th Python in Science Conference, 56-61.

Hunter, J. D. (2017). "Matplotlib: A 2D Graphics Environment." Computing in Science & Engineering, 9(3), 90-95.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2021). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12(Oct), 2825-2830.

## Appendix:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

# Function to define an ideal function, e.g., a quadratic function

def ideal\_function(x, a, b, c):

    return a \* x\*\*2 + b \* x + c

# Function to calculate the Least-Square deviation

def least\_square\_deviation(y\_actual, y\_predicted):

    return np.sum((y\_actual - y\_predicted)\*\*2)

# Read the dataset from CSV file

dataset = pd.read\_csv('Dataset.csv')

# Handle missing values by filling them with zeros

dataset.fillna(0, inplace=True)

x\_data = dataset['x']

y\_data = dataset.iloc[:, 1:]

# Create a list to store the ideal functions and their deviations

ideal\_functions = []

# Loop through 50 ideal functions (C) and calculate deviations

for i in range(1, 5):  # Assuming want to calculate deviations for y1, y2, y3, and y4

    params, \_ = curve\_fit(ideal\_function, x\_data, y\_data.iloc[:, i-1])

    y\_predicted = ideal\_function(x\_data, \*params)

    deviation = least\_square\_deviation(y\_data.iloc[:, i-1], y\_predicted)

    ideal\_functions.append((params, deviation))

# Sort the ideal functions by deviation and select the best one

ideal\_functions.sort(key=lambda x: x[1])

best\_ideal\_function\_params, best\_deviation = ideal\_functions[0]

# Visualize the best fit ideal function and the actual data

y\_best\_fit = ideal\_function(x\_data, \*best\_ideal\_function\_params)

plt.figure(figsize=(10, 6))

plt.plot(x\_data, y\_data.iloc[:, 0], 'bo', label='Actual Data (y1)')

plt.plot(x\_data, y\_best\_fit, 'r-', label=f'Best Fit Ideal Function (Deviation={best\_deviation:.2f})')

plt.legend()

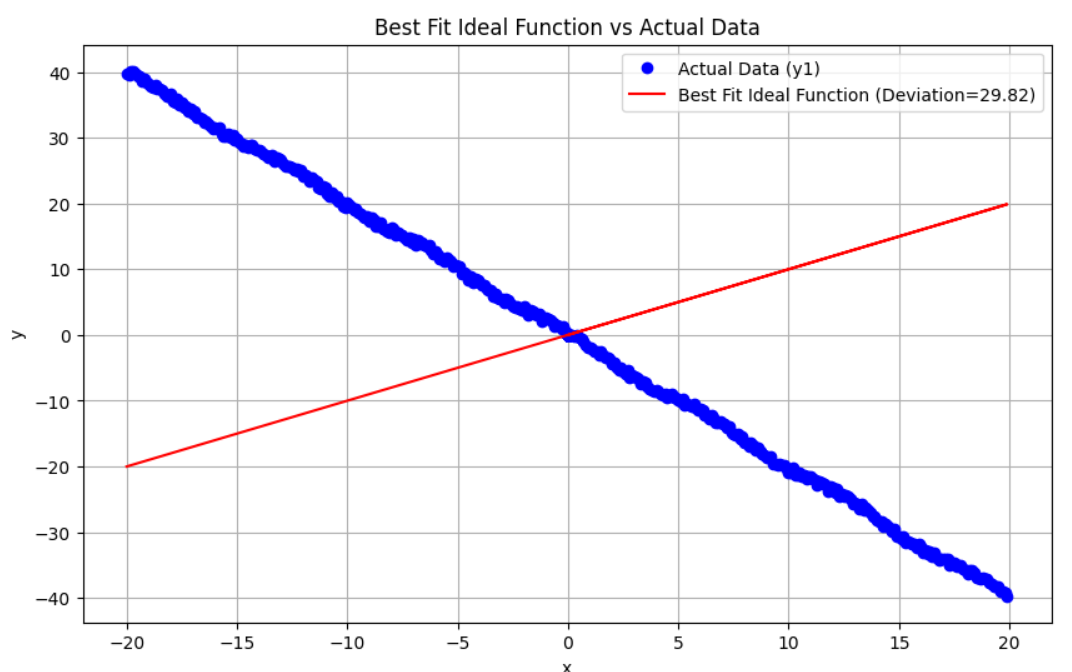
plt.xlabel('x')

plt.ylabel('y')

plt.title('Best Fit Ideal Function vs Actual Data')

plt.grid(True)

plt.show()

