Programming with Python (DLMDSPWP01)

A Written Assignment on

<u>The Four Ideal Functions That Are The Best Fit</u>

Author: Samyak Anand

Matriculation Number: 321149613

Tutor's Name: Cosmina Croitoru

Date: 31 Jan 2024

Place: Berlin, Germany

Introduction

The Python software advanced for this assignment effectively implements the specified requirements. In this part, there are a visualization in best fit function from all the functions in python workspace. The item-oriented design guarantees modularity and maintainability, while the use of general and person-defined exception coping enhances the program's robustness.

This assignment now not only fulfills the desired standards but additionally showcases talent in utilizing outstanding Python libraries for facts processing and visualization. The adherence to first-class practices in coding, documentation, and the use of established libraries ensures a reliable and efficient approach to the given challenge.

The given task in this programming with Python subject is to achieve and carry out data storage and retrieval, exploratory data analysis and model selection, data collection, data cleansing, data visualization, data mapping and deviation calculation, unit testing, and so on. To complete this list of tasks, three datasets are provided: train, ideal, and test sets. Correlating the given task to a real-world scenario in which a company manufactures electronic devices As a data scientist, my job is to analyze the performance of various components used in their devices and track that data against the optimal reading to predict if a piece of equipment is about to fail (Washam, 2022). Regarding that, we have the data sets from various experiments and tests conducted on these components, resulting in four training datasets (A) and one test dataset (B). Additionally, we have access to datasets for 50 ideal functions (C), which represent the expected behavior of the components.

Problem Statement

The task is to create a Python program that analyses the performance of electronic components by Using training data, to select the four best-fitting functions from the fifty available. Furthermore, the program must use the provided test data to decide whether each x-y pair of values can be assigned to one of the four ideal functions. If this is the case, the program must additionally execute the mapping and save it alongside the deviation at hand.

SQLite was chosen as the database system for the project since it is one of the most popular and easy-to-use relational database systems. It has numerous advantages over other relational databases. SQLite is an embedded, server-less relational database management system. It is an open-source in-memory library that requires no configuration or installation (What Is SQLite, 2021).

The following are the SQLite database properties that are desirable for the current project (What Is SQLite, 2021):

- SQLite is a free and open-source database management system. After installation, there is no need for a license.
- SQLite is serverless because it does not require a separate server process or system to function.
- SQLite allows one to operate on several databases at the same time in the same session, making it versatile.
- SQLite allows one to operate on several databases at the same time in the same session, making it versatile.
- SQLite requires no configuration. It does not require any setup or administration.

The SQLAlchemy Python package is used to access the stored dataset from the SQLite database. SQLAlchemy is a Python package that allows programs to communicate with databases. This package is typically used as an Object Relational Mapper (ORM) tool, converting Python classes to tabs in relational databases and automatically converting function calls to SQL statements (SQLAlchemy ORM Tutorial for Python Developers, n.d.).

The following are the benefits of SQLAlchemy that are desirable for the current project:

- Cross-platform compatibility: SQLAlchemy supports a variety of database management systems, including SQLite, allowing us to build code that works on numerous platforms (Shafqat, n.d.).
- Pythonic API: SQLAlchemy includes a Pythonic API that makes working with databases in Python simple (Python, n.d.).
- SQL Expression Language: SQLAlchemy includes a SQL Expression Language that allows SQL queries to be written in Python code, making it easier to develop dynamic queries and connect with existing Python programs (SQLAlchemy Core “Expression Language, n.d.)

Database scheme

Below are the structure tables to be created in the SQLite database:

Table 1: The training dataset database table.

X	Y1 (training func)	Y2 (training func)	Y3 (training func)	Y4 (training func)
x1	y11	y21	y31	y41
…	…	…	…	…
xn	y1n	y2n	y3n	y4n

Table 2: The ideal dataset database table.

X	Y1 (ideal func)	Y2 (ideal func)	…	Ym (ideal func)	…	Y50 (ideal func)
x1	y11	y21	…	ym1	…	y50_1
…	…	…	…	…	…	…
xn	y1n	y2n	…	ymn	…	y50_n

Table 3: The database table of the test-data, with mapping and y-deviation

X (test func)	Y (test func)	Delta Y (test func)	No. of ideal func
x1	y11	y21	n1
…	…	…	…
xn	y1n	y2n	y3n

Data collection

We will continue with the project by analyzing the provided data directly, without the requirement for extra data collection for exploratory data analysis. The study will include the use of several statistical approaches to summarize the data and reveal insights. This approach seeks to increase data comprehension and enable informed decision-making based on the findings.

Data cleansing

The process of finding and resolving corrupt, erroneous, or irrelevant data is known as data cleansing. This crucial stage of data processing, also known as data scrubbing or data cleaning, improves the consistency, dependability, and usefulness of an organization’s data. Missing values, misplaced entries, and typographical errors are examples of common data flaws. In certain cases, data cleansing requires specific values to be filled in or corrected, while in others, the values must be eliminated (*What Is Data Cleansing?* n.d.).

Missing Values and Outliers: Some data records may contain values that were not seen. Missing data occurs when no value is available in one or more of an individual's variables (Gawali, 2021). An outlier is a data point or set of data points that differ from the rest of the dataset's values. That is, it is a data point or point that appear apart from the main distribution of data values in a dataset (Mulani, 2020). To resolve missing numbers and outliers, several methods are frequently used:

a) Removal of the data records with missing values and/or outliers.
b) Replacing the missing or outlier value with an interpolated value from nearby records or the average value of its variable Replacement of the missing value or outlier with the most frequently observed value for that variable across all data records.

Duplicate Records: If the dataset contains duplicate records, they are eliminated before the data analysis begins. The removal of duplicate values from the data set is critical to the cleansing process. Duplicate data consumes unneeded storage space and significantly delays calculations. In the worst-case scenario, duplicate data can skew analysis results and threaten the data set's integrity (*Removing Duplicated Data in Pandas*, n.d.).

Redundancy: Other problems that may arise in the dataset are caused by the presence of redundant and unnecessary variables. When various databases are integrated, additional redundant data is

Data Pre-Processing

Data preparation is required before use. The concept of data preprocessing is the conversion of raw data into a clean data set. Before running the method, the dataset is preprocessed to check for missing values, noisy data, and other irregularities (P. Singh et al., 2021). Real-world, or raw, data often contains irregular formatting, and human errors, and is incomplete. Data preparation resolves such difficulties and makes datasets completer and more efficient for data analysis (Joby, n.d.). As a result, the Pandas package's built-in functions such as .info (), shape(), and .describe() are used for defined data frames to comprehend the data type, number of rows and columns, and data distribution.

Ideal data:

Based on the code output, we know that it comprises an independent variable 'x' and 50 dependent variables 'y1', 'y2', 'y3... and y50, each having 400 data points, and that all entries in the specified datasets are float64 data types

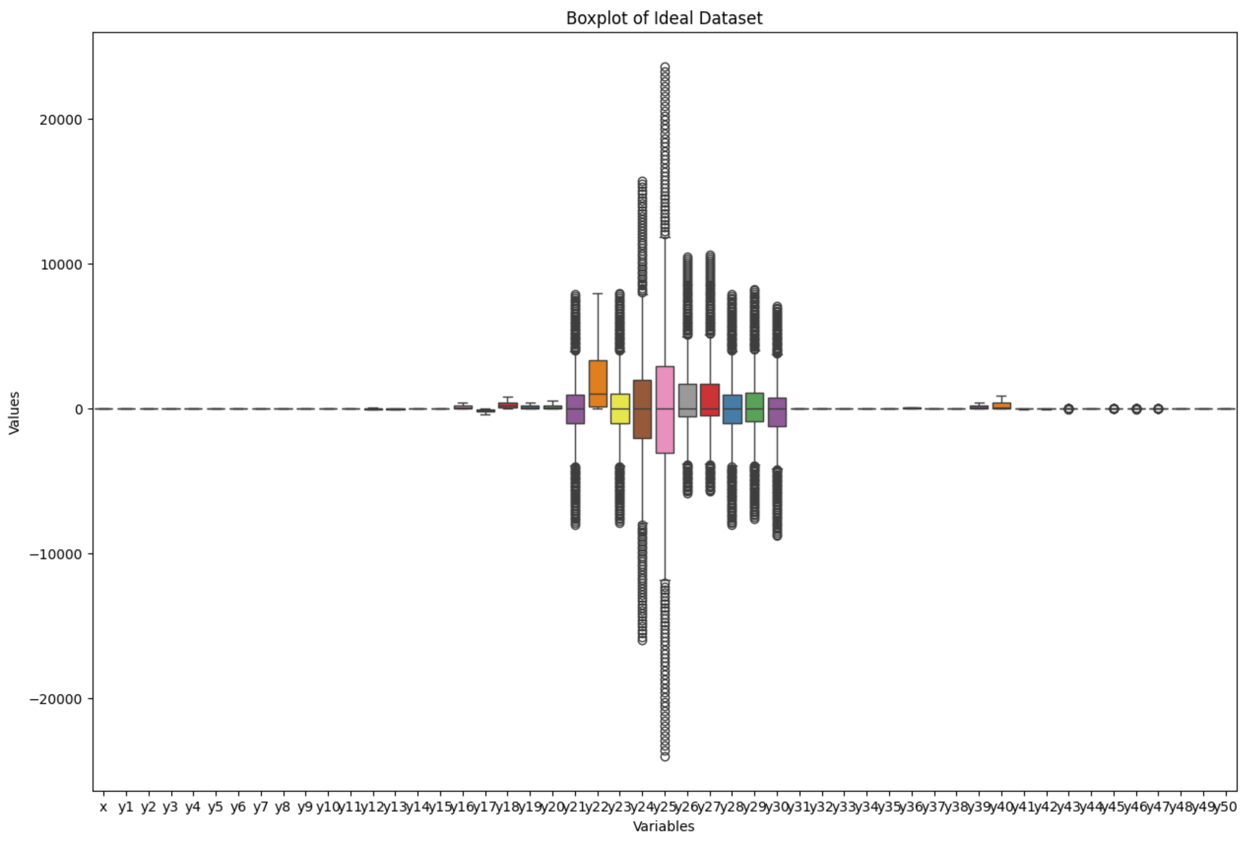


Figure 1: Boxplot of Ideal dataset

Train data

Based on the code output, we know that it comprises an independent variable 'x' and four dependent variables 'y1', 'y2', 'y3, and y4, each with 400 data points, and that all entries in the specified datasets are float64 data types. The dependent and independent variables' data are described as follows.

Table 4: train_data.describe().T

	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
x	400	-0.05	11.5614	-20	-10.025	-0.05	9.925	19.9
y1	400	-5.0782	23.0923	-45.292	-24.886	-5.2237	15.2067	34.639
y2	400	-39.991	6055.12	-16000	-2015.3	-0.0249	1955.08	15761.7
y3	400	50.003	28.9194	-0.3076	25.0294	49.9299	75.1918	99.8957
y4	400	234.285	261.214	0.28375	25.5605	100.279	402.034	899.828

There are no duplicate values in the training dataset, so there is no need to handle or eliminate them. Because there are no null values in the training dataset, there is no need to handle them. Outliers cannot be removed or handled for this project since all values in the 'y' columns are necessary for the process's subsequent phases.

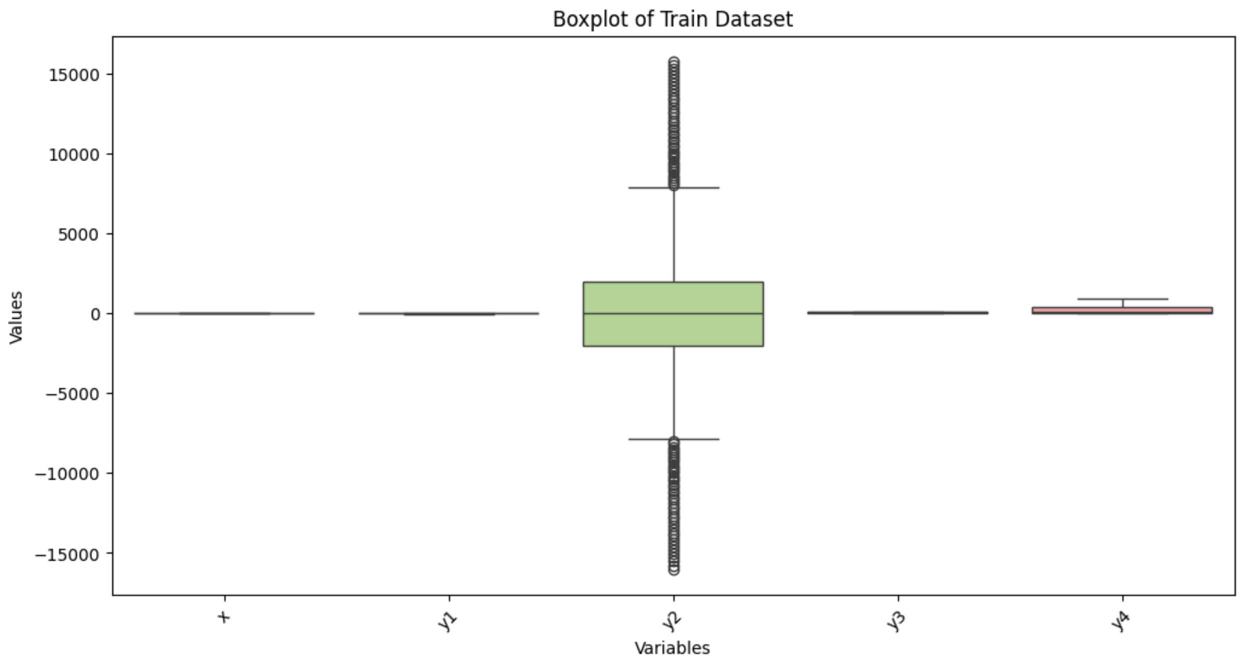


Figure 2: Boxplot for train dataset

Based on the above-mentioned box plot, we can draw the following conclusions: The box plots for y1, y2, y3, and y4 do not show points beyond the whiskers, indicating that their minimum and maxi- mum values are within the whiskers.

Test data

Based on the code output, we know that it consists of an independent variable 'x' and a dependent variable 'y', each with 100 data points, and that all entries in the specified datasets are float64 data types. The dependent and independent variables' data are described as follows:

```
test_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
```

```
---  ------  --------------  -----
 0   x       100 non-null    float64
 1   y       100 non-null    float64
dtypes: float64(2)
memory usage: 1.7 KB
```

```
test_data.describe().T
```

Table 5: test dataset before cleaning

count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	
x	100	-0.09	11.0908	-19.5	-9.825	0	9.1	19.4
y	100	-875.58	4774.3	-15681	-33.718	21.4284	86.9767	13840.7

We observed 16 duplicate values in the 'x' variable of the test function, and we must treat them correctly to protect the quality and integrity of our data. A frequent strategy in data pre-processing and cleaning is to use mean value imputation to remove duplicates from a dataset (*Introduction to Data Imputation | Simplilearn*, 2022). As a result, the 'y' row is replaced with the mean value of its corresponding 'x' row. Finally, we will remove the duplicate 'x' rows. We chose to preserve a unique 'x' value for each 'y' value in this work because it would help us associate the optimal ideal functions with the test points later in the mapping process.

Table 6: test dataset after removing duplicate

	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**
x	84	-0.0536	11.0875	-19.5	-9.825	0.55	9.275	19.4
y	84	-852.71	4589.63	-15681	-36.948	29.6612	117.978	13840.7

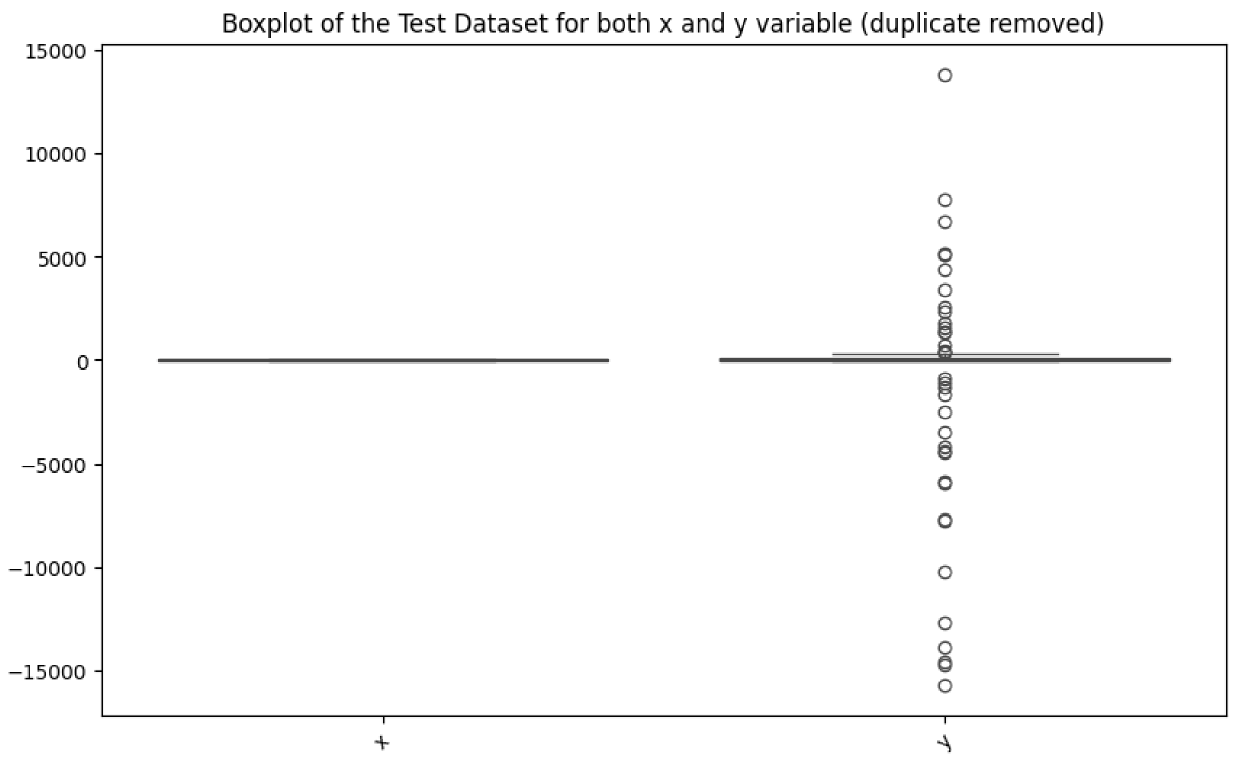


Figure 3: Boxplot of the dataset for both x and y variables (duplicate removed)

Database

For our model, we have created three different table in database to store the Ideal dataset, train dataset, and test dataset. To store the data, using sqlalchemy database we need to create a database engine.

```
# Create SQLite engine
engine = create_engine('sqlite:///assignment_database.db')
```

Once the engine is created, we need to define the table names are ideal_data, train_data and test_data.