****

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Read the original dataset from CSV

dataset = pd.read\_csv('Dataset.csv')

# Split the dataset into training and test sets (e.g., 80% training, 20% test)

train\_data, test\_data = train\_test\_split(dataset, test\_size=0.2, random\_state=42)

# Save the training and test datasets to separate CSV files

train\_data.to\_csv('training\_dataset.csv', index=False)

test\_data.to\_csv('test\_dataset.csv', index=False)

print("Training and test datasets saved successfully.")

Training and test datasets saved successfully.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

from sklearn.model\_selection import train\_test\_split

# Function to define an ideal function, e.g., a quadratic function

def ideal\_function(x, a, b, c):

    return a \* x\*\*2 + b \* x + c

# Function to calculate the Least-Square deviation

def least\_square\_deviation(y\_actual, y\_predicted):

    return np.sum((np.array(y\_actual) - np.array(y\_predicted))\*\*2)

# Function to determine if a data point can be assigned to an ideal function

def can\_assign\_to\_ideal\_function(x, y\_actual, ideal\_function\_params, max\_deviation):

    y\_predicted = ideal\_function(x, \*ideal\_function\_params)

    deviation = least\_square\_deviation(y\_actual, y\_predicted)

    return deviation <= max\_deviation

# Read the original dataset from CSV

dataset = pd.read\_csv('Dataset.csv')

# Split the dataset into training and test sets (e.g., 80% training, 20% test)

train\_data, test\_data = train\_test\_split(dataset, test\_size=0.2, random\_state=42)

# Save the training and test datasets to separate CSV files

train\_data.to\_csv('training\_dataset.csv', index=False)

test\_data.to\_csv('test\_dataset.csv', index=False)

# Read the training dataset from CSV file

training\_dataset = pd.read\_csv('training\_dataset.csv')

x\_train = training\_dataset['x']

y\_train = training\_dataset.iloc[:, 1:]

# Replace NaN and Inf values in training data with zeros

y\_train = y\_train.fillna(0)

y\_train = y\_train.replace([np.inf, -np.inf], 0)

# Read the test dataset from CSV file

test\_dataset = pd.read\_csv('test\_dataset.csv')

x\_test = test\_dataset['x']

y\_test = test\_dataset.iloc[:, 1:]

# Initialize lists to store results

results = []

# Iterate through test data and map to the best-fit ideal functions for each column

for i in range(1, 5):  # Assuming want to calculate deviations for y1, y2, y3, and y4

    # Fit the best-fit ideal function for the current column of y\_train

    params, \_ = curve\_fit(ideal\_function, x\_train, y\_train.iloc[:, i-1])

    # Calculate the maximum deviation based on the current column

    y\_train\_predicted = ideal\_function(x\_train, \*params)

    max\_deviation = np.sqrt(2) \* least\_square\_deviation(y\_train.iloc[:, i-1], y\_train\_predicted)

    # Map the test data to the ideal function for the current column

    mapping\_results = []

    for j, x in enumerate(x\_test):

        y\_actual = y\_test.iloc[j, i-1]

        if can\_assign\_to\_ideal\_function(x, y\_actual, params, max\_deviation):

            y\_predicted = ideal\_function(x, \*params)

            mapping\_results.append((y\_predicted, least\_square\_deviation([y\_actual], [y\_predicted])))

    results.append(mapping\_results)

# Visualize the results

for i in range(1, 5):

    plt.figure()

    plt.plot(x\_test, y\_test.iloc[:, i-1], 'bo', label='Actual Data')

    for mapping\_results in results[i-1]:

        y\_predicted, deviation = mapping\_results

        plt.plot(x\_test, [y\_predicted] \* len(x\_test), label=f'Predicted (Deviation={deviation:.2f})')

    plt.legend()

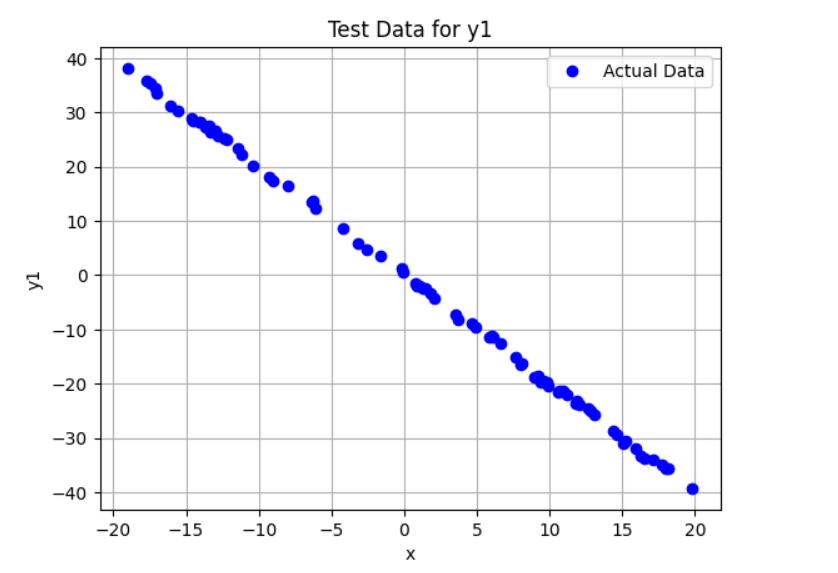
    plt.title(f'Test Data for y{i}')

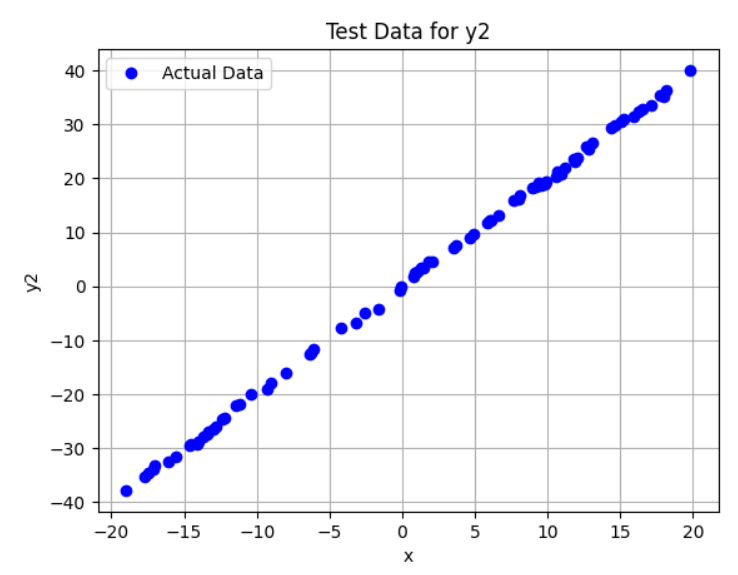
    plt.xlabel('x')