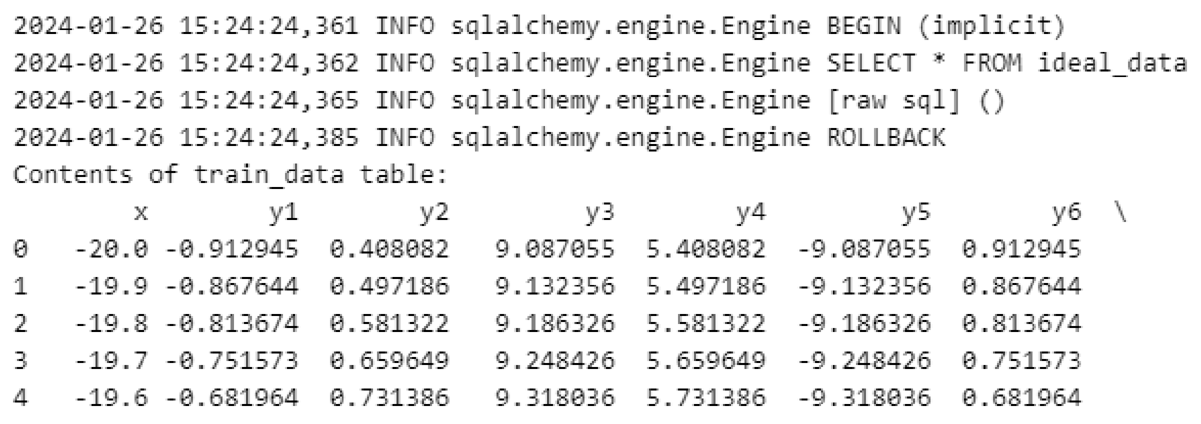
```
# Save datasets to SQLite database
test_data.to_sql('test_data', con=engine, index=False, if_exists='replace')
ideal_data.to_sql('ideal_data', con=engine, index=False, if_exists='replace')
train_data.to_sql('train_data', con=engine, index=False, if_exists='replace')
```

Once the data is inserted into tables, we can display the inserted data by running the statement and calling the function using the sqlalchemy library. We define a function:

```
        def ideal_data_table()
```

which is used to display the contents of the ideal dataset.

```
query = "SELECT * FROM ideal_data"
ideal_data = pd.read_sql_query(query, engine)
```



```
2024-01-26 15:24:24,361 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-01-26 15:24:24,362 INFO sqlalchemy.engine.Engine SELECT * FROM ideal_data
2024-01-26 15:24:24,365 INFO sqlalchemy.engine.Engine [raw sql] ()
2024-01-26 15:24:24,385 INFO sqlalchemy.engine.Engine ROLLBACK
Contents of train_data table:
       x        y1        y2         y3        y4         y5        y6  \
0  -20.0 -0.912945  0.408082  9.087055  5.408082  -9.087055  0.912945
1  -19.9 -0.867644  0.497186  9.132356  5.497186  -9.132356  0.867644
2  -19.8 -0.813674  0.581322  9.186326  5.581322  -9.186326  0.813674
3  -19.7 -0.751573  0.659649  9.248426  5.659649  -9.248426  0.751573
4  -19.6 -0.681964  0.731386  9.318036  5.731386  -9.318036  0.681964
```

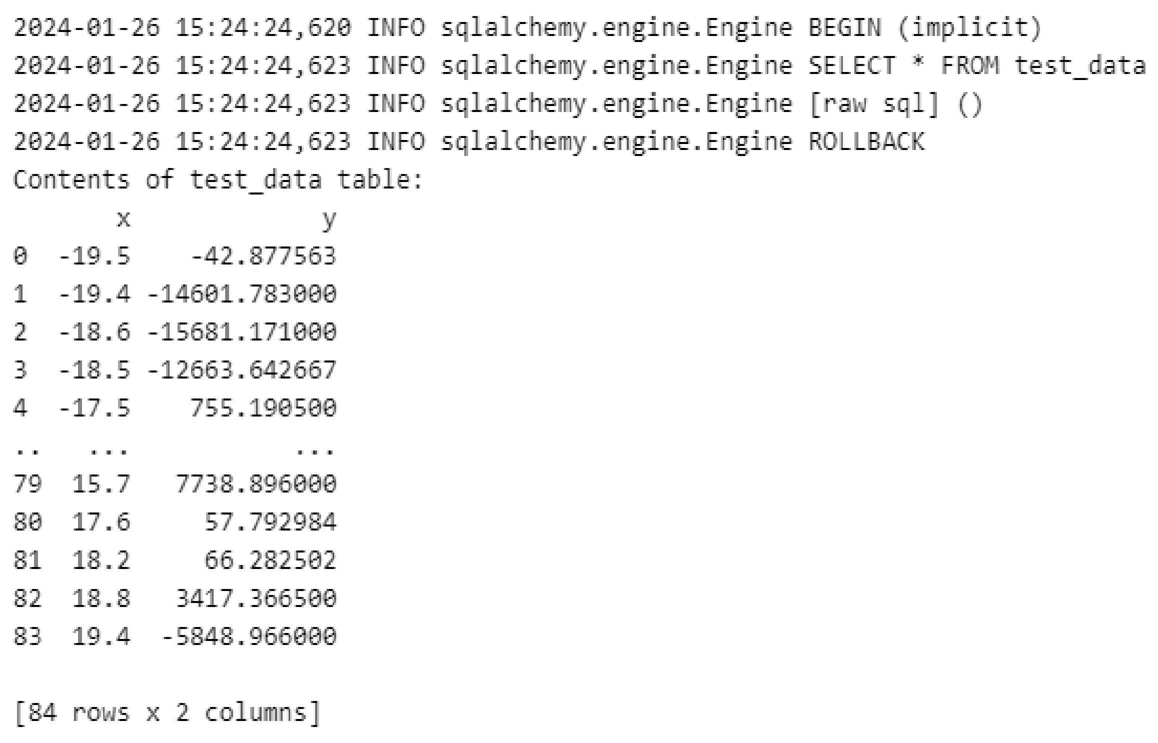
Figure 4: Displaying the content of ideal_data table, present in database

Similalry, for test_data we define a function:

```
        def test_data_table()
```

which is used to display the content of the test dataset. We can see that; the test dataset is clean all the duplicate values are missing and there are 82 unique data available.

```
        query = "SELECT * FROM test_data"
        test_data = pd.read_sql_query(query, engine)
```

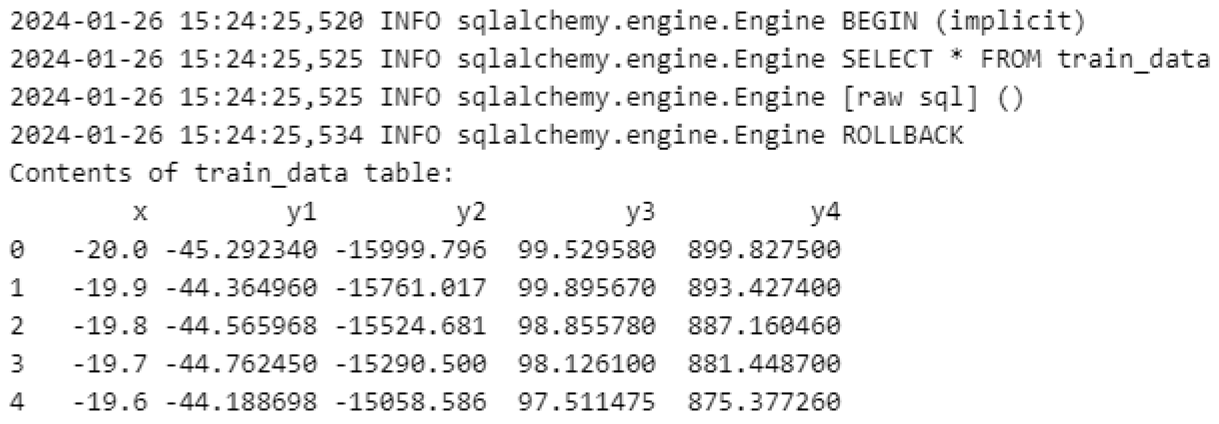


```
2024-01-26 15:24:24,620 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-01-26 15:24:24,623 INFO sqlalchemy.engine.Engine SELECT * FROM test_data
2024-01-26 15:24:24,623 INFO sqlalchemy.engine.Engine [raw sql] ()
2024-01-26 15:24:24,623 INFO sqlalchemy.engine.Engine ROLLBACK
Contents of test_data table:
       x             y
0  -19.5    -42.877563
1  -19.4 -14601.783000
2  -18.6 -15681.171000
3  -18.5 -12663.642667
4  -17.5    755.190500
..   ...           ...
79  15.7   7738.896000
80  17.6     57.792984
81  18.2     66.282502
82  18.8   3417.366500
83  19.4  -5848.966000

[84 rows x 2 columns]
```

Figure 5:Displaying the content of test_data table, present in database.

For train_data we define function
def train_data_table()
which display the content of train dataset.

```
query = "SELECT * FROM train_data"
train_data = pd.read_sql_query(query, engine)
```



```
2024-01-26 15:24:25,520 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2024-01-26 15:24:25,525 INFO sqlalchemy.engine.Engine SELECT * FROM train_data
2024-01-26 15:24:25,525 INFO sqlalchemy.engine.Engine [raw sql] ()
2024-01-26 15:24:25,534 INFO sqlalchemy.engine.Engine ROLLBACK
Contents of train_data table:
       x         y1         y2         y3          y4
0   -20.0 -45.292340 -15999.796  99.529580  899.827500
1   -19.9 -44.364960 -15761.017  99.895670  893.427400
2   -19.8 -44.565968 -15524.681  98.855780  887.160460
3   -19.7 -44.762450 -15290.500  98.126100  881.448700
4   -19.6 -44.188698 -15058.586  97.511475  875.377260
```

Figure 6: Displaying the content of train_data table, present in database.

Mapping Test Dataset with Ideal Function

Linear regression is a statistical modelling technique that is used to assess the relationships between one or more independent variables and a dependent variable. The most common form of regression analysis is linear regression, which involves determining the line that best fits the data based on a mathematical criterion. This procedure is designed to determine how changes in the independent variables affect changes in the dependent variable (“Regression Analysis,” 2023).

$$y = \beta_0 + \beta_1 x$$

Where,
‘y’ is the dependent variable,
‘x’ is the independent variable,
'β_0' is the intercept, and
'β_1' is the slope

Using this method to find predicted values, computing the absolute differences between test points and predicted values, and comparing the resulting differences to the maximum deviation value for the corresponding ideal function The maximum deviation value was determined by calculating the difference between the absolute values of the training and its respective ideal function for all rows of y values and identifying the row with the highest value, which was previously multiplied by the square root of 2. Furthermore, the bar graph below is produced to illustrate the findings up to the bar heights of the greatest deviation value, which is shown in red.

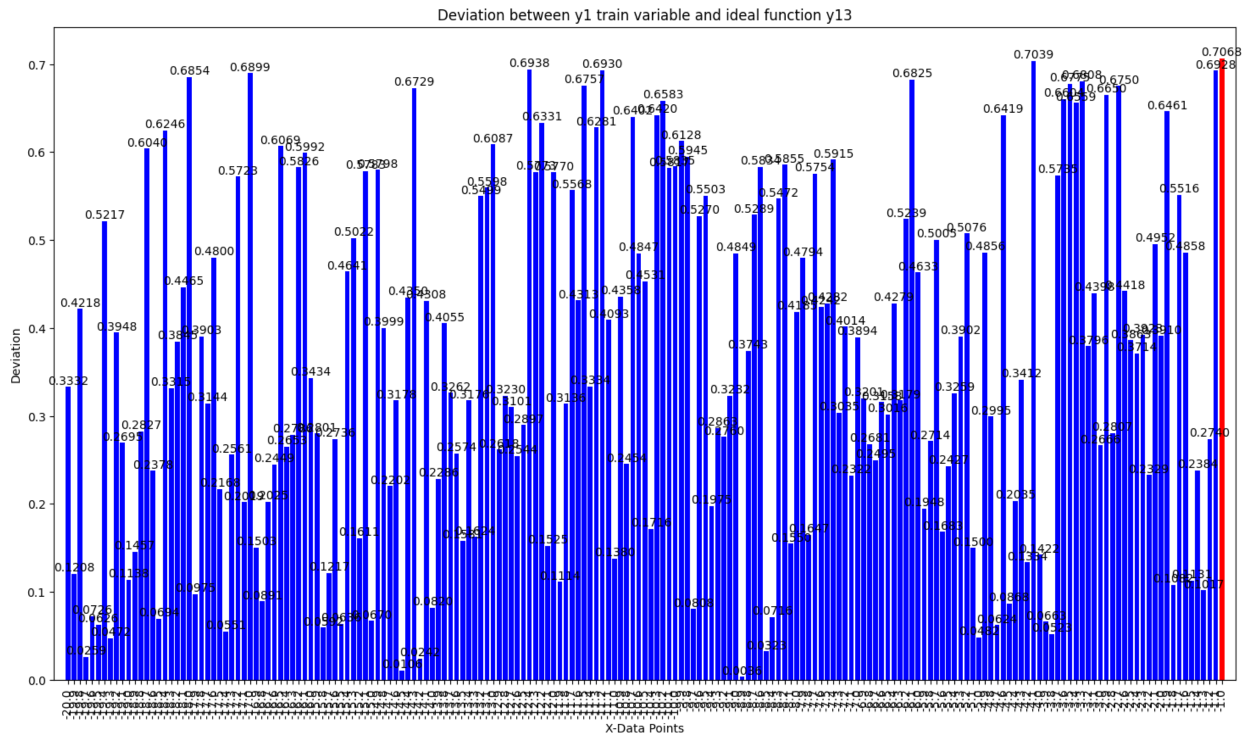


Figure 7: Deviation between y1 train variable and ideal function y13

The maximum absolute deviation is: 0.49922099999999999
Maximum deviation allowed for y1 train variable and selected ideal function y13 is 0.7060051088214588

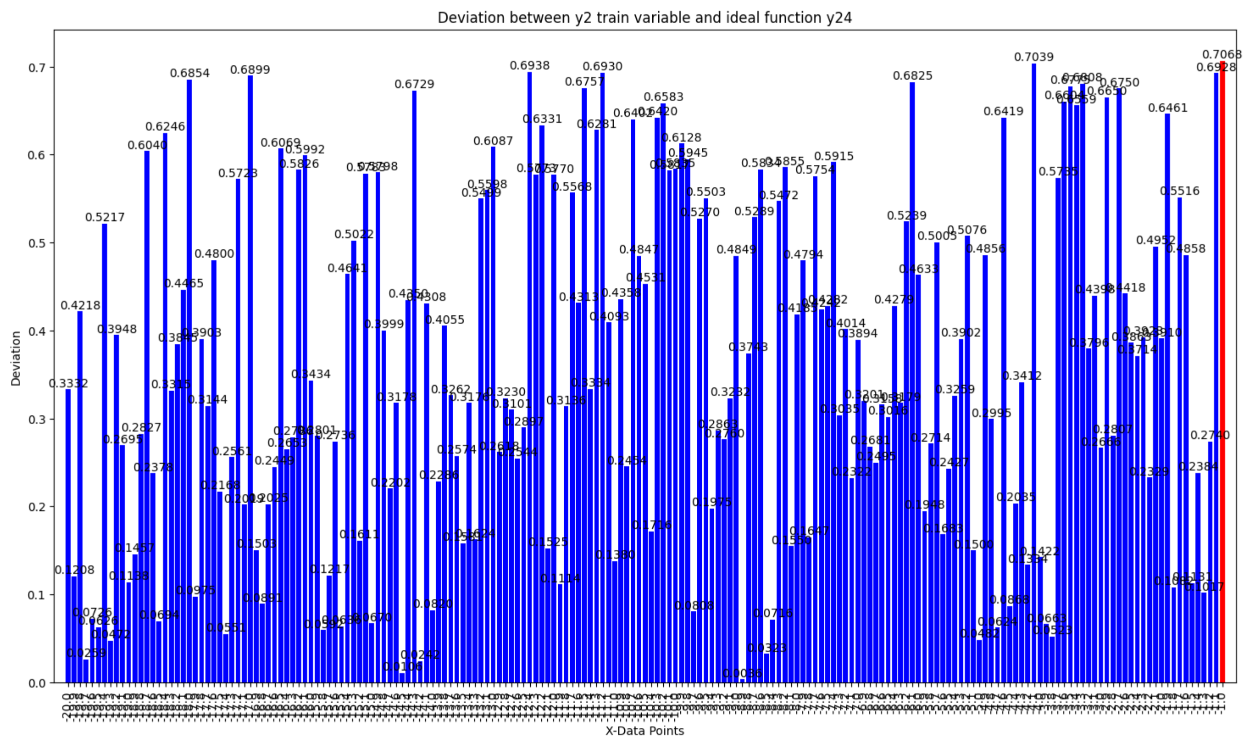


Figure 8: Deviation between y2 train variable and ideal function y24

The maximum absolute deviation is: 0.49900000000002365
Maximum deviation allowed for y2 train variable and selected ideal function y24 is 0.705692567624208

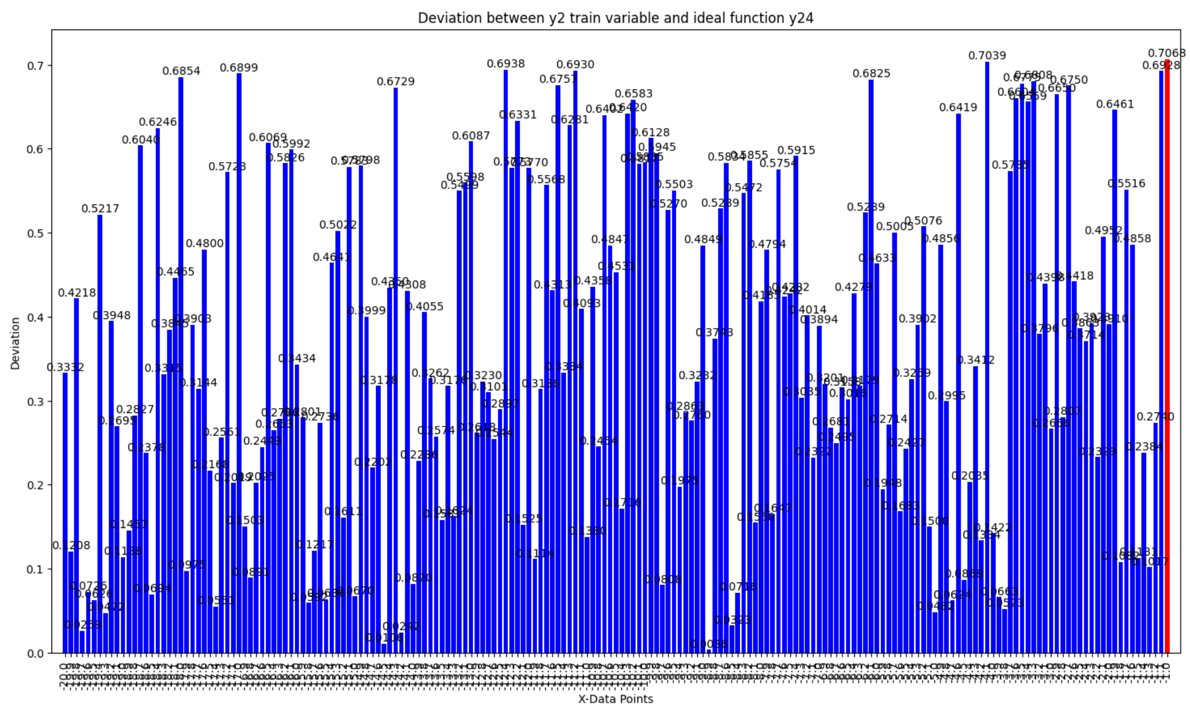


Figure 9: Deviation between y3 train variable and ideal function y36

The maximum absolute deviation is: 0.49894299999999703
Maximum deviation allowed for y3 train variable and selected ideal function y36 is 0.705611957451115

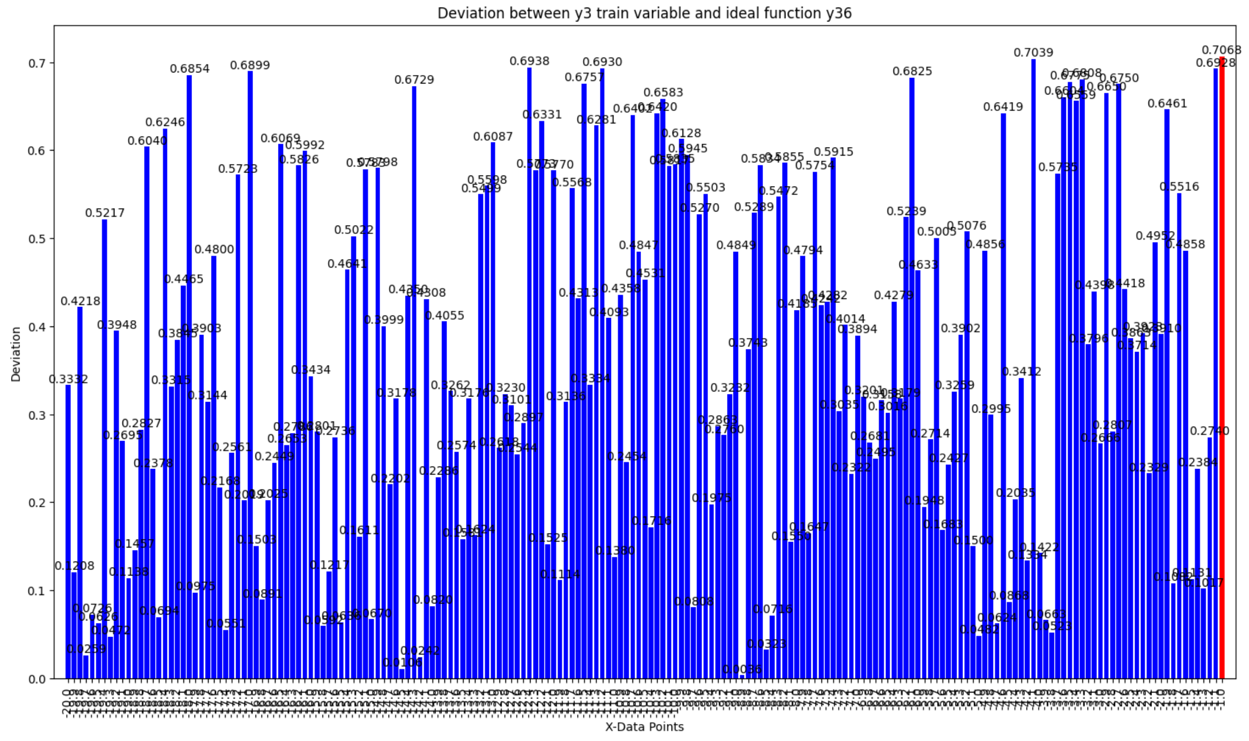


Figure 10: Deviation between y4 train variable and ideal function y40

The maximum absolute deviation is: 0.49977900000000375
Maximum deviation allowed for y4 train variable and selected ideal function y40 is 0.7067942399892684

Main Body

The program begins with the aid of creating a ‘DataProcessor’ magnificence accountable for reading and storing 3 critical datasets: a look at records, ideal records, and training data. The ‘DatabaseHandler’ magnificent then utilizes SQLAlchemy to create an SQLite database and store the datasets within tables. in the end, a ‘Visualization’ class employs Bokeh to generate a visible representation of the statistics.