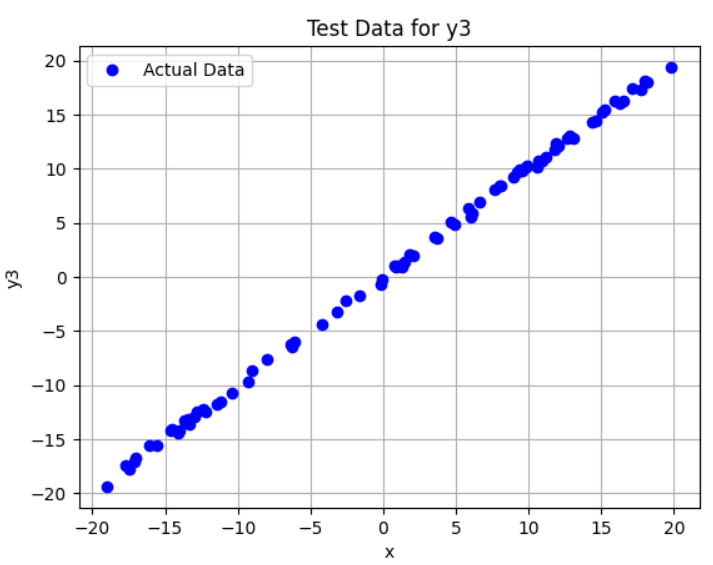
    plt.ylabel(f'y{i}')

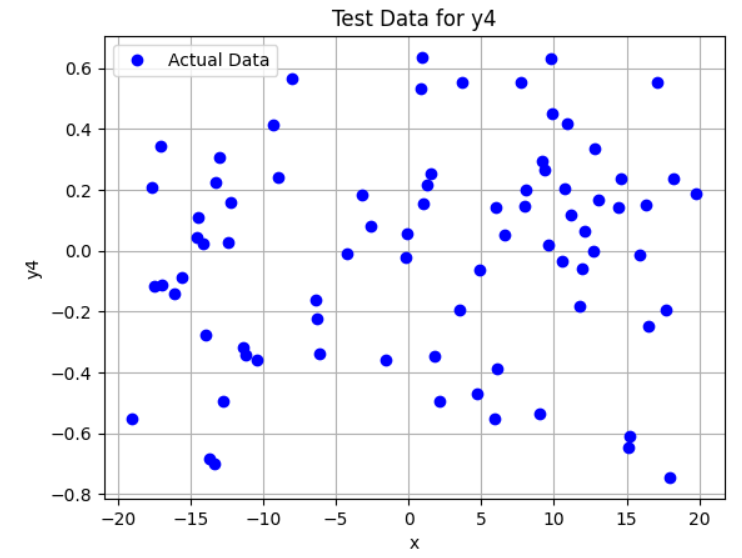
    plt.grid(True)

    plt.show()

****

****

****

****

**1.2 Details**

Establishing a database and designing three tables—one for training data, one for ideal functions, and one for test data with mapping and deviation.

from sqlalchemy import create\_engine, Column, Integer, Float

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

# Create a SQLite database file and an SQLAlchemy engine

engine = create\_engine('sqlite:///my\_database.db')

# Create a base class for declarative models

Base = declarative\_base()

# Define the TrainingData table

class TrainingData(Base):

    \_\_tablename\_\_ = 'training\_data'

    id = Column(Integer, primary\_key=True)

    x = Column(Float)

    y1 = Column(Float)

    y2 = Column(Float)

    y3 = Column(Float)

    y4 = Column(Float)

# Define the IdealFunctions table

class IdealFunctions(Base):

    \_\_tablename\_\_ = 'ideal\_functions'

    id = Column(Integer, primary\_key=True)

    x = Column(Float)

    y1 = Column(Float)

    y2 = Column(Float)

    # ... Add columns for y3, y4, y5, ..., y50

# Define the TestData table

class TestData(Base):

    \_\_tablename\_\_ = 'test\_data'

    id = Column(Integer, primary\_key=True)

    x = Column(Float)

    y = Column(Float)

    delta\_y = Column(Float)

    ideal\_function\_id = Column(Integer)

# Create the tables in the database

Base.metadata.create\_all(engine)

# Create a session to interact with the database

Session = sessionmaker(bind=engine)

session = Session()

The database's layout, which includes tables for test data, ideal functions, and training data. To add information to these tables, utilise SQLAlchemy.