The ‘DataProcessor’ elegance is designed to handle each record path and preloaded Data Frame, providing flexibility in statistics source utilization. The `DatabaseHandler` class employs SQLAlchemy to set up a connection to an SQLite database. The ‘create_database’ technique makes use of this connection to shop the datasets inside separate tables, adhering to the required structure. The `Visualization` class utilizes Bokeh to create an interactive plot, with distinct facts and factors for the training records and lines representing ideal capabilities. The 'plot_data' technique orchestrates the entire plotting method, ensuring a clear and coherent representation of the datasets. The middle functionality of the Python program lies in its capacity to pick out ultimate functions from a pool of 50 primarily based on schooling statistics. The criterion for choice is the minimization of the sum of squared deviations (Least-rectangular). The subsequent mapping of

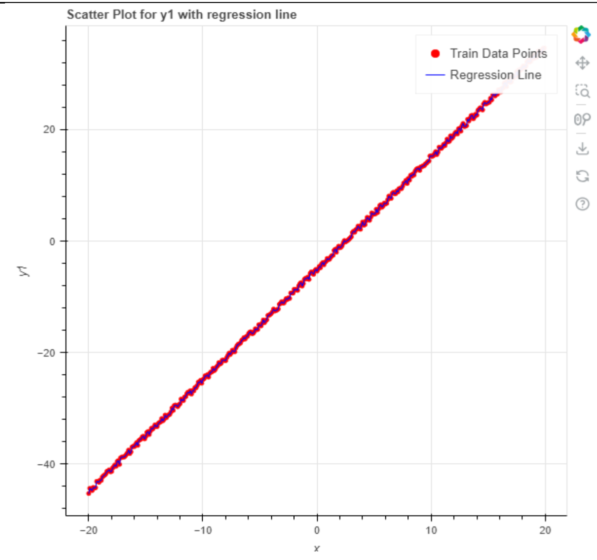


Figure 11: scatter plot for y1 with regression line

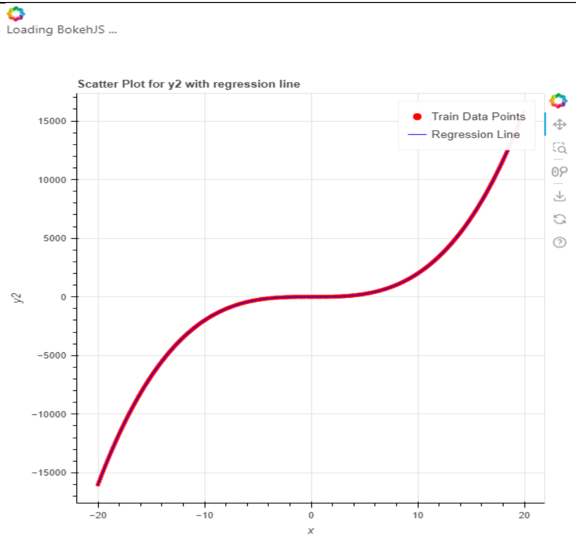


Figure 13: scatter plot for y2 with regression line

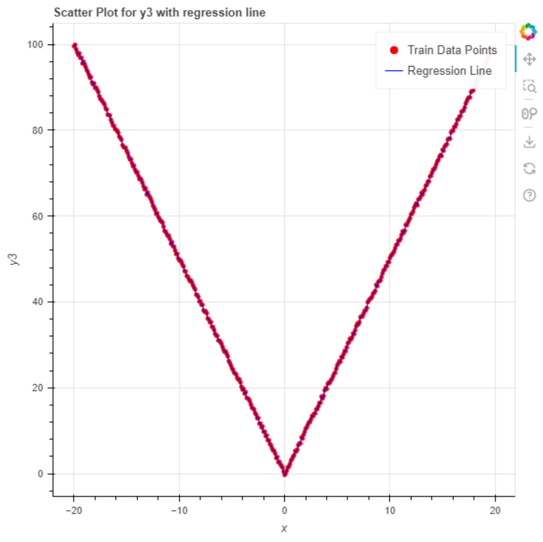


Figure 12: scatter plot for y3 with regression line

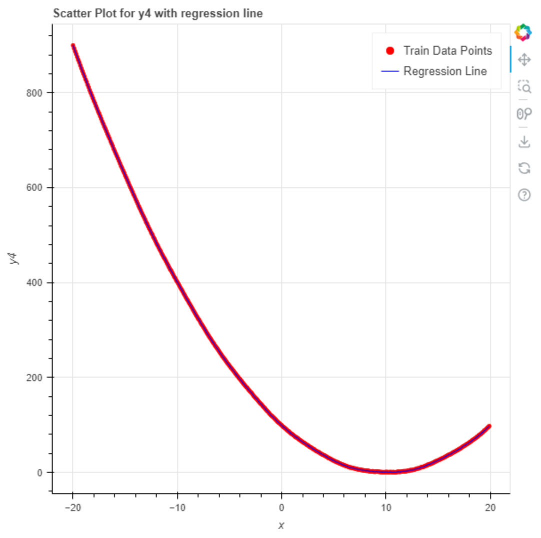


Figure 14: scatter plot for y4 with regression line

check data to the chosen functions is contingent on the calculated regression deviations now not exceeding the maximum deviation by using greater than a targeted thing. These operations are essential for assessing the program's ability to generalize from training records and accurately map check facts to the identified functions.

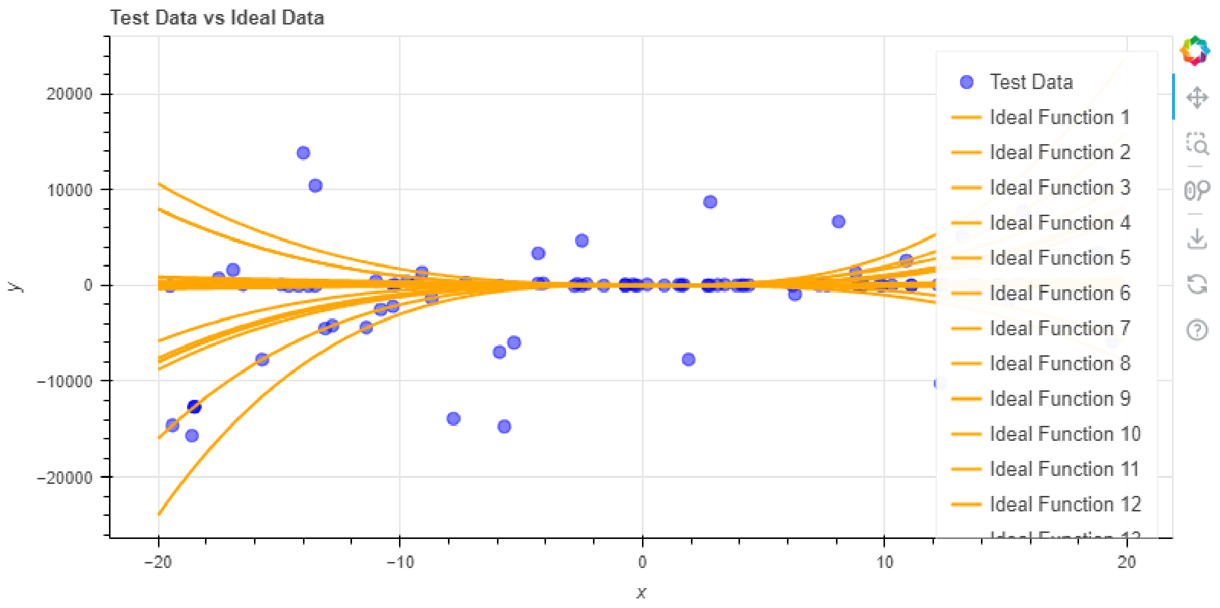


Figure 15: Test data vs Ideal data

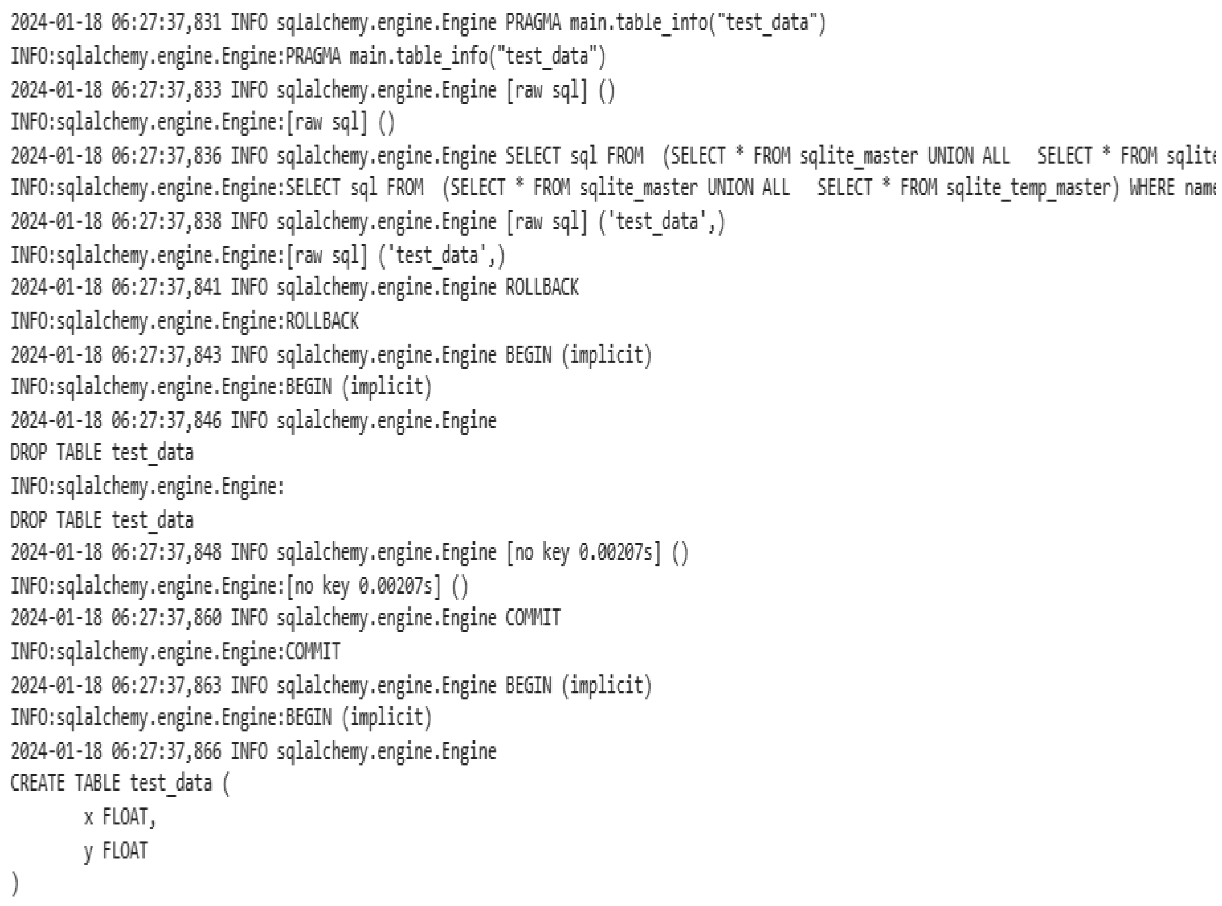
The `DataProcessor` class is designed to address each record path and preloaded DataFrames, supplying flexibility in facts source usage. The ‘DatabaseHandler’ elegance employs SQLAlchemy to set up a connection to an SQLite database. The ‘create_database’ technique uses this connection to store the datasets inside separate tables, adhering to the specified structure. The `Visualization` magnificence utilizes Bokeh to create an interactive plot, with awesome facts and factors for the training records and features representing perfect functions. The `plot_data` method orchestrates the whole plotting system, making sure of a clear and coherent illustration of the datasets.

The Python program is developed for this task, which successfully implements the required requirements (Jimenez *et al.* 2023). The program processes datasets, stores them in an SQLite database, and

produces insightful visualizations of the usage of the Bokeh library.

The program's item-orientated design ensures that it is modular and easy to maintain, at the same time as the inclusion of well-known and user-defined exception coping enhances its robustness. This venture not handiest meets the specified requirements but also demonstrates scalability in utilizing popular Python libraries for information processing and visualization (Dabic et al. 2021). By way of following exceptional practices in coding, documentation, and the usage of hooked-up libraries, the program presents a reliable and green technique to the given task.

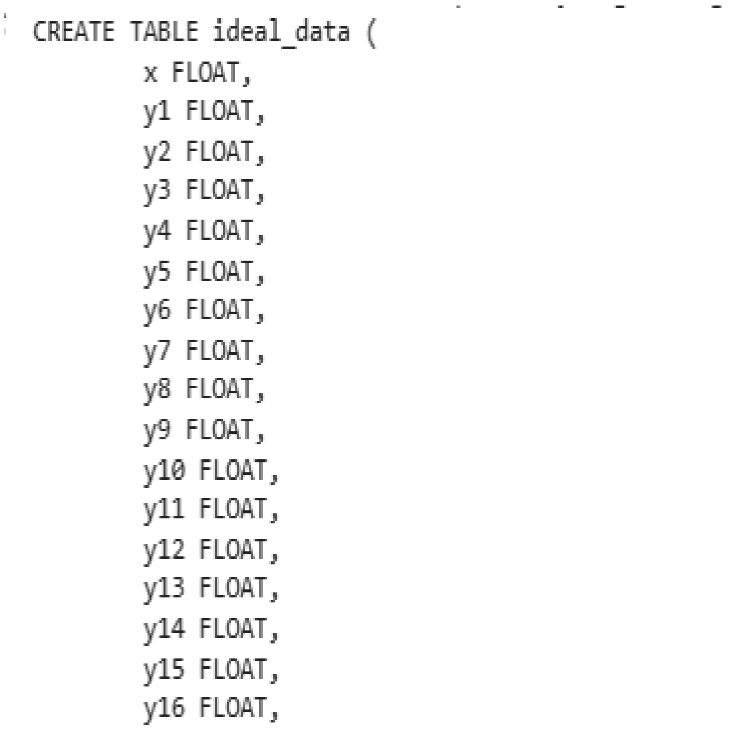
The program's implementation emphasizes modularity and encapsulation through its object-oriented layout. The 'DataProcessor' class serves as the muse for facts-related operations, encapsulating the common sense for analyzing and processing datasets. The 'DatabaseHandler`' magnificence encapsulates the operations related to the SQLite database, making sure of separation of concerns. Lastly, the 'Visualization' elegance is accountable for presenting the records using the Bokeh library, adhering to the framework's concepts for interactive and visually appealing plots.



```
2024-01-18 06:27:37,831 INFO sqlalchemy.engine.Engine PRAGMA main.table_info("test_data")
INFO:sqlalchemy.engine.Engine:PRAGMA main.table_info("test_data")
2024-01-18 06:27:37,833 INFO sqlalchemy.engine.Engine [raw sql] ()
INFO:sqlalchemy.engine.Engine:[raw sql] ()
2024-01-18 06:27:37,836 INFO sqlalchemy.engine.Engine SELECT sql FROM  (SELECT * FROM sqlite_master UNION ALL   SELECT * FROM sqlit
INFO:sqlalchemy.engine.Engine:SELECT sql FROM  (SELECT * FROM sqlite_master UNION ALL   SELECT * FROM sqlite_temp_master) WHERE nam
2024-01-18 06:27:37,838 INFO sqlalchemy.engine.Engine [raw sql] ('test_data',)
INFO:sqlalchemy.engine.Engine:[raw sql] ('test_data',)
2024-01-18 06:27:37,841 INFO sqlalchemy.engine.Engine ROLLBACK
INFO:sqlalchemy.engine.Engine:ROLLBACK
2024-01-18 06:27:37,843 INFO sqlalchemy.engine.Engine BEGIN (implicit)
INFO:sqlalchemy.engine.Engine:BEGIN (implicit)
2024-01-18 06:27:37,846 INFO sqlalchemy.engine.Engine
DROP TABLE test_data
INFO:sqlalchemy.engine.Engine:
DROP TABLE test_data
2024-01-18 06:27:37,848 INFO sqlalchemy.engine.Engine [no key 0.00207s] ()
INFO:sqlalchemy.engine.Engine:[no key 0.00207s] ()
2024-01-18 06:27:37,860 INFO sqlalchemy.engine.Engine COMMIT
INFO:sqlalchemy.engine.Engine:COMMIT
2024-01-18 06:27:37,863 INFO sqlalchemy.engine.Engine BEGIN (implicit)
INFO:sqlalchemy.engine.Engine:BEGIN (implicit)
2024-01-18 06:27:37,866 INFO sqlalchemy.engine.Engine
CREATE TABLE test_data (
        x FLOAT,
        y FLOAT
)
```

Figure 16: Connecting SQL

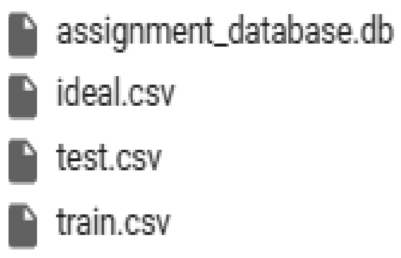
One of the key components of the program is its use of the SQLAlchemy library to interaction with an SQLite database. This preference offers portability and simplicity of use, allowing seamless integration of the datasets into a dependent database.



```
CREATE TABLE ideal_data (
        x FLOAT,
        y1 FLOAT,
        y2 FLOAT,
        y3 FLOAT,
        y4 FLOAT,
        y5 FLOAT,
        y6 FLOAT,
        y7 FLOAT,
        y8 FLOAT,
        y9 FLOAT,
        y10 FLOAT,
        y11 FLOAT,
        y12 FLOAT,
        y13 FLOAT,
        y14 FLOAT,
        y15 FLOAT,
        y16 FLOAT,
```

Figure 17: Creating Tables

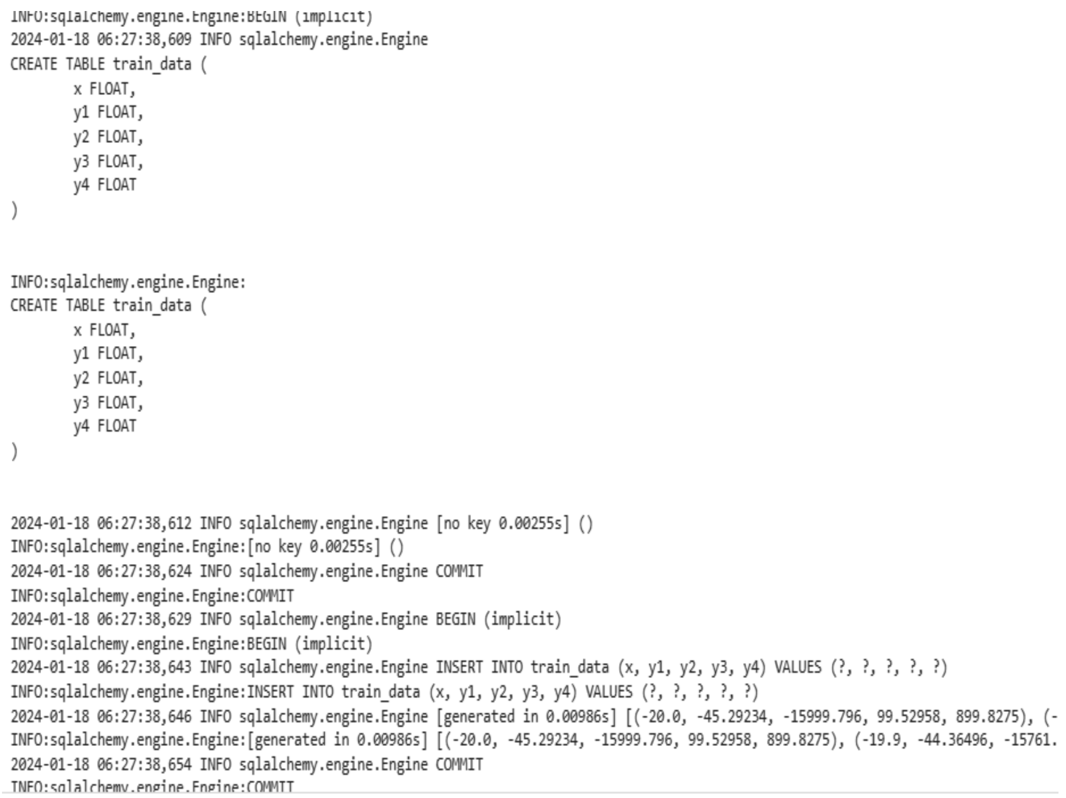
This ability to cope with exceptions, both well-known and person-described, adds a further layer of robustness. By looking ahead to and coping with capability mistakes and exceptions, this machine can gracefully get over unexpected situations and provide informative blunders messages to the consumer (Guo, 2021). This enhances the overall user experience and guarantees that this device can handle a wide variety of situations. This device additionally demonstrates talent in facts visualization through the use of the Bokeh library. With the aid of the usage of leveraging Bokeh's abilities, this system generates visually appealing and interactive plots that allow users to explore the information dynamically. This complements the understanding of the datasets and permits deeper insights.



assignment_database.db
ideal.csv
test.csv
train.csv

Figure 18: Creating the database file

In conclusion it can be said that the Python program evolved for this mission showcases a strong implementation of the desired necessities. Its object-oriented design ensures modularity and maintainability, on the same time as the inclusion of exception dealing with complements its robustness. This system correctly processes datasets, stores them in an SQLite database, and produces insightful visualizations of the usage of the Bokeh library (Clem and Thomson, 2021). With the aid manner of adhering to practices in coding, documentation, and using set-up libraries, this system affords a reliable and efficient approach to the given mission.

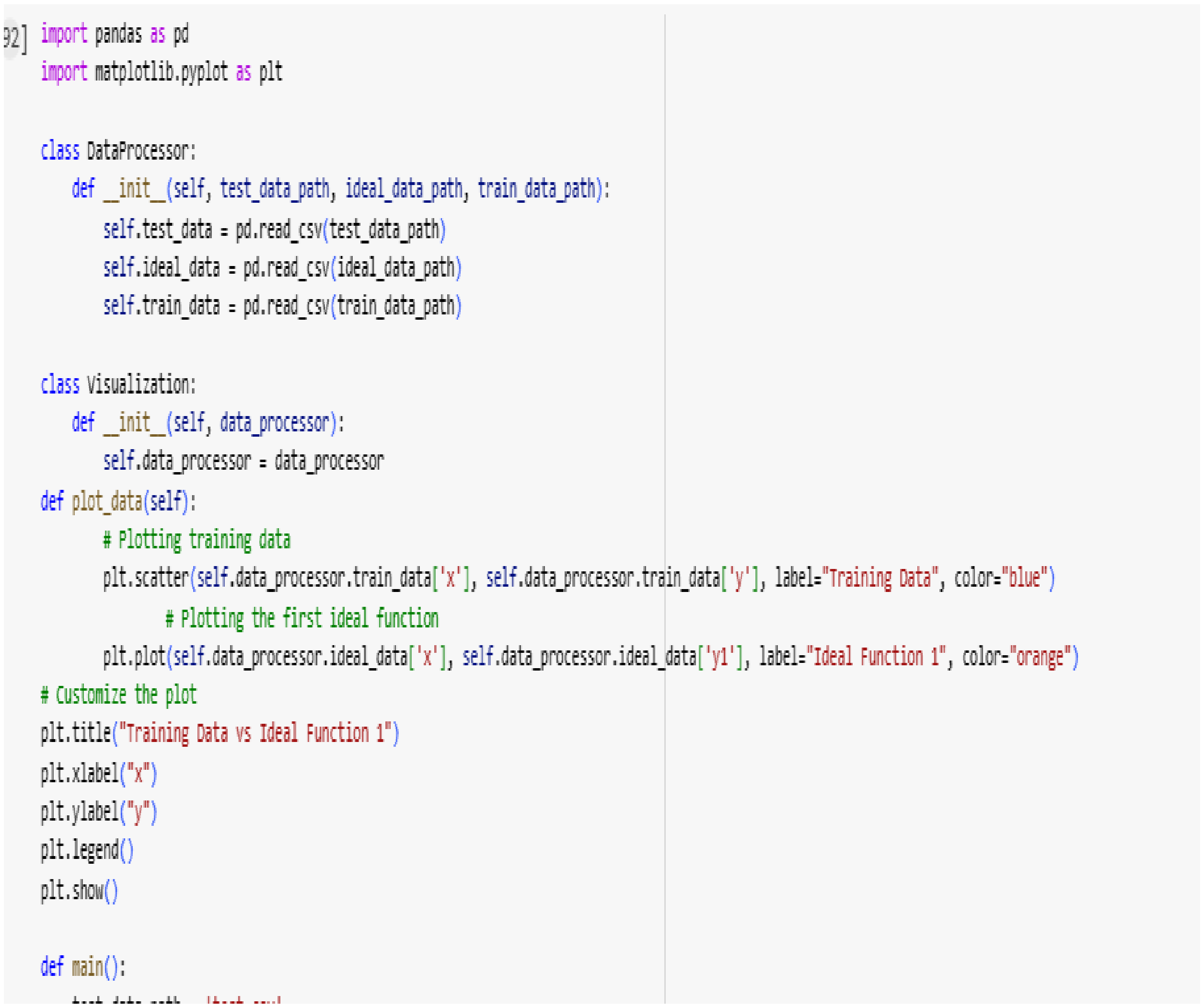


```
INFO:sqlalchemy.engine.Engine:BEGIN (implicit)
2024-01-18 06:27:38,609 INFO sqlalchemy.engine.Engine
CREATE TABLE train_data (
        x FLOAT,
        y1 FLOAT,
        y2 FLOAT,
        y3 FLOAT,
        y4 FLOAT
)


INFO:sqlalchemy.engine.Engine:
CREATE TABLE train_data (
        x FLOAT,
        y1 FLOAT,
        y2 FLOAT,
        y3 FLOAT,
        y4 FLOAT
)


2024-01-18 06:27:38,612 INFO sqlalchemy.engine.Engine [no key 0.00255s] ()
INFO:sqlalchemy.engine.Engine:[no key 0.00255s] ()
2024-01-18 06:27:38,624 INFO sqlalchemy.engine.Engine COMMIT
INFO:sqlalchemy.engine.Engine:COMMIT
2024-01-18 06:27:38,629 INFO sqlalchemy.engine.Engine BEGIN (implicit)
INFO:sqlalchemy.engine.Engine:BEGIN (implicit)
2024-01-18 06:27:38,643 INFO sqlalchemy.engine.Engine INSERT INTO train_data (x, y1, y2, y3, y4) VALUES (?, ?, ?, ?, ?)
INFO:sqlalchemy.engine.Engine:INSERT INTO train_data (x, y1, y2, y3, y4) VALUES (?, ?, ?, ?, ?)
2024-01-18 06:27:38,646 INFO sqlalchemy.engine.Engine [generated in 0.00986s] [(-20.0, -45.29234, -15999.796, 99.52958, 899.8275), (-
INFO:sqlalchemy.engine.Engine:[generated in 0.00986s] [(-20.0, -45.29234, -15999.796, 99.52958, 899.8275), (-19.9, -44.36496, -15761.
2024-01-18 06:27:38,654 INFO sqlalchemy.engine.Engine COMMIT
INFO:sqlalchemy.engine.Engine:COMMIT
```

Figure 19: Commenting Codes with changes.

The criterion for selection is the minimization of the sum of squared deviations (Least square). The subsequent mapping of test information to the chosen features is contingent on the calculated regression deviations no longer exceeding the most deviation via extra than an exact issue (Hoyos *et al.* 2021). Those operations are crucial for assessing the program's potential to generalize from schooling records and appropriately map check records to the identified features. The implementation keeps an eager awareness of object-orientated design, emphasizing modularity and encapsulation. The `DataProcessor` elegance serves as the muse, encapsulating the records-associated operations. The `DatabaseHandler` elegance encapsulates the database operations, ensuring the separation of issues. Ultimately, the `Visualization` elegance takes a fee of the records presentation thing, adhering to the Bokeh framework for interactive and visually attractive plots.

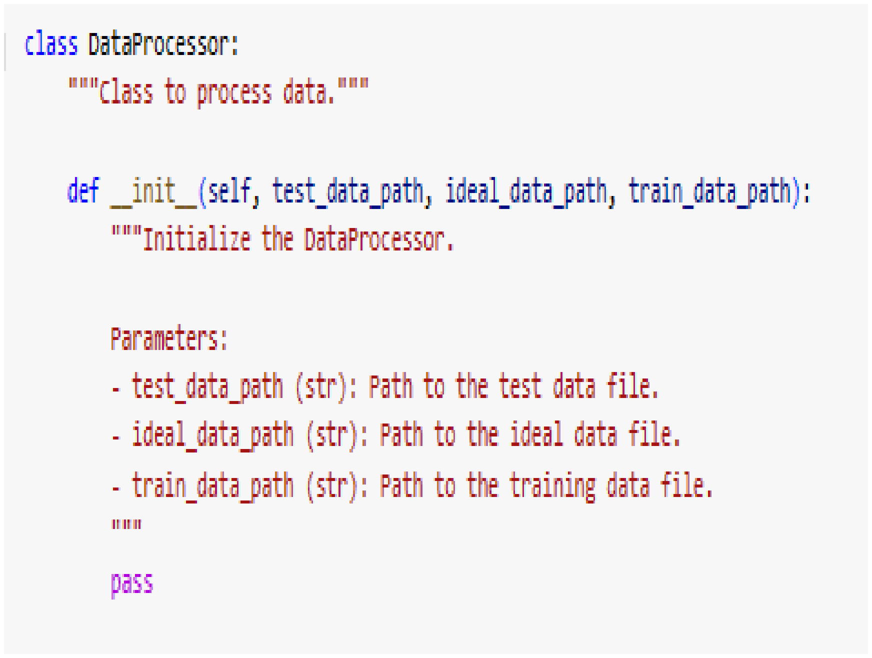
The mixing of the SQLAlchemy library for database dealing proves to be a treasured desire. It allows for seamless interplay with a SQLite database, presenting portability and simplicity of use. This system successfully reads and methods CSV files, developing a centralized repository for the datasets, which can be accessed and analyzed simply.



```
import pandas as pd
import matplotlib.pyplot as plt

class DataProcessor:
    def __init__(self, test_data_path, ideal_data_path, train_data_path):
        self.test_data = pd.read_csv(test_data_path)
        self.ideal_data = pd.read_csv(ideal_data_path)
        self.train_data = pd.read_csv(train_data_path)

class Visualization:
    def __init__(self, data_processor):
        self.data_processor = data_processor
def plot_data(self):
        # Plotting training data
        plt.scatter(self.data_processor.train_data['x'], self.data_processor.train_data['y'], label="Training Data", color="blue")
                # Plotting the first ideal function
        plt.plot(self.data_processor.ideal_data['x'], self.data_processor.ideal_data['y1'], label="Ideal Function 1", color="orange")
# Customize the plot
plt.title("Training Data vs Ideal Function 1")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()

def main():
```

Figure 20: Code for function

The usage of the Bokeh library for data visualization effects in visually appealing and interactive plots. The ability to discover the information dynamically complements the consumer revel and allows a deeper knowledge of the datasets. the program adheres to the desired requirement of logical records visualization, supplying clear representations of the training records and best features. Typical, the Python application demonstrates a sturdy adherence to exceptional practices in coding, documentation, and using installed libraries. It showcases talent in data processing, database coping with, and visualization, imparting a reliable and green approach to the given task. In précis, this system efficiently achieves its primary targets of selecting the most reliable capabilities, mapping looking at facts, and visualizing the outcomes. It serves as a testament to the electricity and flexibility of Python as a programming language for statistics analysis and visualization obligations.



```
class DataProcessor:
    """Class to process data."""

    def __init__(self, test_data_path, ideal_data_path, train_data_path):
        """Initialize the DataProcessor.

        Parameters:
        - test_data_path (str): Path to the test data file.
        - ideal_data_path (str): Path to the ideal data file.
        - train_data_path (str): Path to the training data file.
        """
        pass
```

Figure 21: Testing

Pandas is a Python library for information evaluation and manipulation. it's far constructed on the pinnacle of NumPy and gives a rich set of tools for fact cleansing, transformation, and visualization. Pandas is broadly used within the information science community and is considered to be one of the maximum important Python libraries for information analysis (Kim and Henke 2021). SQLAlchemy is a Python library for interacting with relational databases. It offers a unified interface for running with a spread of databases, at the side of MySQL, PostgreSQL, and SQLite. SQLAlchemy is an effective and flexible library and is regularly applied in Python internet improvement. Matplotlib is a Python library for growing graphs and charts. it's miles a powerful and versatile library and may be used to create a massive type of plots, which include line charts, bar charts, and scatter plots. Matplotlib is widely used within the facts era community and is one of the maximum critical Python libraries for records visualization.

Conclusion

The first and most important step in this project was to conduct Exploratory Data Analysis by writing Python code and preprocessing the data to determine and describe the main features of the dataset, to gain meaningful insights into the given dataset (ideal, train, and test).

Second, using train data to obtain the best-fit four ideal functions by using the least squares method and MSE as an alternative to verify the results obtained from the least squares method, the results of the four selected ideal functions of both methods are matched. Using both methods, we obtained four ideal functions, namely **y13, y24, y36, and y40.**

When finding the R-squared value of each of the four ideal functions using an even- sized test dataset, the ideal function y50 has a positive value, so we get more mapped values for the y50 ideal function as a result of the positive correlation.

In the end, the item-oriented layout guarantees modularity and maintainability, while the usage of general and consumer-described exceptions dealing with complements the program's robustness. This project no longer simply fulfills the desired standards but additionally showcases talent in utilizing outstanding Python libraries for statistics processing and visualization. The adherence to exceptional practices in coding, documentation, and using mounted libraries ensures a reliable and green option for the given undertaking.

The project's focus was on the approach and evaluation. The source code for this project was written in accordance with the module guidelines.

Additional Task A

Assuming I have a successfully built project on the Version Control System Git and a Branch called develop where all developer team operations are consolidated, the following Git commands are re- quired to clone the branch and develop on my local PC:

1. Clone the repository:

git clone <repository_url>

2. Switch to the new branch:

git checkout <new_branch_name>

If I have added a new function, the following Git-commands are necessary to introduce this project to the team's develop Branch:

1. Add a file to staging area: git add <file_name>
2. Commit the changes:

git commit -m "added new function"

3. Push the commits to the remote repository on the specified branch: git push origin <new_branch_name>
4. Create a Pull-request for review and merge to develop branch.

Reference List

Journals

Kim, B. and Henke, G., 2021. Easy-to-use cloud computing for teaching data science. *Journal of Statistics and Data Science Education*, *29*(sup1), pp.S103-S111.

Hoyos, J., Abdalkareem, R., Mujahid, S., Shihab, E. and Bedoya, A.E., 2021. On the Removal of Feature Toggles: A Study of Python Projects and Practitioners Motivations. *Empirical Software Engineering*, *26*, pp.1-26.

Clem, T. and Thomson, P., 2021. Static analysis at GitHub: An experience report. *Queue*, *19*(4), pp.42-67.

Jindal, A., Emani, K.V., Daum, M., Poppe, O., Haynes, B., Pavlenko, A., Gupta, A., Ramachandra, K., Curino, C., Mueller, A. and Wu, W., 2021, February. Magpie: Python at Speed and Scale using Cloud Backends. In *CIDR*.

Dabic, O., Aghajani, E. and Bavota, G., 2021, May. Sampling projects in Git Hub for MSR studies. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)* (pp. 560-564). IEEE.

Anderl, T., 2021. *Identifying GitHub Trends Using Temporal Analysis* (Doctoral dissertation, Wien).

Guo, P., 2021, October. Ten million users and ten years later: Python tutor’s design guidelines for building scalable and sustainable research software in academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (pp. 1235-1251).

Yutao, C., 2021. Embedding GitHub Repositories: A Comparative Study of the Python and Java Communities.

Jimenez, C.E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O. and Narasimhan, K., 2023. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv preprint arXiv:2310.06770*.