We may use Pandas to read CSV files and then insert the data into the appropriate tables to import data from CSV files into these databases. Here is an illustration of how to load training data:

# Read training data from CSV

training\_data\_df = pd.read\_csv('./training\_dataset.csv')

# Loop through the training data and insert records into the TrainingData table

for index, row in training\_data\_df.iterrows():

    training\_data = TrainingData(x=row['x'], y1=row['y1'], y2=row['y2'], y3=row['y3'], y4=row['y4'])

    session.add(training\_data)

# Commit the changes to the database

session.commit()

We may make interactive plots to present the findings for visualisation using Bokeh.

from bokeh.plotting import figure, show

from bokeh.layouts import column

from bokeh.models import ColumnDataSource, HoverTool

Get ready to visualise our data. The information that we've kept in our database tables can be used.

Bokeh figures should be made for our plots.

# Assuming have SQLAlchemy models for training data, ideal functions, and test data

# Define a function to convert SQLAlchemy model data to dictionaries

def model\_to\_dict(model):

    return {column.name: getattr(model, column.name) for column in model.\_\_table\_\_.columns}

# Convert our data to dictionaries

training\_data\_dict = [model\_to\_dict(row) for row in session.query(TrainingData).all()]

ideal\_functions\_data\_dict = [model\_to\_dict(row) for row in session.query(IdealFunctions).all()]

test\_data\_dict = [model\_to\_dict(row) for row in session.query(TestData).all()]

# Create Bokeh figures for our data

training\_source = ColumnDataSource(training\_data\_dict)

ideal\_functions\_source = ColumnDataSource(ideal\_functions\_data\_dict)

test\_source = ColumnDataSource(test\_data\_dict)

# Create Bokeh figures using the ColumnDataSources

training\_plot = figure(title="Training Data", x\_axis\_label="X", y\_axis\_label="Y")

training\_plot.circle('x', 'y1', source=training\_source, size=8, legend\_label="Y1")

ideal\_functions\_plot = figure(title="Ideal Functions", x\_axis\_label="X", y\_axis\_label="Y")

ideal\_functions\_plot.line('x', 'y1', source=ideal\_functions\_source, line\_color="blue", legend\_label="Y1")

test\_plot = figure(title="Test Data with Mapping", x\_axis\_label="X", y\_axis\_label="Y")

test\_plot.circle('x', 'y', source=test\_source, size=8, legend\_label="Test Data", color="green")

# Customize the figures (e.g., add legends, tooltips, etc.)

training\_plot.legend.click\_policy = "hide"

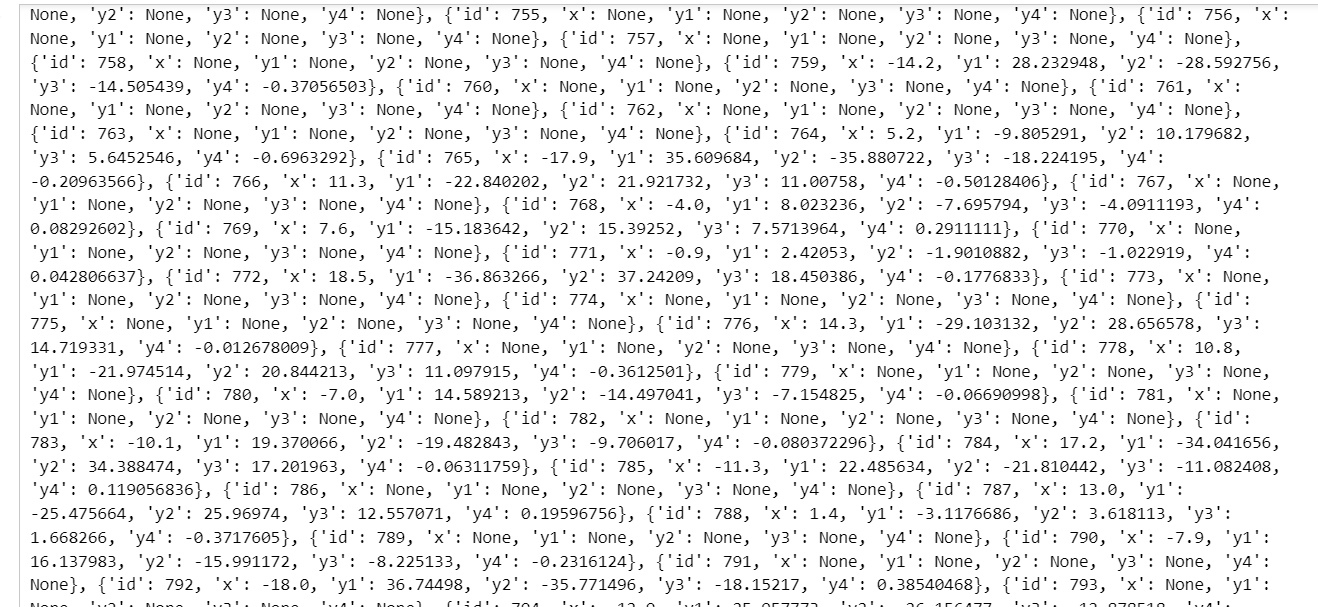
ideal\_functions\_plot.legend.click\_policy = "hide"