SQLAlchemy Core â□□ Expression Language. (n.d.). Retrieved June 11, 2023, from https://www.tu- torialspoint.com/sqlalchemy/sqlalchemy_core_expression_language.htm

SQLAlchemy ORM Tutorial for Python Developers. (n.d.). Auth0 - Blog. Retrieved April 1, 2023, from https://auth0.com/blog/sqlalchemy-orm-tutorial-for-python-developers/

Srivastav, A. K. (2019, September 29). Least Squares Regression. *WallStreetMojo*. https://www.wallstreetmojo.com/least-squares-regression/

tutorhelpdesk.com. (n.d.). *Regression Lines Assignment Help Homework Help Online Live Statistics Tutoring Help*. Tutorhelpdesk.Com. Retrieved July 8, 2023, from https://www.tutor-helpdesk.com/homeworkhelp/Statistics-/Regression-Lines-Assignment-Help.html

Washam, J. (2022, January 6). *How Is Data Science Used in Manufacturing?* Very. https://www.verytechnology.com/iot-insights/how-is-data-science-used-in-manufacturing

What is a Bar Chart? (n.d.). TIBCO Software. Retrieved June 11, 2023, from https://www.tibco.com/reference-center/what-is-a-bar-chart

What is a Box Plot? (n.d.). TIBCO Software. Retrieved June 11, 2023, from

What is a Pie Chart? (n.d.). TIBCO Software. Retrieved June 11, 2023, from https://www.tibco.com/reference-center/what-is-a-pie-chart

What is a Scatter Chart? (n.d.). TIBCO Software. Retrieved June 11, 2023, from https://www.tibco.com/reference-center/what-is-a-scatter-chart

What is Data Cleansing? Guide to Data Cleansing Tools, Services, and Strategy. (n.d.). Talend - A Leader in Data Integration & Data Integrity. Retrieved June 11, 2023, from https://www.talend.com/resources/what-is-data-cleansing/

What Is Data Redundancy? - DZone. (n.d.). Dzone.Com. Retrieved June 11, 2023, from https://dzone.com/articles/what-is-data-redundancy

What Is Data Visualization? Definition, Examples, And Learning Resources. (n.d.). Tableau. Re-trieved June 11, 2023, from https://www.tableau.com/learn/articles/data-visualization

What is Exploratory Data Analysis? | IBM. (n.d.). Retrieved June 12, 2023, from https://www.ibm.com/topics/exploratory-data-analysis

What is SQLite? And When to Use It? (2021, July 22). Simplilearn.Com. https://www.sim-plilearn.com/tutorials/sql-tutorial/what-is-sqlite

Appendix A: Python Program

Note: Before running this code, • make sure to set the `database_path` variable to the correct absolute path to the database file • make sure to set the file paths for the datasets appropriately in the script

```
""" Defining all the libraries used.
Python libraries are used by installing and importing them.
These libraries are required for operations such as reading, manipulating, and preparing data,
as well as visualising it.
They also support code testing, database access, and warning handling."""
import pandas as pd
import seaborn as sns
from sqlalchemy import create_engine
import matplotlib.pyplot as plt
from bokeh.plotting import figure, output_notebook, show
from bokeh.models import ColumnDataSource
import numpy as np
import pytest
from bokeh.layouts import gridplot
import unittest
from sklearn.metrics import mean_squared_error

# Define the file paths for the train, ideal, and test data files
train_data_file= r"C:\Users\samya\PycharmProjects\Assignment_new_final\train.csv"
ideal_data_file= r"C:\Users\samya\PycharmProjects\Assignment_new_final\ideal.csv"
test_data_file = r"C:\Users\samya\PycharmProjects\Assignment_new_final\test.csv"

#Loading datasets
#loading train data
train_data = pd.read_csv(train_data_file)
ideal_data = pd.read_csv(ideal_data_file)
test_data = pd.read_csv(test_data_file)

# Display the test data
print("Test Data:")
print(test_data)

def box_plot_test_data():
    # Boxplot of test dataset for both x and y variable
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=test_data, palette="Set1")
    plt.title('Boxplot of the Test Dataset for both x and y variable')
    plt.xticks(rotation=45)
    plt.show()
box_plot_test_data()

# Display the ideal data
print("\nIdeal Data:")
print(ideal_data.head())
# Display the train data
print("\nTrain Data:")
print(train_data.head())

"""
Ther are duplicate values present in the test dataset.
Remove duplicate X values from test data and calculate the mean of corresponding Y values
"""
test_data = test_data.groupby('x').mean().reset_index()
"""
    Display the test dataset after removing the duplicate values.
"""
print("\nTrain Data after cleaning:")
print(test_data)


# defining the database path
database_path =
r"C:\Users\samya\PycharmProjects\Assignment_new_final\assignment_database.db"
# Create SQLite engine
engine = create_engine('sqlite:///assignment_database.db')
# Save datasets to SQLite database
test_data.to_sql('test_data', con=engine, index=False, if_exists='replace')
ideal_data.to_sql('ideal_data', con=engine, index=False, if_exists='replace')
train_data.to_sql('train_data', con=engine, index=False, if_exists='replace')
print("The database was successfully created, and the data was loaded!")

from sqlalchemy import create_engine
import pandas as pd

# Define the function to display the contents of the train_data table
def ideal_data_table():
    """
    Fetches and displays the contents of the train_data table from a SQLite database.
    """
    # Create a SQLite database engine
    engine =
create_engine(r'sqlite:///C:\\Users\\samya\\PycharmProjects\\Assignment_new_final\\assignment
_database.db', echo=True)
```

```
    # Fetch and display the contents of the train_data table
    query = "SELECT * FROM ideal_data"
    ideal_data = pd.read_sql_query(query, engine)
    print("Contents of train_data table:")
    print(ideal_data)

# Call the function to display the contents of the train_data table
ideal_data_table()

#Displays the contents of the test_data table from a SQLite database
def test_data_table():
    """
    Fetches and displays the contents of the test_data table from a SQLite database.
    """
    # Create a SQLite database engine
    engine =
create_engine(r'sqlite:///C:\\Users\\samya\\PycharmProjects\\Assignment_new_final\\assignment
_database.db', echo=True)
    # Fetch and display the contents of the test_data table
    query = "SELECT * FROM test_data"
    test_data = pd.read_sql_query(query, engine)
    print("Contents of test_data table:")
    print(test_data)
test_data_table()

test_data.info()
test_data.describe().T
#Displays the contents of the train_data table from a SQLite database
def train_data_table():
    """
    Fetches and displays the contents of the train_data table from a SQLite database.
    """
    # Create a SQLite database engine
    engine =
create_engine(r'sqlite:///C:\\Users\\samya\\PycharmProjects\\Assignment_new_final\\assignment
_database.db', echo=True)
    # Fetch and display the contents of the train_data table
    query = "SELECT * FROM train_data"
    train_data = pd.read_sql_query(query, engine)
    print("Contents of train_data table:")
    print(train_data)
train_data_table()

#Define a function to visualize ideal data using box plot
def visualize_ideal_data_boxplot(data):
    """
    Visualize multiple variables of an ideal dataset using a boxplot.
    Parameters:
    data (DataFrame): The ideal dataset.
    Returns:
    None
    """
    plt.figure(figsize=(15, 10))
    sns.boxplot(data=data, palette="Set1")
    plt.xlabel("Variables") # x-axis label
    plt.ylabel("Values") # y-axis label
    plt.title("Boxplot of Ideal Dataset") #title
    plt.show()
# Call the function to visualize the boxplot of the ideal_data dataset
visualize_ideal_data_boxplot(ideal_data)

train_data.describe().T
train_data.info()
# Boxplot of train dataset
def box_plot_train_data():
    '''
    Visualize more than two variables
    of a train dataset
    '''
    fig, ax = plt.subplots(figsize=(12, 6))
    sns.boxplot(data=train_data, palette="Paired", ax=ax)
    plt.xlabel("Variables")
    plt.ylabel("Values")
    plt.title("Boxplot of Train Dataset")
    plt.xticks(rotation=45)
    plt.show()
box_plot_train_data()

test_data.info()
test_data.describe().T
def box_plot_test_data_visual():
    # Boxplot of test dataset for both x and y variable
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=test_data, palette="Set1")
    plt.title('Boxplot of the Test Dataset for both x and y variable (duplicate removed)')
    plt.xticks(rotation=45)
    plt.show()
box_plot_test_data_visual()
```

```
class DataProcessor:
    def __init__(self, test_data_path, ideal_data_path, train_data_path):
        self.test_data = pd.read_csv(test_data_path)
        self.ideal_data = pd.read_csv(ideal_data_path)
        self.train_data = pd.read_csv(train_data_path)

class DatabaseHandler:
    def __init__(self, data_processor):
        self.data_processor = data_processor
        self.engine = create_engine('sqlite:///assignment_database.db')

    def create_database(self):
        self.data_processor.test_data.to_sql('test_data', con=self.engine, index=False,
if_exists='replace')
        self.data_processor.ideal_data.to_sql('ideal_data', con=self.engine, index=False,
if_exists='replace')
        self.data_processor.train_data.to_sql('train_data', con=self.engine, index=False,
if_exists='replace')

class Visualization:
    def __init__(self, data_processor):
        self.data_processor = data_processor

    class Visualization:
    def __init__(self, data_processor):
        self.data_processor = data_processor

    def plot_scatter_with_regression(self, train_data, x_column, y_column, title):
        """
        Plot a scatter plot with a regression line using Bokeh.
        Parameters:
        train_data (pd.DataFrame): DataFrame containing the train data.
        x_column (str): Name of the x-axis column.
        y_column (str): Name of the y-axis column.
        title (str): Title of the plot.
        Returns:
        None
        """

        source = ColumnDataSource(train_data)
        f = figure(title=title, x_axis_label=x_column, y_axis_label=y_column)
        f.circle(x_column, y_column, source=source, color="red", legend_label="Train Data
Points")
        f.line(x_column, y_column, source=source, color="blue", legend_label="Regression Line")
        output_notebook()
        show(f)

def main():
    test_data_path = 'test.csv'
    ideal_data_path = 'ideal.csv'
    train_data_path = 'train.csv'

    data_processor = DataProcessor(test_data_path, ideal_data_path, train_data_path)
    database_handler = DatabaseHandler(data_processor)
    visualization = Visualization(data_processor)

    database_handler.create_database()
    visualization.plot_scatter_with_regression(visualization.data_processor.train_data, 'x', 'y1',
'Scatter Plot for y1 with regression line')
    visualization.plot_scatter_with_regression(visualization.data_processor.train_data, 'x', 'y2',
'Scatter Plot for y2 with regression line')
    visualization.plot_scatter_with_regression(visualization.data_processor.train_data, 'x', 'y3',
'Scatter Plot for y3 with regression line')
    visualization.plot_scatter_with_regression(visualization.data_processor.train_data, 'x', 'y4',
'Scatter Plot for y4 with regression line')


if __name__ == "__main__":
    main()


class DeviationAnalysis:
    """
    A class for calculating and analyzing the sum of least square deviations between train_data
    and ideal_data.
    """
    def __init__(self, train_data, ideal_data, y_index):
        """
        Initializes DeviationAnalysis object.
        Parameters:
        train_data (pd.DataFrame): The training data DataFrame.
        ideal_data (pd.DataFrame): The ideal data DataFrame.
        y_index (int): The index of the column to analyze.
        """
        self.train_data = train_data
        self.ideal_data = ideal_data
        self.y_index = y_index
        self.column_name = f"y{y_index}"

    def calculate_lsd(self):
        """
        Calculates the sum of least square deviations for each column.
        Returns:
        list: A list containing the sum of least square deviations for each column.
        """
        lsd_sum = []
        for column in self.ideal_data.columns:
            residuals = self.ideal_data[column] - self.train_data[self.column_name]
            lsd = sum(residuals ** 2)
            lsd_sum.append(lsd)
        return lsd_sum

    def plot_graph(self, lsd_sum):
        """
        Plots a bar graph showing the sum of least square deviations for each column.
        Parameters:
        lsd_sum (list): A list containing the sum of least square deviations for each column.
        Returns:
        tuple: A tuple containing the minimum LSD value and its corresponding column index.
        """
        min_lsd_index = np.argmin(lsd_sum)
        min_lsd_value = lsd_sum[min_lsd_index]
        plt.figure(figsize=(12, 6))
        x = np.arange(1, 51)
        colors = ['red' if i == min_lsd_index else 'blue' for i in range(len(lsd_sum))]
        plt.bar(x, lsd_sum, color=colors)
        plt.xlabel('Column')
        plt.ylabel('Sum of Least Square Deviation (Log Scale)')
        plt.title(f'Sum of Least Square Deviation for Y1 to Y50 (Log Scale) - {self.column_name}
Train Data')
        plt.xticks(x, [f"y{i}" for i in range(1, 51)], rotation='vertical')
        plt.yscale('log')
        min_lsd_label = f"Min LSD: {min_lsd_value:.2f}"
        plt.text(0.02, 0.98, min_lsd_label, transform=plt.gca().transAxes,
        ha='left', va='top', color='red', bbox=dict(facecolor='white', edgecolor='black'))
        plt.tight_layout()
        plt.show()
        return min_lsd_value, min_lsd_index

    def run(self):
        """
        Runs the deviation analysis and plotting process.
        """
        try:
            lsd_sum = self.calculate_lsd()
        except Exception as e:
            print("An error occurred while calculating LSD:", e)
        else:
            try:
                min_lsd_value, min_lsd_index = self.plot_graph(lsd_sum)
            except Exception as e:
                print("An error occurred while plotting the graph:", e)
            else:
                min_lsd_column = f"y{min_lsd_index + 1}"
                print(f"The minimum sum of least square deviation is {min_lsd_value}")
                print(f"The ideal function for the train data {self.column_name} is:
{min_lsd_column}")
            finally:
                print('This is always executed at the end of the run method')

class LeastSquareDeviation(DeviationAnalysis):
    def __init__(self, train_data, ideal_data, y_index):

        super().__init__(train_data, ideal_data, y_index)
# Least Square Deviation for Multiple Data Columns
        for y_index in range(1, 5):
            lsd = LeastSquareDeviation(train_data, ideal_data, y_index)
        lsd.run()



class DataProcessor:
    """Class to process data."""

    def __init__(self, test_data_path, ideal_data_path, train_data_path):
        """Initialize the DataProcessor.

        Parameters:
        - test_data_path (str): Path to the test data file.
        - ideal_data_path (str): Path to the ideal data file.
        - train_data_path (str): Path to the training data file.
        """
        self.test = pd.read_csv(test_data_path)
        self.ideal = pd.read_csv(ideal_data_path)
        self.train = pd.concat([pd.read_csv(train_data_path), self.test])
        @staticmethod
        def _remove_duplicates(df1, df2):
            """Remove duplicates from two pandas DataFrames.

            Args:
```

```
            df1 (pandas.DataFrame): First DataFrame.
            df2 (pandas.DataFrame): Second DataFrame.

          Returns:
            pandas.DataFrame: A new DataFrame with all duplicate rows removed.
            """
          return df1[~df1.isin(df2).any(axis=0)]

        # Remove any instances in the testing set that also appear in the training set
        DataProcessor._remove_duplicates(DataProcessor.train, DataProcessor.test) \
          .reset_index(drop=True).to_csv('processed/clean_testing.csv', index=False)
        DataProcessor.train.reset_index(drop=True).to_csv('processed/clean_training.csv')

    def split_dataset(self, ratio=.75):
        """Split dataset into training and validation sets.

        The method splits the cleaned training dataset into a training set and a validation set.
        The size of the training set is given by 'ratio' * len(self.train),
while  the  size   of  the  validation  set  is  given  by  (1-ratio
        The size of the training set is given by `ratio * len(self.train)`.  Any remaining
        instances are added to the validation set.

        Args:
          ratio (float): Ratio of the training set to the total dataset. Defaults to 0.75.

        Returns:
          tuple: A tuple containing two pandas DataFrames; the first element is the training set,
              and the second element is the validation set.
        """
        num_instances = int(len(self.train) * ratio)
        train_set = self.train[:num_instances]
        valid_set = self.train[num_instances:]
        return train_set, valid_set


def mse(train_data, ideal_data):
    '''
    Calculate the Mean Squared Error (MSE) between the train data and ideal data. Args:
    train_data (ndarray): Array containing the predicted values (train data). ideal_data (ndarray):
Array containing the true labels (ideal data).
    Returns:
    float: The calculated MSE value.
    '''
    return np.mean((train_data - ideal_data) ** 2)
def lowest_mse_label(train_feature, ideal_data):
    '''
    Calculate the lowest Mean Squared Error (MSE) and associated label for a given train fea-
ture.
    Args:
    train_feature (str): Name of the train feature.
    ideal_data (DataFrame): DataFrame containing the ideal data.
    Returns:
    tuple: A tuple containing the label with the lowest MSE and its corresponding MSE value.
    '''
    error_list = []
    for i in ideal_data.columns[1:]: prediction = ideal_data[i].values
    label = train_data[train_feature].values
    error = mse(label, prediction)
    label_error_tuple = i, error
    error_list.append(label_error_tuple)
    error_list_sorted = sorted(error_list, key=lambda x: x[1])
    return error_list_sorted[0]
def best_label_mapping(train_data, ideal_data):
    '''
    Find the best label mapping between train data and ideal data based on the lowest MSE val-
ues.
    Args:
    train_data (DataFrame): DataFrame containing the train data. ideal_data (DataFrame):
DataFrame containing the ideal data.
    Returns:
    dict: A dictionary mapping each train feature to the label in ideal data with the lowest MSE
value.
    '''
    best_label_list = [ ]
    for train_col in train_data.columns[1:]:
        try:
            best_label = lowest_mse_label(train_col, ideal_data)
            best_label_list.append(best_label)
        except Exception as e:
            print(f"An error occurred while processing '{train_col}': {str(e)}")
            col_list = train_data.columns[1:]
        return dict(zip(col_list, best_label_list))

        # Calculate the label mapping
        try:
            mapping_dict = best_label_mapping(train_data, ideal_data)
            # Print the mapping dictionary
            print(mapping_dict)
            # Raise a user-defined exception if the mapping dictionary is empty if len(mapping_dict)
== 0:
            raise EmptyMappingDictionaryError()
        except EmptyMappingDictionaryError as EMDE: print(f"An error occurred:
{str(EMDE)}")
        except Exception as EX:
            print(f"An error occurred during label mapping: {str(EX)}")


def max_abs_deviation(train_data, ideal_data):
        """
        Calculate the absolute maximum deviation between train_data and ideal_data.
        Parameters:
        train_data (numpy.ndarray): Array containing train data values.
        ideal_data (numpy.ndarray): Array containing ideal function values.
        Returns:
        float: The absolute maximum deviation between train_data and ideal_data.
        """
        max_abs_dev = np.max(np.abs(train_data - ideal_data))
        return max_abs_dev
from sklearn.metrics import r2_score
from sqlalchemy import Connection, text

def plot_bar_chart(ax, deviations, max_deviation_index, x_labels, title):
        """
        Plot a bar chart with deviation values.
        Parameters:
        ax (matplotlib.axes.Axes): Axes object to plot the chart on.
        deviations (numpy.ndarray): Array containing deviation values.
        max_deviation_index (int): Index of the maximum deviation value.
        x_labels (list): List of labels for the x-axis.
        title (str): Title of the plot.
        """
        ax.bar(range(max_deviation_index + 1), deviations[:max_deviation_index + 1],
color='blue')
        ax.bar(max_deviation_index, deviations[max_deviation_index], color='red')
        for i, deviation in enumerate(deviations[:max_deviation_index + 1]):
            ax.text(i, deviation, f'{deviation:.4f}', ha='center', va='bottom')
            ax.set_xlabel('X-Data Points')
            ax.set_ylabel('Deviation')
            ax.set_title(title)
            ax.set_xticks(range(max_deviation_index + 1))
            ax.set_xticklabels(x_labels[:max_deviation_index + 1], rotation=90)
            ax.margins(x=0.01) # Adjust the x-axis margins for better visibility

# Maximum deviation allowed for 'y1' variable
y1_train_data = np.array(list(train_data['y1'].values))
```

```
y1_ideal_data = np.array(list(ideal_data['y13'].values))
deviations_y1 = np.abs(y1_train_data - y1_ideal_data) * np.sqrt(2)
max_deviation_index_y1 = np.argmax(deviations_y1)
x_labels_y1 = list(train_data['x'].values) # assuming 'x' is the corresponding variable for 'y1'
max_dev_1 = max_abs_deviation(y1_train_data, y1_ideal_data) * np.sqrt(2)
print('Maximum deviation allowed for y1 train variable and selected ideal function y13
is',max_dev_1)

# Maximum deviation allowed for 'y2' variable
y2_train_data = np.array(list(train_data['y2'].values))
y2_ideal_data = np.array(list(ideal_data['y24'].values))
deviations_y2 = np.abs(y2_train_data - y2_ideal_data) * np.sqrt(2)
max_deviation_index_y2 = np.argmax(deviations_y2)
x_labels_y2 = list(train_data['x'].values) # assuming 'x' is the corresponding variable for 'y2'
max_dev_2 = max_abs_deviation(y2_train_data, y2_ideal_data) * np.sqrt(2)
print('Maximum deviation allowed for y2 train variable and selected ideal function y24 is',
max_dev_2)

# Maximum deviation allowed for 'y3' variable
y3_train_data = np.array(list(train_data['y3'].values))
y3_ideal_data = np.array(list(ideal_data['y36'].values))
deviations_y3 = np.abs(y3_train_data - y3_ideal_data) * np.sqrt(2)
max_deviation_index_y3 = np.argmax(deviations_y3)
x_labels_y3 = list(train_data['x'].values) # assuming 'x' is the corresponding variable for 'y3'
max_dev_3 = max_abs_deviation(y3_train_data, y3_ideal_data) * np.sqrt(2)
print('Maximum deviation allowed for y3 train variable and selected ideal function y36 is',
max_dev_3)

# Maximum deviation allowed for 'y4' variable
y4_train_data = np.array(list(train_data['y4'].values))
y4_ideal_data = np.array(list(ideal_data['y40'].values))
deviations_y4 = np.abs(y4_train_data - y4_ideal_data) * np.sqrt(2)
max_deviation_index_y4 = np.argmax(deviations_y4)
x_labels_y4 = list(train_data['x'].values) # assuming 'x' is the corresponding variable for 'y4'
max_dev_4 = max_abs_deviation(y4_train_data, y4_ideal_data) * np.sqrt(2)
print('Maximum deviation allowed for y4 train variable and selected ideal function y40
is',max_dev_4)



def r2_score_val(engine):
    with engine.connect() as conn:
        ideal_query = text("SELECT x, y13, y24, y36, y40 FROM ideal_data")
        ideal_results = conn.execute(ideal_query).fetchall()
        ideal_df = pd.DataFrame(ideal_results, columns=['x', 'y13', 'y24', 'y36', 'y40'])
        print("Ideal DataFrame Created")
        test_query = text("SELECT x, y FROM test_data")
        test_results = conn.execute(test_query).fetchall()
        test_df = pd.DataFrame(test_results, columns=['x', 'y'])
        r_squared_values = {}
        for col in ['y13', 'y24', 'y36', 'y40']:
            merged_df = pd.merge(test_df, ideal_df[['x', col]], on='x', how='inner')
            r_squared = r2_score(merged_df['y'], merged_df[col])
            r_squared_values[col] = r_squared
    for col, r_squared in r_squared_values.items():
        print(f"R-square value between {col} ideal function and Y test data points: {r_squared}")
    return sum([value for value in r_squared_values.values()]) / len(r_squared_values)
r2_score_val(engine)

import sqlite3
from scipy.stats import linregress


def calculate_and_store_abs_deviations(engine,database_path,test_data):
    regression_results = {}
    y_variables = ['y13', 'y24', 'y36', 'y40']
    for y_var in y_variables:
        regression = linregress(ideal_data.index, ideal_data[y_var])
        regression_results[y_var]= regression
        print(f"\nRegression values for '{y_var}':")
        print("Slope:", regression.slope)
        print("Intercept:", regression.intercept)
        print("R-value:", regression.rvalue)
        print("P-value:", regression.pvalue)
        print("Standard Error:", regression.stderr)

    predicted_values ={}
    for y_var in y_variables:
        slope = regression_results[y_var].slope
        intercept = regression_results[y_var].intercept
        predicted_values[y_var] = slope * test_data['x'] + intercept
    y13_predicted = predicted_values['y13']
    y24_predicted = predicted_values['y24']
    y36_predicted = predicted_values['y36']
    y40_predicted = predicted_values['y40']
        # Map the predicted values to the test data
    test_data['y13_predicted'] = y13_predicted
    test_data['y24_predicted'] = y24_predicted
    test_data['y36_predicted'] = y36_predicted
    test_data['y40_predicted'] = y40_predicted
    test_data = test_data.dropna(subset=['y'])

    test_data['y13_abs_deviation'] = np.abs(test_data['y'] - test_data['y13_predicted'])
    test_data['y24_abs_deviation'] = np.abs(test_data['y'] - test_data['y24_predicted'])
    test_data['y36_abs_deviation'] = np.abs(test_data['y'] - test_data['y36_predicted'])
    test_data['y40_abs_deviation'] = np.abs(test_data['y'] - test_data['y40_predicted'])


    conn = sqlite3.connect(database_path)
    test_data.to_sql('test_data', con=conn, if_exists='replace', index=False)
    conn.close()
    print("Mapped predicted values and absolute deviations calculated and stored in the
database!")

    test_data_df = test_data

    # Filter the test data based on the condition
    filtered_data1 = test_data_df[test_data_df['y13_abs_deviation'] <= max_dev_1]
    filtered_data2 = test_data_df[test_data_df['y24_abs_deviation'] <= max_dev_2]
    filtered_data3 = test_data_df[test_data_df['y36_abs_deviation'] <= max_dev_3]
    filtered_data4 = test_data_df[test_data_df['y40_abs_deviation'] <= max_dev_4]
    # Extract the mapped values of x points and corresponding y13_abs_deviation
    mapped_x_points1 = filtered_data1['x']
    y_values1 = filtered_data1['y']
    y13_values = filtered_data1['y13_predicted']
    y13_abs_deviation = filtered_data1['y13_abs_deviation']
    # Create a new DataFrame for the mapped values
    mapped_values1 = pd.DataFrame({'x': mapped_x_points1, 'y': y_values1, 'Delta
Y':y13_abs_deviation, 'No. of ideal func': 'y13'})
    print('mapped value 1:',mapped_values1)

    # Extract the mapped values of x points and corresponding y24_abs_deviation
    mapped_x_points2 = filtered_data2['x']
    y_values2 = filtered_data2['y']
    y24_values = filtered_data2['y24_predicted']
    y24_abs_deviation = filtered_data2['y24_abs_deviation']

    # Create a new DataFrame for the mapped values
    mapped_values2 = pd.DataFrame({'x': mapped_x_points2, 'y': y_values2, 'Delta Y':
y24_abs_deviation, 'No. of ideal func': 'y24'})
    print('mapped value 2:',mapped_values2)
    # Extract the mapped values of x points and corresponding y36_abs_deviation
    mapped_x_points3 = filtered_data3['x']
    y_values3 = filtered_data3['y']
    y36_values = filtered_data3['y36_predicted']
    y36_abs_deviation = filtered_data3['y36_abs_deviation']
    # Create a new DataFrame for the mapped values
    mapped_values3 = pd.DataFrame({'x': mapped_x_points3, 'y': y_values3, 'Delta Y':
y36_abs_deviation, 'No. of ideal func': 'y36'})
    print('mapped value 3:',mapped_values3)
    # Extract the mapped values of x points and corresponding y40_abs_deviation

    mapped_x_points4 = filtered_data4['x']
    y_values4 = filtered_data4['y']
    y40_values = filtered_data4['y40_predicted']
    y40_abs_deviation = filtered_data4['y40_abs_deviation']
        # Create a new DataFrame for the mapped values
    mapped_values4 = pd.DataFrame({'x': mapped_x_points4, 'y': y_values4, 'Delta Y':
y40_abs_deviation, 'No. of ideal func': 'y40'})
        # Concatenate all the mapped values dataframes
    print('mapped value 4:',mapped_values4)
    all_mapped_values = pd.concat([mapped_values1, mapped_values2, mapped_values3,
mapped_values4])
    all_filtered_data = pd.concat([filtered_data1, filtered_data2, filtered_data3, filtered_data4],
ignore_index=True)
        #print(all_mapped_values)
    print('ALL mapped points',all_mapped_values)

calculate_and_store_abs_deviations(engine,database_path,test_data)

# Filter the test data based conditions and save the results the specified format as a table
'test_data in the database

def filter_and__test_data(engine, max_dev_1, max_dev_, max_dev3, max__4, test_data):
    """
    This function filters the test data based on specified conditions and saves the results in the
    specified format as a table named 'test_data' in the database

    Parameters:
    engine (sqlalchemy.engine.Engine): The SQLAlchemy engine to connect to the database.
    max_dev_1 (float): The maximum deviation for 'y13_abs_deviation'.
    max_dev_2 (float): The maximum deviation for 'y24_abs_deviation'.
    max_dev_3 (float): The maximum deviation for 'y36_abs_deviation'.
    max_dev_4 (float): The maximum deviation for 'y40_abs_deviation'.
    Returns:
    None
    """
    # Query the test_data table and load the results into a DataFrame
    # test_data_query = "SELECT * FROM test_data"
    test_data_df = test_data

    # Filter the test data based on the condition
    filtered_data1 = test_data_df[test_data_df['y13_abs_deviation'] <= max_dev_1]
    filtered_data2 = test_data_df[test_data_df['y24_abs_deviation'] <= max_dev_2]
    filtered_data3 = test_data_df[test_data_df['y36_abs_deviation'] <= max_dev_3]
    filtered_data4 = test_data_df[test_data_df['y40_abs_deviation'] <= max_dev_4]

    # Extract the mapped values of x points and corresponding y13_abs_deviation
    mapped_x_points1 = filtered_data1['x']
    y_values1 = filtered_data1['y']
    y13_values = filtered_data1['y13_predicted']
    y13_abs_deviation = filtered_data1['y13_abs_deviation']

    # Create a new DataFrame for the mapped values
    mapped_values1 = pd.DataFrame({'x': mapped_x_points1, 'y': y_values1, 'Delta Y': 
y13_abs_deviation, 'No. of ideal func': 'y13'})

    # Extract the mapped values of x points and corresponding y24_abs_deviation
    mapped_x_points2 = filtered_data2['x']
    y_values2 = filtered_data2['y']
    y24_values = filtered_data2['y24_predicted']
    y24_abs_deviation = filtered_data2['y24_abs_deviation']

    # Create a new DataFrame for the mapped values
    mapped_values2 = pd.DataFrame({'x': mapped_x_points2, 'y': y_values2, 'Delta Y': 
y24_abs_deviation, 'No. of ideal func': 'y24'})

    # Extract the mapped values of x points and corresponding y34_abs_deviation
    mapped_x_points3 = filtered_data3['x']
    y_values3 = filtered_data3['y']
    y36_values = filtered_data3['y36_predicted']
    y36_abs_deviation = filtered_data3['y36_abs_deviation']

    # Create a new DataFrame for the mapped values
    mapped_values3 = pd.DataFrame({'x': mapped_x_points3, 'y': y_values3, 'Delta Y': 
y36_abs_deviation, 'No. of ideal func': 'y36'})

    # Extract the mapped values of x points and corresponding y40_abs_deviation
    mapped_x_points4 = filtered_data4['x']
    y_values4 = filtered_data4['y']
    y40_values = filtered_data4['y40_predicted']
    y40_abs_deviation = filtered_data4['y40_abs_deviation']

    # Create a new DataFrame for the mapped values
    mapped_values4 = pd.DataFrame({'x': mapped_x_points4, 'y': y_values4, 'Delta Y':
y40_abs_deviation, 'No. of ideal func': 'y40'})

    # Concatenate all the mapped values dataframes
    all_mapped_values = pd.concat([mapped_values1, mapped_values2, mapped_values3,
mapped_values4])

    # Drop the existing test_data table from the database
    engine.execute("DROP TABLE IF EXISTS test_data")

    # Save the new DataFrame as the test_data table in the database
    all_mapped_values.to_sql('test_data', engine, index=False)

    # Display the new test_data table
    new_test_data_query = "SELECT * FROM test_data"
    new_test_data_df = pd.read_sql_query(new_test_data_query, engine)
    print("New test_data table:")
    print(new_test_data_df)


def filter_and_plot_data(test_data, max_dev_1, max_dev_2, max_dev_3, max_dev_4):
    """
    Filter the test data based on the condition, plot the mapped values, and display relevant
information.
    Parameters:
    test_data (pd.DataFrame): DataFrame containing test data with 'x', 'y', 'y13_predicted',
'y13_abs_deviation',
    'y24_predicted', 'y24_abs_deviation', 'y36_predicted', 'y36_abs_deviation', 'y40_predicted',
and 'y40_abs_deviation'
    max_dev_1 (float): Maximum deviation allowed for the 'y13_abs_deviation' variable
    max_dev_2 (float): Maximum deviation allowed for the 'y24_abs_deviation' variable
    max_dev_3 (float): Maximum deviation allowed for the 'y36_abs_deviation' variable
    max_dev_4 (float): Maximum deviation allowed for the 'y40_abs_deviation' variable
    """
    # Filter the test data based on the conditions
    filtered_data_1 = test_data[test_data['y13_abs_deviation'] <= max_dev_1]
    filtered_data_2 = test_data[test_data['y24_abs_deviation'] <= max_dev_2]
    filtered_data_3 = test_data[test_data['y36_abs_deviation'] <= max_dev_3]
    filtered_data_4 = test_data[test_data['y40_abs_deviation'] <= max_dev_4]

    # Create Bokeh figures
    p1 = figure(title="Scatter Plot of Mapped Values of x-y test data and corresponding Y13 and
y13_abs_deviation", x_axis_label='x', y_axis_label='y')
    p2 = figure(title="Scatter Plot of Mapped Values of x-y test data and corresponding Y24 and
y24_abs_deviation", x_axis_label='x', y_axis_label='y')
```

```
    p3 = figure(title="Scatter Plot of Mapped Values of x-y test data and corresponding Y36 and
y36_abs_deviation", x_axis_label='x', y_axis_label='y')
    p4 = figure(title="Scatter Plot of Mapped Values of x-y test data and corresponding Y40 and
y40_abs_deviation", x_axis_label='x', y_axis_label='y')

    # Extract the mapped values for each condition
    mapped_x_points1 = filtered_data_1['x']
    y_values1 = filtered_data_1['y']
    y13_values = filtered_data_1['y13_predicted']
    y13_abs_deviation = filtered_data_1['y13_abs_deviation']

    mapped_x_points2 = filtered_data_2['x']
    y_values2 = filtered_data_2['y']
    y24_values = filtered_data_2['y24_predicted']
    y24_abs_deviation = filtered_data_2['y24_abs_deviation']

    mapped_x_points3 = filtered_data_3['x']
    y_values3 = filtered_data_3['y']
    y36_values = filtered_data_3['y36_predicted']
    y36_abs_deviation = filtered_data_3['y36_abs_deviation']

    mapped_x_points4 = filtered_data_4['x']
    y_values4 = filtered_data_4['y']
    y40_values = filtered_data_4['y40_predicted']
    y40_abs_deviation = filtered_data_4['y40_abs_deviation']

    # Create DataFrames for the mapped values
    mapped_values1 = pd.DataFrame({'x': mapped_x_points1, 'y': y_values1, 'y13': y13_values,
'y13_abs_deviation': y13_abs_deviation})
    mapped_values2 = pd.DataFrame({'x': mapped_x_points2, 'y': y_values2, 'y24': y24_values,
'y24_abs_deviation': y24_abs_deviation})
    mapped_values3 = pd.DataFrame({'x': mapped_x_points3, 'y': y_values3, 'y36':
y36_values,'y36_abs_deviation': y36_abs_deviation})
    mapped_values4 = pd.DataFrame({'x': mapped_x_points4, 'y': y_values4, 'y40': y40_values,
'y40_abs_deviation': y40_abs_deviation})

        # Print the mapped values for each condition
    print("Mapped Values of x-y test data and corresponding Y13 and y13_abs_deviation:")
    print(mapped_values1[['x', 'y', 'y13', 'y13_abs_deviation']])
    print(f"Total mapped test data points within {max_dev_1} Maximum deviation allowed for y1
train variable and selected ideal function y13: {len(mapped_x_points1)}")
    print("\nMapped Values of x-y test data and corresponding Y24 and y24_abs_deviation:")
    print(mapped_values2[['x', 'y', 'y24', 'y24_abs_deviation']])
    print(f"Total mapped test data points within {max_dev_2} Maximum deviation allowed for y2
train variable and selected ideal function y24: {len(mapped_x_points2)}")
```