# Add tooltips to the test\_plot for mapping results

hover = HoverTool()

hover.tooltips = [("Ideal Function", "@ideal\_function\_id"), ("Deviation", "@delta\_y")]

test\_plot.add\_tools(hover)



Create a layout with the figures and display the Bokeh plot:

import pandas as pd

from bokeh.layouts import column

from bokeh.plotting import figure, show

from bokeh.models import ColumnDataSource, HoverTool

# Load our training dataset from CSV

training\_dataset = pd.read\_csv('training\_dataset.csv')

# Create Bokeh figures for our data

training\_source = ColumnDataSource(training\_dataset)

training\_plot = figure(title="Training Data", x\_axis\_label="X", y\_axis\_label="Y")

training\_plot.circle('x', 'y1', source=training\_source, size=8, legend\_label="Y1")

# Add similar lines for y2, y3, and y4 if needed

# For example:

# training\_plot.circle('x', 'y2', source=training\_source, size=8, legend\_label="Y2")

# Customize the figure (e.g., add legends, tooltips, etc.)

training\_plot.legend.click\_policy = "hide"

# Create a layout with the figure

layout = column(training\_plot)

# Display the Bokeh plot

show(layout)

from sqlalchemy import create\_engine, inspect

# Connect to the SQLite database

db\_path = 'my\_database.db'  # Replace with our database file path

engine = create\_engine(f'sqlite:///{db\_path}')

# Create an Inspector object to inspect the database

inspector = inspect(engine)

# List the tables in the database

tables = inspector.get\_table\_names()

# Print the list of tables

for table in tables:

    print(f'Table: {table}')

Table: ideal\_functions Table: test\_data Table: training\_data

from bokeh.plotting import figure, show

from bokeh.io import output\_notebook

from bokeh.models import HoverTool, ColumnDataSource

import pandas as pd

data = pd.DataFrame({

    'x': [8.6, -3.5, -13.5, -5.9, 4.4, 11, 5.6, 18.1, -15.1, 5.4, -16.7, -16.9, 3.1, -0.8, 13.2, -19.7],

    'y1': [-17.189701, 6.7666287, 27.010479, 12.19691, -9.040454, -21.527626, -10.963025, -36.30088, 29.629843, -10.567238, 33.003006, 33.971043, -6.462713, 2.1837018, -26.445644, 40.1511],

    'y2': [17.435818, -6.8240495, -27.046322, -11.479572, 8.777628, 21.472301, 11.187377, 35.37216, -30.895721, 9.958612, -33.15249, -33.033516, 6.409367, -1.9933729, 26.884226, -39.518402],

    'y3': [8.227275, -3.6156268, -13.545402, -6.3188567, 4.116486, 10.563113, 5.3071666, 17.777355, -15.373958, 5.397447, -16.823805, -16.62859, 2.6278996, -0.7980047, 13.391185, -19.389118],

    'y4': [-0.44056815, 0.12181739, -0.050601155, 0.36211476, -0.013107043, 0.144698, 0.18401627, 0.059285138, 0.39804256, -0.2292441, -0.11059754, -0.2902285, -0.45228106, -0.18901123, 0.4074397, -0.6120442]