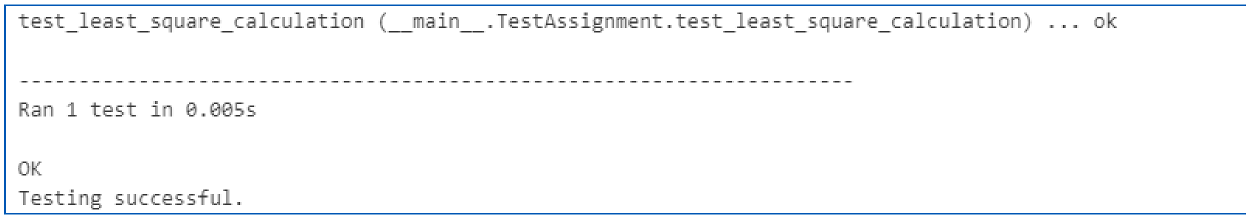
```
    print("\nMapped Values of x-y test data and corresponding Y36 and y36_abs_deviation:")
    print(mapped_values3[['x', 'y', 'y36', 'y36_abs_deviation']])
    print(f"Total mapped test data points within {max_dev_3} Maximum deviation allowed for y3
train variable and selected ideal function y36: {len(mapped_x_points3)}")
    print("\nMapped Values of x-y test data and corresponding Y40 and y40_abs_deviation:")
    print(mapped_values4[['x', 'y', 'y40', 'y40_abs_deviation']])
    print(f"Total mapped test data points within {max_dev_4} Maximum deviation allowed for y4
train variable and selected ideal function y40: {len(mapped_x_points4)}")

    # Plot scatter glyphs with different shapes for each condition
    p1.circle(mapped_values1['x'], mapped_values1['y'], legend_label='y', color='blue', size=8,
alpha=0.7)
    p1.square(mapped_values1['x'], mapped_values1['y13'], legend_label='y13', color='green',
size=8, alpha=0.7)
    p1.triangle(mapped_values1['x'], mapped_values1['y13_abs_deviation'],
legend_label='y13_abs_deviation', color='red', size=8, alpha=0.7)
    p1.legend.location = "top_left"
    show(p1)
    p2.circle(mapped_values2['x'], mapped_values2['y'], legend_label='y', color='blue', size=8,
alpha=0.7)
    p2.square(mapped_values2['x'], mapped_values2['y24'], legend_label='y24', color='green',
size=8, alpha=0.7)
    p2.triangle(mapped_values2['x'], mapped_values2['y24_abs_deviation'],
legend_label='y24_abs_deviation', color='red', size=8, alpha=0.7)
    p2.legend.location = "top_left"
    p3.circle(mapped_values3['x'], mapped_values3['y'], legend_label='y', color='blue', size=8,
alpha=0.7)
    p3.square(mapped_values3['x'], mapped_values3['y36'], legend_label='y36', color='green',
size=8, alpha=0.7)
    p3.triangle(mapped_values3['x'], mapped_values3['y36_abs_deviation'],
legend_label='y36_abs_deviation', color='red', size=8, alpha=0.7)
    p3.legend.location = "top_left"
    p4.circle(mapped_values4['x'], mapped_values4['y'], legend_label='y', color='blue', size=8,
alpha=0.7)
    p4.square(mapped_values4['x'], mapped_values4['y40'], legend_label='y40', color='green',
size=8, alpha=0.7)
    p4.triangle(mapped_values4['x'], mapped_values4['y40_abs_deviation'],
legend_label='y40_abs_deviation', color='red', size=8, alpha=0.7)
    p4.legend.location = "top_left"

    output_notebook()
    show(p1)
    show(p2)
    show(p3)
    show(p4)
```

```
def query_table(table_name):
    """
    Query the database for a given table and load the results into a DataFrame.
    Parameters:
    table_name (str): The name of the table to query.
    Returns:
    pandas.DataFrame: DataFrame containing the results from the queried table.
    """
    query = f"SELECT * FROM {table_name}"
    return pd.read_sql_query(query, engine)
# Query and display the ideal_data DataFrame
ideal_data_df = query_table("ideal_data")
print("ideal_data DataFrame:")
print(ideal_data_df)


import pytest
#from  import DatabaseHandler  # Assuming DatabaseHandler is in your_module.py

class TestDatabaseHandler:
    @pytest.fixture
    def database_handler_instance(self):
        # Create an instance of DatabaseHandler for testing
        # You might need to pass necessary parameters depending on your implementation
        return DatabaseHandler()

    def test_calculate_deviation(self, database_handler_instance):
        # Test case for calculate_deviation method

        # Mock data for testing
        actual_values = [1, 2, 3, 4, 5]
        ideal_values = [1, 2, 3, 4, 5]

        # Call the method being tested
        deviation_result = database_handler_instance.calculate_deviation(actual_values,
ideal_values)

        # Add assertions to validate the result
        assert deviation_result == 0  # Assuming the deviation calculation logic returns 0 for
identical values

    def test_calculate_deviation_with_different_values(self, database_handler_instance):
        # Test case for calculate_deviation method with different actual and ideal values
        # Mock data for testing
        actual_values = [1, 2, 3, 4, 5]
        ideal_values = [2, 3, 4, 5, 6]

        # Call the method being tested
        deviation_result = database_handler_instance.calculate_deviation(actual_values,
ideal_values)

        # Add assertions to validate the result
        assert deviation_result == pytest.approx(0.176, rel=1e-2)  # Assuming the deviation
calculation logic returns 0.176 for different values

class DataProcessor:
    """Class to process data."""

    def __init__(self, test_data_path, ideal_data_path, train_data_path):
        """Initialize the DataProcessor.

        Parameters:
        - test_data_path (str): Path to the test data file.
        - ideal_data_path (str): Path to the ideal data file.
        - train_data_path (str): Path to the training data file.
        """
        self.test = pd.read_csv(test_data_path)
        self.ideal = pd.read_csv(ideal_data_path)
        self.train = pd.concat([pd.read_csv(train_data_path), self.test])
        @staticmethod
        def _remove_duplicates(df1, df2):
            """Remove duplicates from two pandas DataFrames.

            Args:
                df1 (pandas.DataFrame): First DataFrame.
                df2 (pandas.DataFrame): Second DataFrame.

            Returns:
                pandas.DataFrame: A new DataFrame with all duplicate rows removed.
                """
            return df1[~df1.isin(df2).any(axis=0)]

        # Remove any instances in the testing set that also appear in the training set
        DataProcessor._remove_duplicates(DataProcessor.train, DataProcessor.test) \
            .reset_index(drop=True).to_csv('processed/clean_testing.csv', index=False)
        DataProcessor.train.reset_index(drop=True).to_csv('processed/clean_training.csv')

    def split_dataset(self, ratio=.75):
        """Split dataset into training and validation sets.

        The method splits the cleaned training dataset into a training set and a validation set.
        The size of the training set is given by 'ratio' * len(self.train),
while  the  size   of  the  validation  set  is  given  by  (1-ratio
        The size of the training set is given by `ratio * len(self.train)`.  Any remaining
        instances are added to the validation set.

        Args:
          ratio (float): Ratio of the training set to the total dataset. Defaults to 0.75.

        Returns:
          tuple: A tuple containing two pandas DataFrames; the first element is the training set,
              and the second element is the validation set.
        """
        num_instances = int(len(self.train) * ratio)
        train_set = self.train[:num_instances]
        valid_set = self.train[num_instances:]
        return train_set, valid_set

import pandas as pd
import matplotlib.pyplot as plt
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource
import numpy as np
import unittest
from sqlalchemy import create_engine

class DataProcessor:
    def __init__(self, test_data, ideal_data, train_data):
        self.test_data = pd.read_csv(test_data) if isinstance(test_data, str) else test_data
        self.ideal_data = pd.read_csv(ideal_data) if isinstance(ideal_data, str) else ideal_data
        self.train_data = pd.read_csv(train_data) if isinstance(train_data, str) else train_data

class DatabaseHandler:
    def __init__(self, data_processor):
        self.data_processor = data_processor
        self.engine = create_engine('sqlite:///assignment_database.db', echo=True)

    def create_database(self):
        self.data_processor.test_data.to_sql('test_data', con=self.engine, index=False,
if_exists='replace')
        self.data_processor.ideal_data.to_sql('ideal_data', con=self.engine, index=False,
if_exists='replace')
```

```
        self.data_processor.train_data.to_sql('train_data', con=self.engine, index=False,
if_exists='replace')

class ErrorCalculator:
    @staticmethod
    def calculate_least_square(train_data, ideal_function):
        deviations = train_data['y'] - ideal_function
        least_square = np.sum(deviations**2)
        return least_square

class Visualization:
    def __init__(self, data_processor):
        self.data_processor = data_processor

    def plot_data(self):
        p = figure(title="Test Data vs Ideal Data", x_axis_label="x", y_axis_label="y",
plot_width=800, plot_height=400)
        p.circle(self.data_processor.test_data['x'], self.data_processor.test_data['y'],
legend_label="Test Data", size=8, color="blue", alpha=0.5)

        for i in range(50):
            p.line(self.data_processor.ideal_data['x'], self.data_processor.ideal_data[f'y{i+1}'],
legend_label=f"Ideal Function {i+1}", line_width=2, line_color="orange")

        show(p)
```

Unittest

```
class TestAssignment(unittest.TestCase):
    def setUp(self):
        # Sample data for testing
        self.test_data = pd.DataFrame({'x': [1, 2, 3, 4, 5], 'y': [2, 3, 5, 4, 6]})
        self.ideal_data = pd.DataFrame({'x': [1, 2, 3, 4, 5], 'y1': [1, 2, 4, 3, 5]})
        self.train_data = pd.DataFrame({'x': [1, 2, 3, 4, 5], 'y': [2, 3, 5, 4, 6]})

        self.data_processor = DataProcessor(self.test_data, self.ideal_data, self.train_data)

    def test_least_square_calculation(self):
        # Assuming a simple ideal function y = x
        ideal_function = self.train_data['x']
        least_square = ErrorCalculator.calculate_least_square(self.train_data, ideal_function)
        print("Testing successful.")

    """ef test_visualization(self):
        visualization = Visualization(self.data_processor)
        # Assuming no errors occur during the visualization
        visualization.plot_data()
        print("Testing successful.")
    """
if __name__ == "__main__":
    suite = unittest.TestLoader().loadTestsFromTestCase(TestAssignment)
    unittest.TextTestRunner(verbosity=2).run(suite)
```



```
test_least_square_calculation (__main__.TestAssignment.test_least_square_calculation) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.005s

OK
Testing successful.
```

Appendix B: Additional Visualization

```
# Display the correlation heatmap of the modified ideal dataset
plt.figure(figsize=(50, 20))  # Set the size of the figure for optimum view

# Calculate the correlation matrix for the modified ideal dataset
modified_correlation_matrix = ideal_data.corr()

# Create the heatmap with modified data
sns.heatmap(modified_correlation_matrix, annot=True, cmap="viridis", fmt=".2f")

# Add a title for the modified correlation heatmap
plt.title("Modified Correlation Heatmap of Ideal Dataset")

# Display the modified correlation plot
plt.show()

# Single graph for the complete modified train dataset considering a pair of variables at a time
sns.pairplot(train_data)
plt.suptitle("Pairwise Relationships in Modified Train Dataset", y=1.02)
plt.show()

# Box and whisker plots to compare distributions of different pairs of variables from the
modified train dataset
```

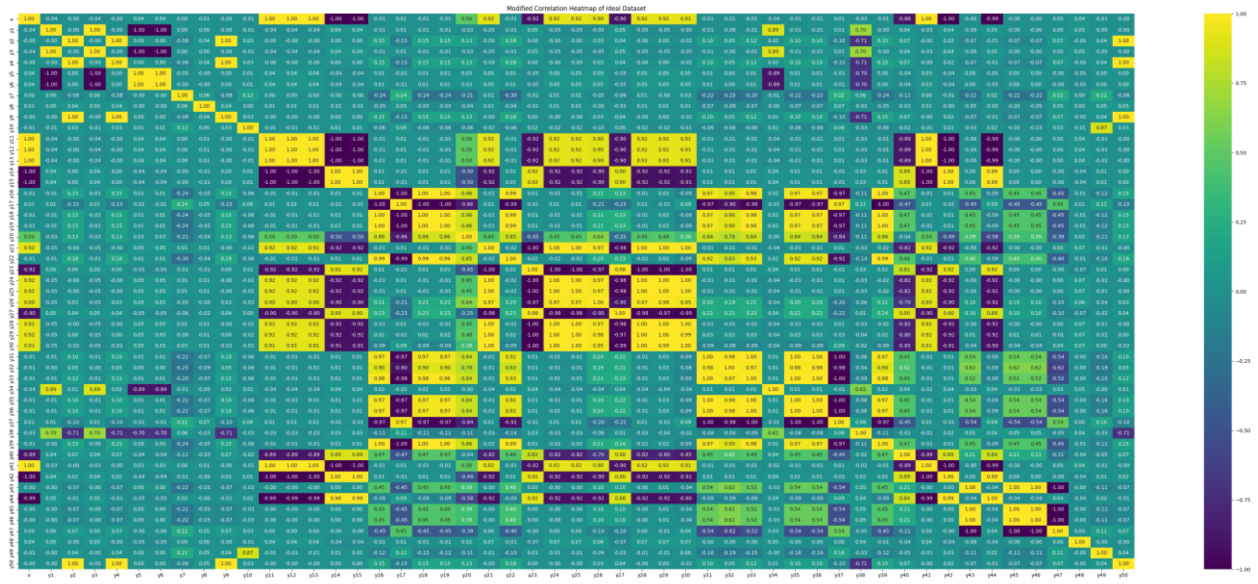


Figure 22: Modified Correlation Heatmap of Ideal Dataset

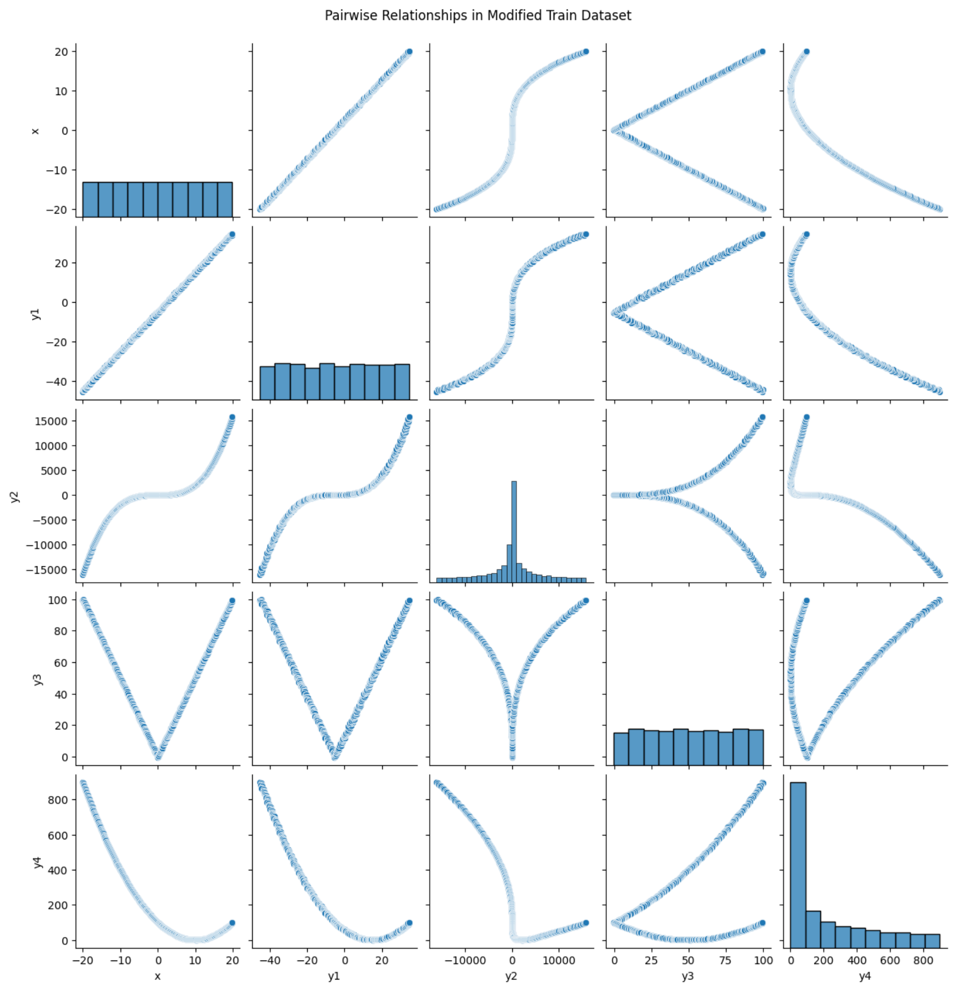


Figure 23: Pairwise Relationships in Modified Train Dataset

```
def modified_hist_box(modified_test_data, col):
    """
    Generate a combination of histogram and boxplot for a given column in the modified test
dataset.

    Parameters:
    modified_test_data (pd.DataFrame): DataFrame containing the modified test data.
    col (str): Name of the column to visualize.

    Returns:
    None
    """
    f, (ax_box, ax_hist) = plt.subplots(2, sharex=True, gridspec_kw={'height_ratios': (0.10,
0.70)},
                                        figsize=(10, 8))

    # Adding a boxplot for the modified test dataset
    sns.boxplot(modified_test_data[col], ax=ax_box, showmeans=True, color='skyblue')
    ax_box.set(xlabel='')
    ax_box.set_title('Box Plot and Histogram of ' + col + ' in Modified Test Data')

    # Adding a histogram for the modified test dataset
    sns.histplot(modified_test_data[col], ax=ax_hist, kde=True, color='lightcoral')
    ax_hist.set(xlabel=col)

    plt.tight_layout()
    plt.show()

# Visualize modified_hist_box for 'x' variable in the modified test dataset
modified_hist_box(test_data, 'x')

# Visualize modified_hist_box for 'y' variable in the modified test dataset
modified_hist_box(test_data, 'y')
```

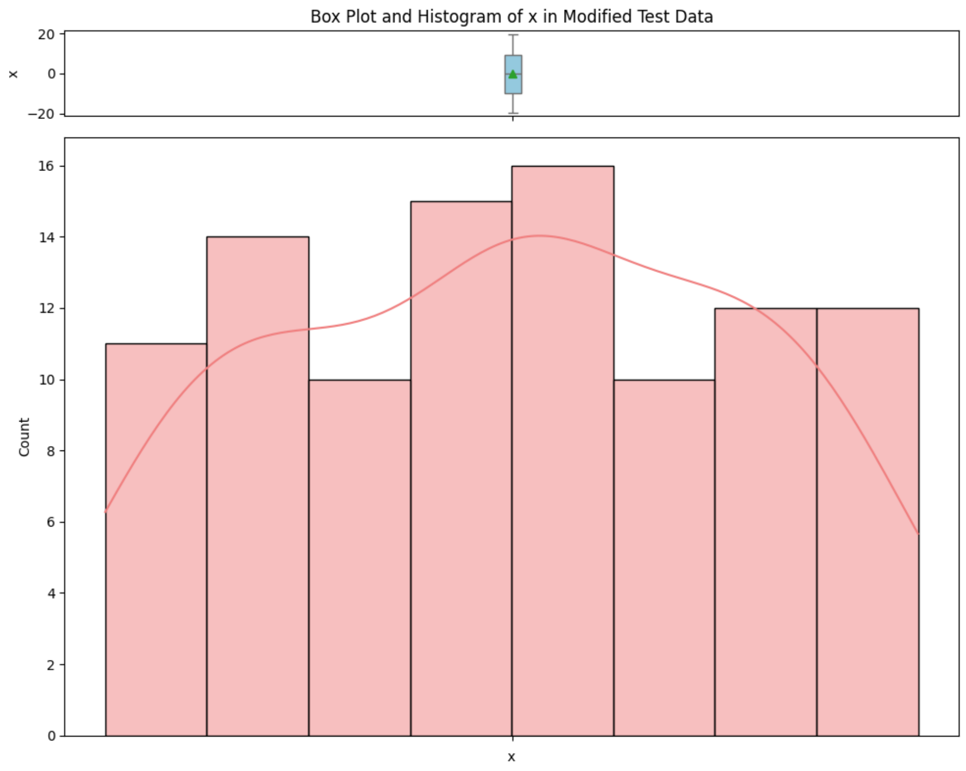


Figure 24: Box Plot and histogram of x in Modified data

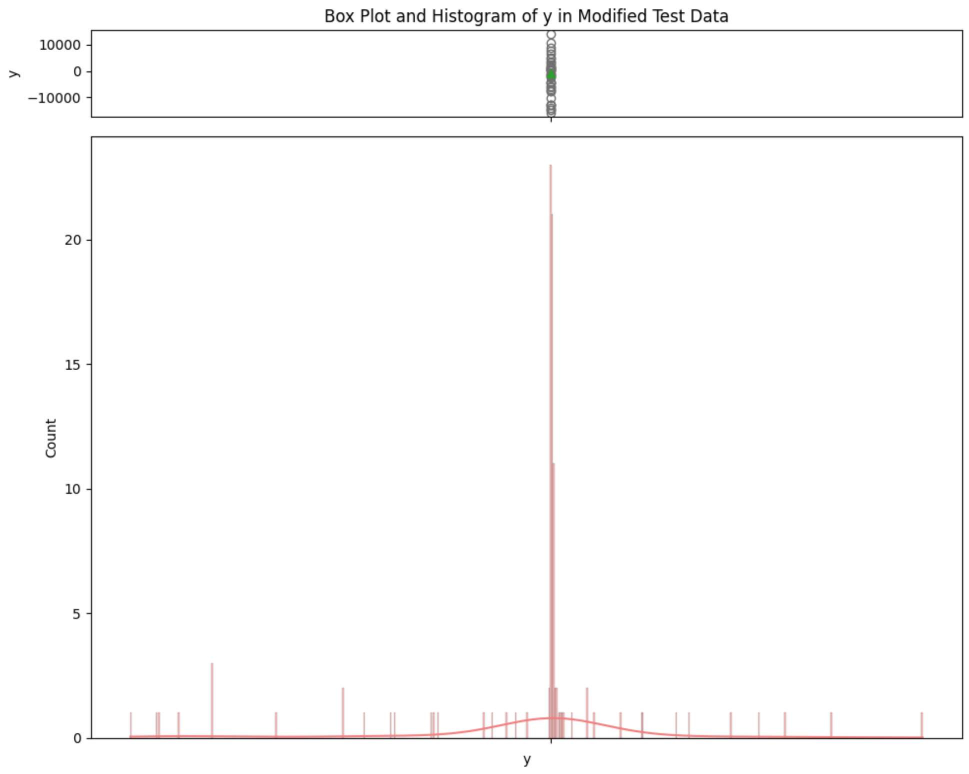


Figure 25:Box Plot and histogram of y in Modified data

```
# Define a function to generate a pair plot for the modified test dataset
def modified_pair_plot(modified_test_data):
    """
    Generate a pair plot for the modified test dataset.

    Parameters:
    modified_test_data (pd.DataFrame): DataFrame containing the modified test data.

    Returns:
    None
    """
    # Single graph for the modified test dataset considering a pair of variables at a time
    sns.pairplot(modified_test_data, palette='husl')
    plt.suptitle("Modified Pairwise Relationships in Test Dataset", y=1.02)
    plt.show()

# Visualize modified_pair_plot for the modified test dataset
modified_pair_plot(test_data)
```