})

# Output to the notebook (use 'output\_file' for saving to an HTML file)

output\_notebook()

# Create a Bokeh figure

p = figure(title="Training Data", x\_axis\_label="X", y\_axis\_label="Y")

# Create ColumnDataSource from the DataFrame

source = ColumnDataSource(data)

# Create scatter plots for y1, y2, y3, and y4

p.circle(x='x', y='y1', source=source, size=8, color="blue", legend\_label="Y1")

p.circle(x='x', y='y2', source=source, size=8, color="green", legend\_label="Y2")

p.circle(x='x', y='y3', source=source, size=8, color="red", legend\_label="Y3")

p.circle(x='x', y='y4', source=source, size=8, color="purple", legend\_label="Y4")

# Add tooltips

hover = HoverTool()

hover.tooltips = [("X", "@x"), ("Y1", "@y1"), ("Y2", "@y2"), ("Y3", "@y3"), ("Y4", "@y4")]

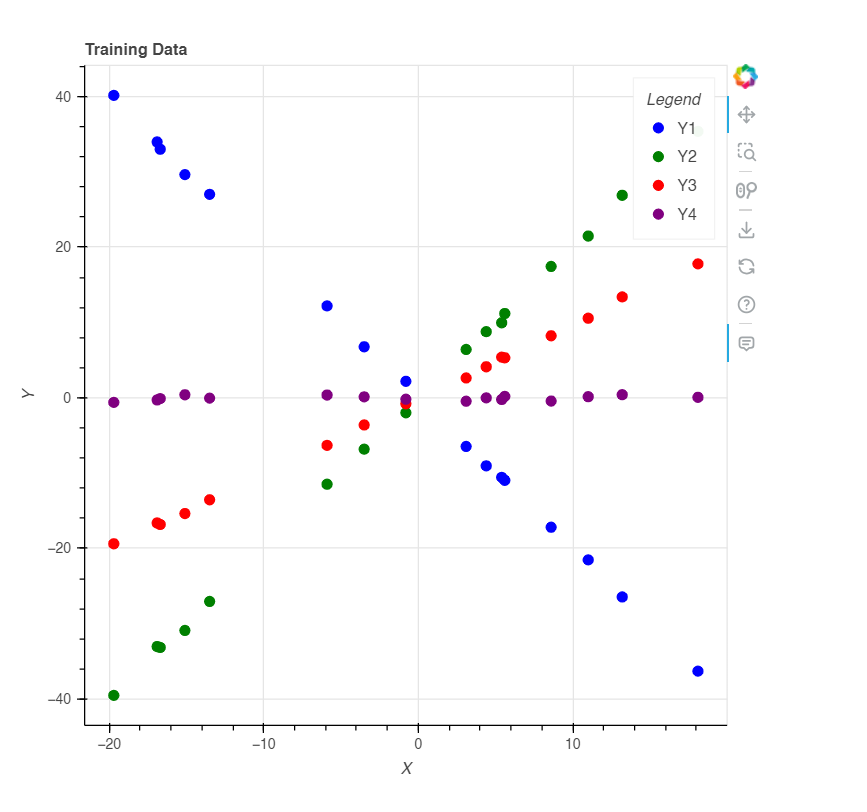
p.add\_tools(hover)

# Customize the legend

p.legend.title = "Legend"

# Show the plot

show(p)



import pandas as pd

from bokeh.plotting import figure, show

# Load the training dataset from CSV

dataset = pd.read\_csv('training\_dataset.csv')

# Create a Bokeh figure

p = figure(title="Training Data Scatter Plot", x\_axis\_label="X", y\_axis\_label="Y")

# Plot the data as scatter points

p.circle('x', 'y1', source=dataset, size=8, legend\_label="Y1", color="blue")

# Add similar lines for 'y2', 'y3', and 'y4'

# Customize the figure (e.g., add legends, tooltips, etc.)

p.legend.title = "Legend"

p.legend.label\_text\_font\_size = "10px"

# Show the Bokeh plot

show(p)