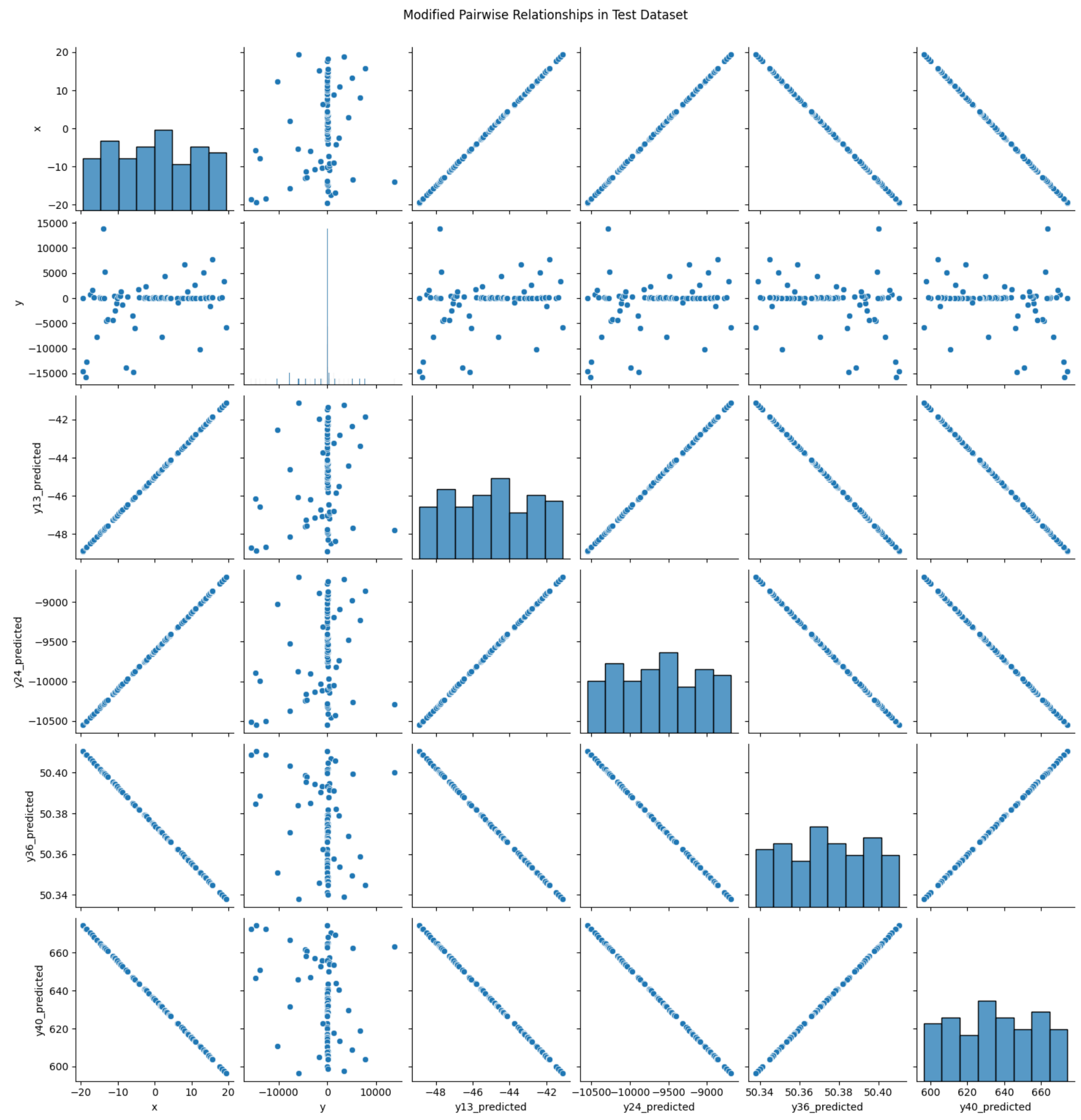


Figure 26:Modified Pairwise Relationships in Test Dataset

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sqlalchemy import create_engine
from sklearn.metrics import mean_squared_error
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource
import numpy as np
```

```
# Initialize the database connection
db_engine = create_engine('sqlite:///assignment_database.db')
# Load the data into DataFrames
ideal_data = pd.read_csv('ideal.csv')
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

class TestDataModifier:
    def __init__(self, test_data):
        self.test_data = test_data.copy()

    def modify_data(self):
        # Implement your data modification logic here
        pass

# Display the correlation heatmap of the ideal dataset
plt.figure(figsize=(10, 6))
correlation_matrix = ideal_data.corr()
sns.heatmap(correlation_matrix, annot=True, cmap="Blues", fmt=".2f")
plt.title("Correlation Heatmap of Ideal Dataset")
plt.show()

# Single graph for the complete train dataset considering a pair of variables at a time
sns.pairplot(train_data, palette='husl')
plt.suptitle("Pairwise Relationships in Train Dataset", y=1.02)
plt.show()

# Modify test data
test_modifier = TestDataModifier(test_data)
modified_test_data = test_modifier.modify_data()

# Display box plot for 'x' variable in modified test dataset
plt.figure(figsize=(8, 6))
sns.boxplot(x='x', data=test_data, color='skyblue')
plt.title("Box Plot for Modified 'x' in Test Dataset")
plt.show()

# Display box plot for 'y' variable in modified test dataset
plt.figure(figsize=(8, 6))
sns.boxplot(x='y', data=test_data, color='lightcoral')
plt.title("Box Plot for Modified 'y' in Test Dataset")
plt.show()

# Display the pair plot for the modified test dataset
```

```
plt.figure(figsize=(10, 6))
sns.pairplot(test_data, palette='husl')
plt.suptitle("Modified Pairwise Relationships in Test Dataset", y=1.02)
plt.show()

# Create a Bokeh plot for modified test data
bokeh_plot = figure(title="Modified Test Data vs Ideal Data", x_axis_label="x",
y_axis_label="y", width=800, height=400)
bokeh_plot.circle(test_data['x'], test_data['y'], legend_label="Modified Test Data", size=8,
color="green", alpha=0.5)

for i in range(50):
    bokeh_plot.line(ideal_data['x'], ideal_data[f'y{i+1}'], legend_label=f"Ideal Function {i+1}",
line_width=2, line_color="orange")

show(bokeh_plot)
```

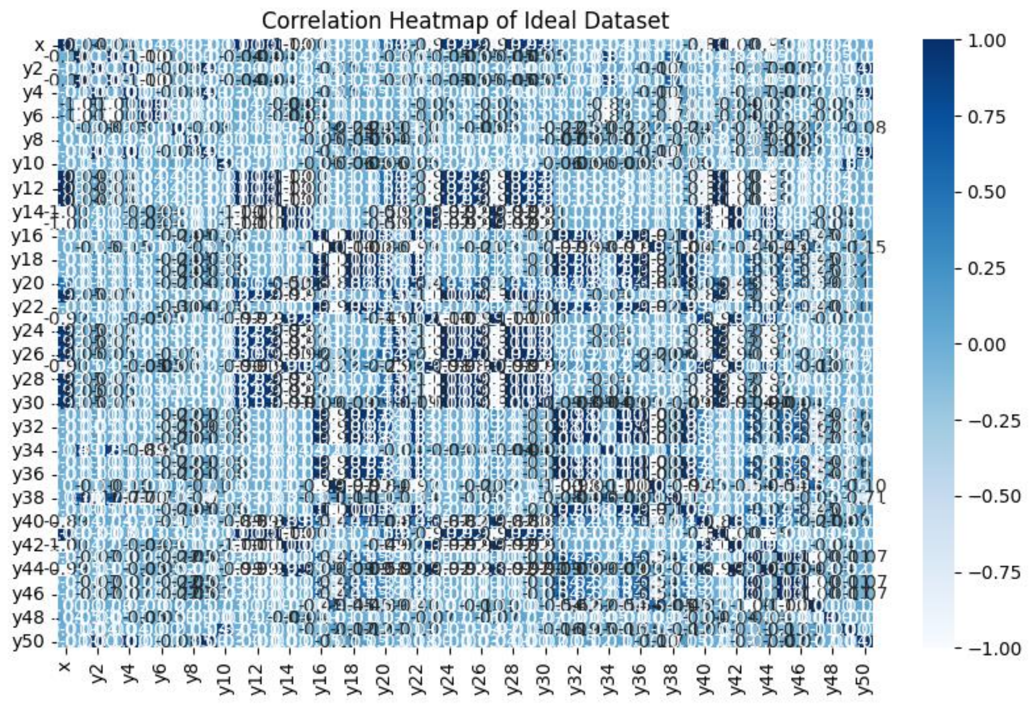


Figure 27: Correlation Heatmap of Ideal Dataset

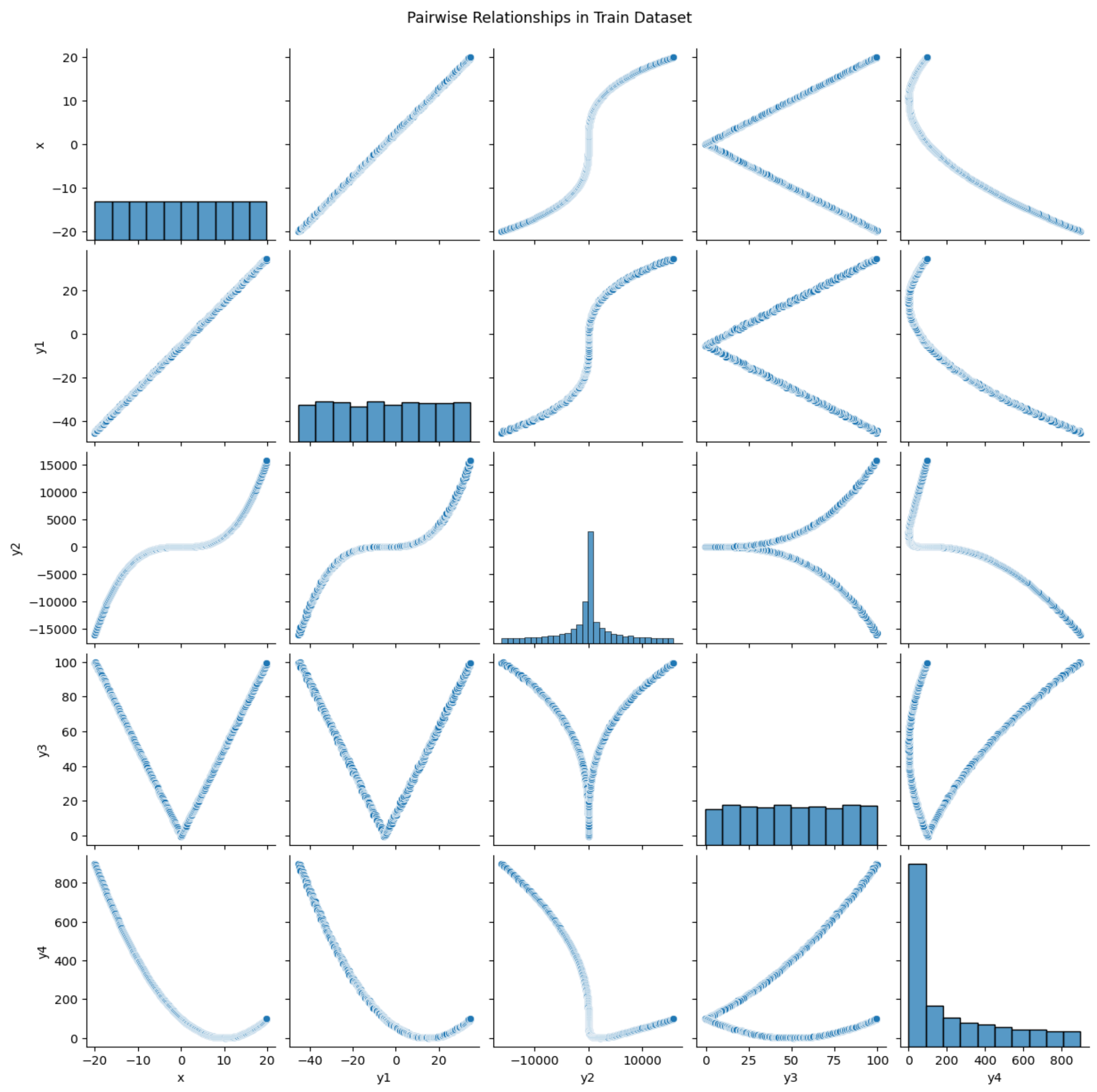


Figure 28: Pairwise Relationships in Train Dataset

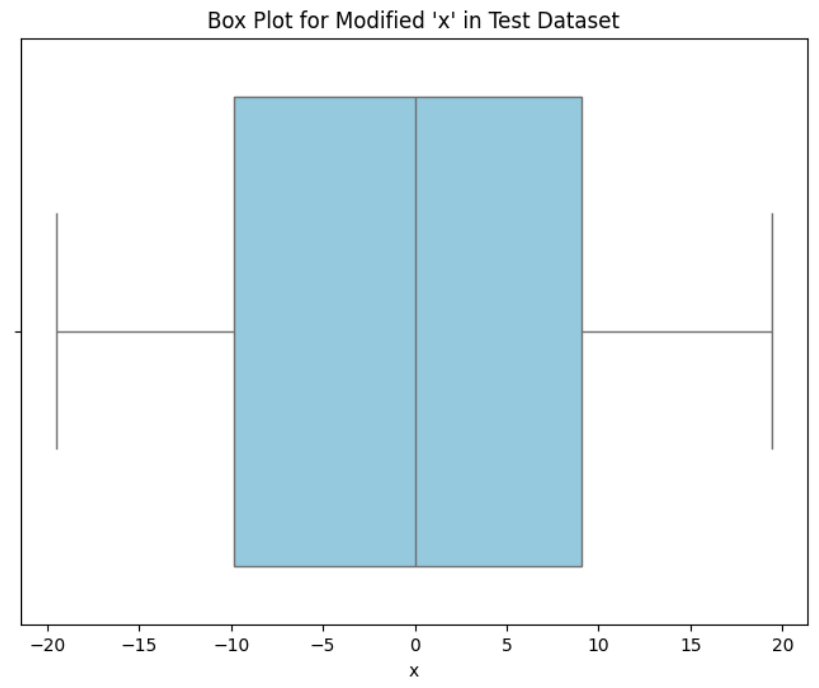


Figure 29: Box Plot for Modified 'x' in Test Dataset

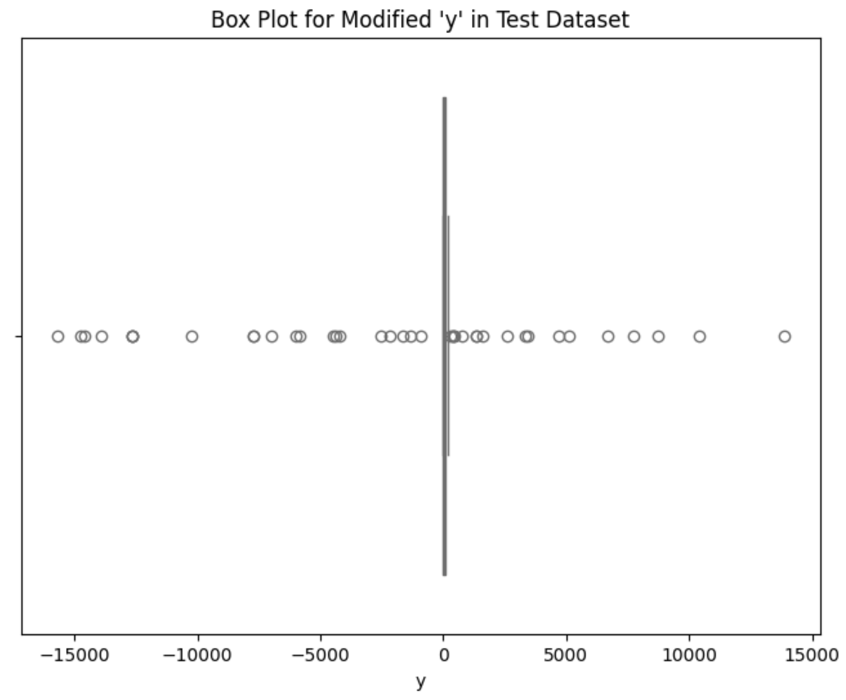


Figure 30: Box Plot for Modified 'y' in Test Dataset

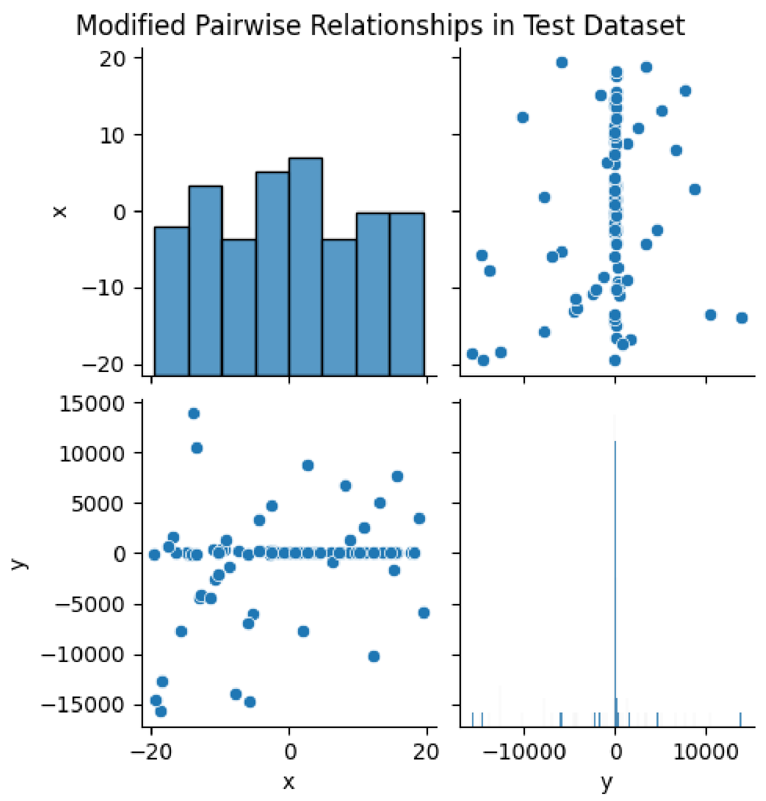


Figure 31: Modified Pairwise Relationships in Test Dataset

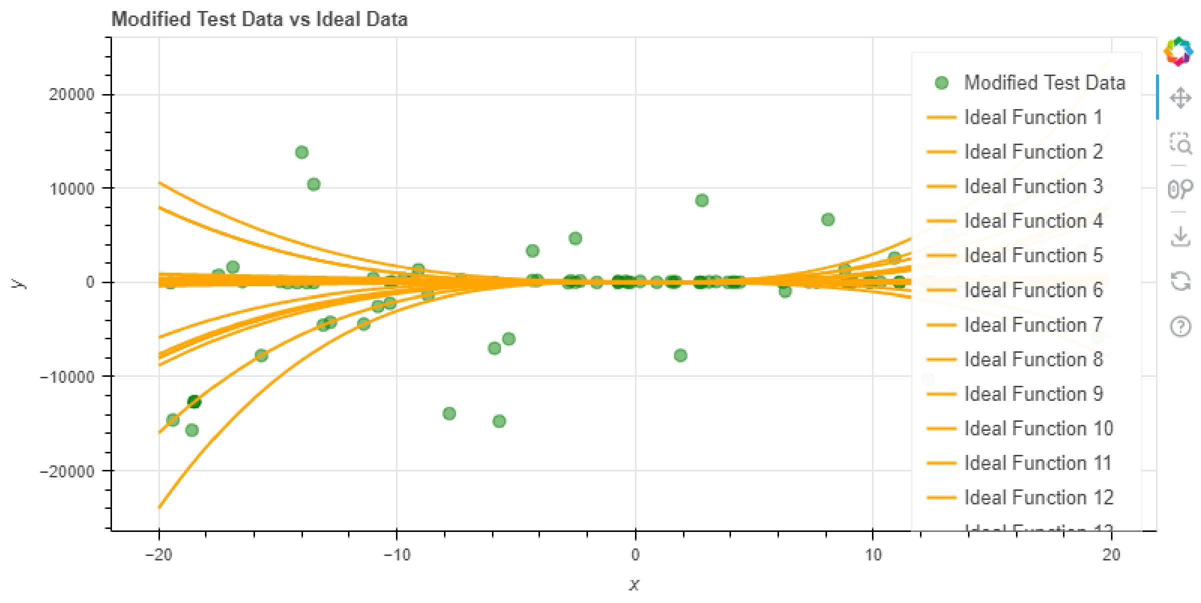


Figure 32: Modified Test data vs Ideal data

List of Figures

List of Tables: